POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Re-BERT OQA : Un système de question-réponse dans le domaine ouvert

DYLAN FARVACQUE
Département de génie informatique et génie logiciel

Mémoire présenté en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
Génie informatique

Août 2020

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Ce mémoire intitulé :

Re-BERT OQA : Un système de question-réponse dans le domaine ouvert

présenté par **Dylan FARVACQUE**
en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
a été dûment accepté par le jury d'examen constitué de :

Guillaume-Alexandre BILODEAU, président
Amal ZOUAQ, membre et directrice de recherche
Michel GAGNON, membre et codirecteur de recherche
Gilles PESANT, membre

DÉDICACE

À Sid-Ahmed ...

REMERCIEMENTS

Merci à mes directeurs, Amal Zouaq et Michel Gagnon, de m'avoir accompagné dans ce long voyage qui, parfois, pouvait sembler sans fin. Un merci tout particulier à Michel, de m'avoir donné l'envie et la confiance d'entreprendre ce voyage. Et bien sûr, merci Amal pour ta rigueur et tes **quelques** commentaire**S** sur mon travail. Vous êtes allés chercher le meilleur de moi et je ne sais toujours pas comment vous l'avez fait !

Merci à mes parents, sans qui je ne serais là aujourd'hui. Merci d'avoir été assez fous pour aller jusqu'à Montréal et merci de m'avoir transmis cette folie. Votre support, votre confiance, votre amour pour moi a été et sera toujours mon carburant pour continuer à avancer.

Merci à mon frère, Steve, qui a toujours été là pour moi et à qui je voudrais ressembler quand je serai grand !

Merci à tous mes amis et particulièrement Jo, Émilie & Mathieu, je n'ai malheureusement pas assez de place pour vous donner toutes les raisons de ce remerciement, le fait est : j'ai enfin fini !

Finalement, merci Sandra. Pour tes encouragements sans faille, et pour m'avoir poussé à continuer dans les moments les plus difficiles.

Merci à la vie.

RÉSUMÉ

Dans le présent mémoire, nous abordons la tâche de *question-réponse dans le domaine ouvert*, c'est-à-dire la tâche qui a pour but de répondre à une question en utilisant son corpus de connaissances (qu'il soit structuré ou non) comme seule ressource. Plus spécifiquement notre but est de proposer un système de *question-réponse dans le domaine ouvert* capable de répondre à des questions factuelles en utilisant Wikipédia comme corpus de connaissances. En général, ce genre de système se divise en deux modules. Le premier, responsable de la recherche d'information, permet de trouver des documents pertinents dans le corpus de connaissance. Le second, le module d'extraction de réponse, a pour objectif d'extraire des candidats de réponse provenant des documents précédemment sélectionnés puis de déterminer une réponse finale parmi les candidats. Dans les dernières années, les avancées dans le domaine de la compréhension de lecture automatique ont été une grande source d'inspiration pour le module d'extraction résultant en la création des systèmes de *question-réponse dans le domaine ouvert* les plus efficaces à ce jour. Cependant, il apparaît assez clairement que la transition de la compréhension de lecture automatique à la tâche de *question-réponse dans le domaine ouvert* introduit aussi une baisse de performances. Ainsi, notre but est de proposer une architecture de système de *question-réponse dans le domaine ouvert* qui permet d'améliorer l'intégration des approches provenant de la compréhension de lecture automatique afin de réduire cette baisse de performances. Pour ce faire, nous proposons de travailler avec un corpus Wikipédia indexé par des documents de taille semblable à ceux que l'on retrouve dans le domaine de compréhension de lecture automatique. De plus nous ajoutons un module après le module de recherche d'information qui effectue un ordonnancement des documents du module précédent en utilisant RoBERTa, un des meilleurs modèles de langue à ce jour. Grâce à ce module, nous réduisons le nombre de documents que le module d'extraction de réponse doit traiter en ne gardant que les documents les plus pertinents. Ainsi, nous facilitons l'intégration du dernier module à notre système. Nos résultats montrent que nous obtenons de bonnes performances avec l'approche proposée par notre système, Re-BERT OQA.

ABSTRACT

In this thesis, we tackle the *Open Domain Question-Answering* task, where the goal is to be able to answer a question using a knowledge source (either structured like DBpedia or unstructured such as Wikipedia). Specifically, our goal is to propose an open domain question-answering system capable of answering factoid questions using Wikipedia as knowledge source. In general, these types of systems are divided in two sub-modules. The first one, responsible of the information retrieval step, enables the system to find relevant documents in its knowledge source. The second, the answer extraction module, extracts answer candidates from the previously selected documents and then determines the final answer within the candidates. In recent years, the progress achieved in the machine reading comprehension field has driven the development of improved answer extraction modules resulting in the creation of the best open domain question answering systems to date. However, it is quite clear that the transition from the machine reading comprehension task to the open domain question answering task leads to a decrease of performance. Therefore, our goal is to propose an open domain question-answering architecture that better integrates the approaches coming from the machine reading comprehension field in order to reduce the loss of performance. To achieve our goal, we propose to work with a Wikipedia dump indexed by documents of length comparable to the ones we encounter in the machine reading comprehension field. Furthermore, we add a ranking module after the information retrieval step to perform a ranking of the documents using RoBERTa, one of the best language models to date. This allows us to reduce the number of documents that the answer extraction module has to process by keeping only the most relevant ones. Our results show that our system, Re-BERT OQA, is one of the best performing systems on the open domain question-answering task.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES ET ABRÉVIATIONS

ADAM	ADAptive Moment estimation
ALBERT	A Lite BERT
BERT	Bidirectional Encoder Representations from Transformers
BRNN	Bidirectional recurrent neural networks
DS	Distantly Supervised
EM	Exact Match
GLUE	General Language Understanding Evaluation
LSTM	Long Short-Term Memory
MRC	Machine Reading Comprehension
NLI	Natural Language Inference
PQNLI	Passage QNLI
PSQuAD	Positive SQuAD
QNLI	Question-answering NLI
RNN	Recurrent Neural Networks
RoBERTa	Robustly optimized BERT approach
TALN	Traitement Automatique du Langage Naturel
TF-IDF	Term Frequency–Inverse Document Frequency
SQuAD	Stanford Question Answering Dataset

CHAPITRE 1 INTRODUCTION

> “ Je ne cherche pas à connaître les réponses, je cherche à comprendre les questions. ”
>
> *Confucius*

La quête de connaissances est profondément humaine et c'est tout naturellement que pratiquement aussitôt que l'ordinateur fut inventé, l'homme a voulu être capable de lui poser des questions, et que l'ordinateur lui réponde. En effet, dès les années 60, on voit l'apparition de systèmes tels que BASEBALL [1] et LUNAR [2] qui, à l'instar de Confucius, étaient respectivement capable de comprendre et de répondre à des questions sur le baseball et sur les faits scientifiques de la mission Apollo. Ces systèmes de question-réponse dans un domaine spécifique utilisaient chacun un des deux paradigmes majeurs pour cette tâche à savoir les systèmes basés sur la recherche d'information et ceux basés sur les bases de connaissances.

Les systèmes de question-réponse basés sur les bases de connaissances sont des systèmes qui construisent une représentation sémantique de la question en la faisant correspondre à une représentation logique, par exemple :

> Question : When was Ada Lovelace born ?
> Représentation : birth-year (Ada Lovelace, ?x)

Cette représentation est ensuite utilisée pour interroger une base de connaissances comme DBpedia [3]. Ce genre de base de connaissances représente les informations sous forme de graphes qui sont ensuite interrogés avec des requêtes afin d'obtenir une information sur un sujet.

Le second paradigme, l'objet de ce mémoire, est constitué des systèmes de question-réponse basés sur la recherche d'information. Ce genre de système se base sur l'énorme quantité d'information textuelle disponible sur le web ou dans des corpus comme PubMed ou Wikipédia. À partir de la question utilisée comme entrée, le système utilise des techniques de recherche d'information afin de trouver des documents au sein de leur corpus. Par la suite, ce genre de système utilise des algorithmes de compréhension en lecture automatique (*Machine Reading Comprehension*, MRC) afin d'extraire la réponse à la question [4]. On retrouve donc très souvent un pipeline commençant par la recherche d'information et suivi d'un système de compréhension de lecture automatique [5–8].

Ainsi, quel que soit le paradigme utilisé, un tel système a pour but de répondre à une question factuelle à partir de sa source d'informations. Étant donné que la tâche de question-réponse

est vaste, plusieurs sous-tâches ont été créées, que ce soit en restreignant la tâche à une tâche de compréhension en lecture automatique [9] où l'on donne alors un document dans lequel il faut extraire la réponse à la question, ou bien en spécifiant ou non le domaine des questions posées.

1.1 Définitions et concepts de base

Comme mentionné plus tôt, la tâche de question-réponse est vaste. Ainsi, elle peut être divisée en plusieurs sous-tâches que nous définissons de la façon suivante.

Compréhension en lecture automatique

Dans le domaine de question-réponse, il existe la sous-tâche de Compréhension en lecture automatique [10]. Cette tâche, que nous appelons aussi la tâche de question-réponse fermée est en fait la seconde partie d'un système de question-réponse, à savoir, la tâche d'extraire la réponse dans un document. Ce sous-domaine est très dynamique et a récemment évolué, notamment avec la mise au point de jeux de données de grande taille comme SQuAD [9,11] qui permettent d'employer des approches neuronales pour résoudre ce problème [9]. Dans ce genre de jeu de données, les documents ont la taille d'un passage de plusieurs phrases voire quelques paragraphes. Assez souvent, les avancées faites dans le domaine de question-réponse fermée servent de source d'inspiration pour améliorer les systèmes de question-réponse complets ou ouverts.

Systèmes de question réponse ouverts ou spécifiques à un domaine

Bien que les premiers systèmes développés pour la tâche de question-réponse étaient spécifiques à un domaine, les recherches récentes tendent de plus en plus vers le développement de systèmes pour la tâche de *question-réponse dans le domaine ouvert*. C'est-à-dire des systèmes dont l'objectif est, étant donné une question, d'y répondre quel que soit le domaine de la question (médecine, informatique, culture générale ...). Ce genre de système est aussi libre d'utiliser le corpus de leur choix, le plus populaire étant Wikipédia.

Comme on peut s'en douter, le développement d'un système pour la tâche de *question-réponse dans le domaine ouvert* implique bien plus de défis. En effet, quel que soit le paradigme employé, un tel système doit utiliser une source d'information (que ce soit un corpus ou une base de connaissance) qui est la plus exhaustive possible. Ainsi, un système de question-réponse basé sur la recherche d'information doit être capable, à partir d'une question, de trouver l'information importante afin de pouvoir rechercher le ou les bons documents et être capable

d'extraire la réponse à la question. Ici, la définition d'un document peut varier suivant le système. On utilise parfois des documents de la taille d'une page Wikipédia, donc de plusieurs paragraphes ou bien de plusieurs phrases ou encore d'une centaine de mots.

1.2 Problématique

Dans ce mémoire, comme mentionné plus haut, nous nous concentrons sur les systèmes de question-réponse basés sur la recherche d'information. Plus spécifiquement, nous nous intéressons à ces systèmes dans la tâche de *question-réponse dans le domaine ouvert*. Cette tâche est souvent divisée en deux parties, la première étant la recherche d'information où le système doit être capable de trouver un ou plusieurs documents pertinents pour la question au sein de son corpus. La seconde tâche est d'extraire la réponse à la question dans l'un des documents trouvés à l'étape précédente [4].

Récemment, la tâche de *question-réponse fermée* a reçu beaucoup d'attention grâce à l'émergence de nouveaux jeux de données comme SQuAD [9, 11]. Ainsi, les approches d'extraction de réponse se basant sur l'auto-attention [12], utilisées dans les modèles de langues pré-entraînés comme BERT [13] obtiennent de meilleurs résultats que l'homme sur la tâche de *question-réponse fermée*.

À la lumière de ces avancées, beaucoup de recherches se sont confrontées à l'objectif de réutiliser ce genre d'approche pour la tâche de *question-réponse dans le domaine ouvert*. Cependant, ils se sont heurtés à de nombreux problèmes, à savoir la difficulté qu'ont ces approches à évoluer vers de grands corpus [14]. En effet, ces approches ont été développées pour extraire une réponse dans un document fourni avec la question, or dans la tâche de *question-réponse dans le domaine ouvert*, plusieurs documents (et parfois de plus grandes tailles que ceux rencontrés dans la tâche de *question-réponse fermée*) sont associés à une même question. De plus, ces approches ont tendance à se concentrer sur la mauvaise partie d'un document lorsque celui-ci est trop long ou trompeur (par exemple, dans le cas d'un document qui possède un vocabulaire proche de celui de la question) [15].

Dans ce mémoire, nous nous attaquons à ce problème de transition entre la tâche de *question-réponse fermée* et de *question-réponse dans le domaine ouvert*. Ainsi, nous allons utiliser l'approche de l'état de l'art, à savoir une architecture basée sur le modèle de langue BERT comme module d'extraction de réponse. Afin de répondre aux problèmes mentionnés plus haut et pour mieux intégrer le module d'extraction de réponse dans un pipeline d'un système de *question-réponse dans le domaine ouvert*, nous allons travailler uniquement avec des documents de taille similaire à ceux utilisés dans des architectures se basant sur les modèles

de langues comme BERT. Par ailleurs, nous proposons essentiellement d'ajouter une tâche d'ordonnancement entre la recherche d'information et l'extraction de réponse. Cette tâche d'ordonnancement a pour but d'identifier parmi les documents trouvés lors de la recherche d'information, les documents les plus pertinents qui vont être utilisés pour l'extraction de réponse. Cet ordonnancement est basé sur la tâche de classification de paires de phrases. Dans notre contexte, cette tâche a pour but de déterminer, à partir d'une paire de phrases question-document, si le document contient ou non la réponse à la question.

Tout au long de ce mémoire, nous visons à répondre à la question suivante :

> Dans le contexte de *question-réponse dans le domaine ouvert*, pouvons-nous trouver de meilleurs documents pour la tâche d'extraction de réponse en utilisant un ordonnancement basé sur la classification de paires de phrases ?

1.3 Plan

Ce mémoire est divisé en 6 chapitres. Le chapitre 2 présente en détail le contexte de notre recherche, à savoir, les différentes techniques et mécanismes utilisés dans le domaine du TALN ainsi qu'une revue littéraire de l'état de l'art des systèmes de *question-réponse dans le domaine ouvert*. Le chapitre 3 décrit DrQA [5], le système sur lequel nous nous basons afin de développer notre système de *question-réponse dans le domaine ouvert*. Nous y exposons aussi ses limites ainsi que ses performances. Par la suite, dans le chapitre 4, nous présentons les approches et modules utilisés dans le système que nous avons développé, Re-BERT OQA, dans le but de répondre à notre question de recherche. Cette présentation est suivie du chapitre 5 qui discute des résultats de nos différents modules individuellement puis de ceux de notre système complet. Finalement, le chapitre 6 résume notre travail, en présente les limites ainsi que les potentielles améliorations futures.

CHAPITRE 2 CONTEXTE & REVUE DE LITTÉRATURE

Le traitement de langage naturel automatique (TALN ou *NLP* en Anglais) a beaucoup évolué au cours des dernières années. Cette évolution a été rythmée, d'une part, par l'amélioration de la représentation des mots où nous sommes passés de la représentation de mots par des vecteurs clairsemés aux représentations par vecteurs denses et d'autre part, par l'émergence de réseaux de neurones comme des LSTM [16] ou des Transformateurs [12]. Ces réseaux de neurones ont pour but d'encoder le contexte en utilisant tous les jetons (*tokens*) de la séquence d'entrée et ont contribué à de grandes avancées dans le domaine du TALN.

Dans les sections suivantes, nous présenterons tout d'abord les différents procédés afin d'obtenir une représentation d'une séquence de jetons sous forme de vecteurs. Par la suite, nous détaillerons les différentes architectures desquelles nous nous inspirons ainsi que celles que nous utilisons dans nos expériences. De plus, nous expliquerons comment les différents types de modèles de langue fonctionnent afin d'obtenir une meilleure représentation d'un mot ou d'une séquence de jetons. Finalement, nous parlerons des trois principales tâches qui permettent d'obtenir un système de *question-réponse dans le domaine ouvert*.

2.1 Représentation d'une séquence

Afin d'utiliser les différents modèles présentés plus tôt, il faut être capable de transformer une séquence en un vecteur la représentant.

2.1.1 Vecteur clairsemé

L'approche la plus naïve pour représenter une séquence est le modèle vectoriel (*Vector space model*) [17] et plus précisément les vecteurs dits clairsemés. C'est une méthode algébrique de représentation d'un document. Cette méthode est encore utilisée de nos jours dans la tâche de recherche documentaire, c'est-à-dire la tâche de trouver des documents pertinents pour une requête (souvent une question). L'ensemble de représentation des documents est un vocabulaire comprenant des termes d'indexation. Ceux-ci sont typiquement les mots les plus significatifs du corpus considéré : le corpus est alors filtré pour retirer les mots vides (*stop words*) [18] et les jetons restants sont normalisés soit par racination [19, 20] qui cherche la plus petite racine commune d'un jeton ou par lemmatisation [21] qui cherche la forme canonique d'un mot (par exemple un verbe à l'infinitif) [18]. Finalement, le vocabulaire peut être un ensemble de n-grammes, c'est-à-dire une sous-séquence de n éléments construite à partir d'un

corpus filtré ou non. Chaque entrée du vocabulaire est un ou plusieurs mots (au maximum n mots) qui se suivent [22]. À chaque élément du vocabulaire est associé un index unique arbitraire.

Chaque contenu est ainsi représenté par un vecteur v, dont la dimension correspond à la taille du vocabulaire. Chaque élément v_i du vecteur v consiste en un poids associé au terme d'indice i et à l'échantillon de texte. Chaque document est donc représenté par un vecteur clairsemé (*sparse*). Un exemple simple est d'identifier v_i au nombre d'occurrences du terme i dans l'échantillon de texte. La composante du vecteur représente donc le poids du mot i dans le document. L'un des schémas de pondération les plus usités est le TF-IDF (*term frequency-inverse document frequency*). Cette mesure statistique permet d'évaluer l'importance d'un terme contenu dans un document, relativement à une collection ou un corpus [23]. Le poids augmente proportionnellement au nombre d'occurrences du mot dans le document. Il varie également en fonction de la fréquence du mot dans le corpus.

Grâce à cette représentation, on peut introduire une notion d'espace vectoriel sur l'espace des documents en langage naturel. On en arrive à la notion mathématique de proximité entre documents. Étant donné que chaque document est un vecteur, nous sommes capables de calculer sa similarité avec un autre document, ou une requête qui serait représentée dans le même espace vectoriel avec des mesures telles que la similarité cosinus.

Ce modèle, bien que simple, reste tout de même efficace en autant que la qualité de la représentation des documents soit bonne. Cependant, ces modèles restent limités. En effet, ils reposent sur l'hypothèse que tous les termes représentatifs sont indépendants et que donc l'ordre des mots n'est pas pris en compte. Aussi, ces modèles sont incapables de prendre en compte des synonymes qui seront donc considérés comme deux termes indépendants.

2.1.2 Vecteur dense

Les vecteurs denses, et plus particulièrement les plongements de mots, permettent de pallier les lacunes des vecteurs clairsemés. Un plongement de mots ou plongement lexical est une représentation de tout mot par un vecteur dense de nombres réels. Grâce à celle-ci, les mots présents dans des contextes similaires ont une représentation relativement proche dans l'espace vectoriel. L'idée derrière cette technique repose sur l'hypothèse de Harris [24] qui veut que les mots apparaissant dans des contextes similaires aient des significations apparentées. Ainsi, en utilisant le voisinage d'un mot pour représenter ce dernier, on crée une représentation qui peut encoder son sens.

Ainsi, on introduit aussi le concept de similarité dans la représentation des mots, c'est-à-dire

que des mots similaires apparaissent dans des contextes similaires et seront donc représentés de façon similaire.

La représentation de mots par vecteur dense nécessite un modèle qui a préalablement appris à représenter des mots en traitant de très large corpus. Les modèles les plus utilisés pour générer ces vecteurs de mots sont GloVe [25], word2Vec [26] et FastText [27].

Spécifiquement, GloVe est un modèle compteur (*count-based model*) qui apprend à représenter un mot en un vecteur en faisant une réduction de dimensionnalité sur la matrice comptant les cooccurrences [28]. Premièrement, ils construisent une très grande matrice de mots par rapport à leur contexte X. Cette matrice compte pour chaque mot (qui sont les lignes de la matrice) le nombre de fois qu'ils apparaissent dans un certain contexte (une colonne de la matrice).

GloVe apprend à représenter la relation entre un mot et son contexte, car Pennington & al. ont déterminé que les relations entre des mots sont révélées par le ratio de probabilité de cooccurrence avec différents mots. Le modèle GloVE a donc pour but d'apprendre ce ratio plutôt que les probabilités elles-mêmes.

Spécifiquement, GloVe prédit les mots aux alentours de façon à maximiser la probabilité qu'un mot du contexte apparaisse sachant un mot central en calculant une régression logistique dynamique. Ainsi, à partir de la matrice X, GloVe apprend à construire les vecteurs $\vec{w}$ de mots de sorte que le produit scalaire des vecteurs de deux mots i et j soit égal à leur probabilité de co-occurrent :

$$\vec{w_i}^T \vec{w}_j + b_i + b_j = \log X_{ij}$$

Avec b_i et b_j des biais associés aux mots i et j respectivement et X_{ij} représente à quelle fréquence le mot i apparaît dans le contexte j.

Pour ce faire, GloVe cherche à minimiser la somme pondérée de toutes les erreurs au carré pondéré par la fonction f :

$$\text{Loss} = \sum_{i,j=1}^{V} f(X_{ij})(\vec{w_i}^T \vec{w}_j + b_i + b_j - \log X_{ij})^2$$

Avec V la taille du vocabulaire.

Pour ce qui est de Word2Vec, c'est un modèle prédictif (*predictive model*) [28]. Il se décline en deux architectures : le modèle de sac de mots continus (*continuous bag of words*, CBOW) et celui de skip-gram [26]. Le CBOW vise à prédire un mot étant donné son contexte, c'est-à-dire

un nombre de mots avant et après le mot à prédire et est plus rapide à apprendre, mais donne de moins bons résultats [29] par rapport au modèle skip-gram qui vise à prédire les mots du contexte étant donné un mot en entrée.

Dans les deux cas, word2vec se base sur un réseau de neurones à deux couches. La couche cachée contient quelques centaines de neurones et constitue, à l'issue de la représentation, le plongement lexical permettant de représenter un mot en un vecteur dense. La couche de sortie permet d'implémenter une tâche de classification au moyen d'une softmax.

L'apprentissage du modèle word2vec est non supervisé et donc ne nécessite aucun label. Les données et plus particulièrement la proximité des mots au sein du corpus d'entraînement permettent au modèle d'apprendre la représentation des mots en vecteur dense.

Dans les deux cas, le réseau a pour but de maximiser la moyenne des log-probabilités. La seule différence étant que nous avons $P(w_{t+j}|w)$ (c'est-à-dire la probabilité du contexte sachant le mot central) pour le modèle skip-gram et $P(w|wt+j$ (c'est à dire la probabilité du mot central sachant le contexte) pour le modèle CBOW. Le modèle word2vec est résumé à la Figure 2.1.

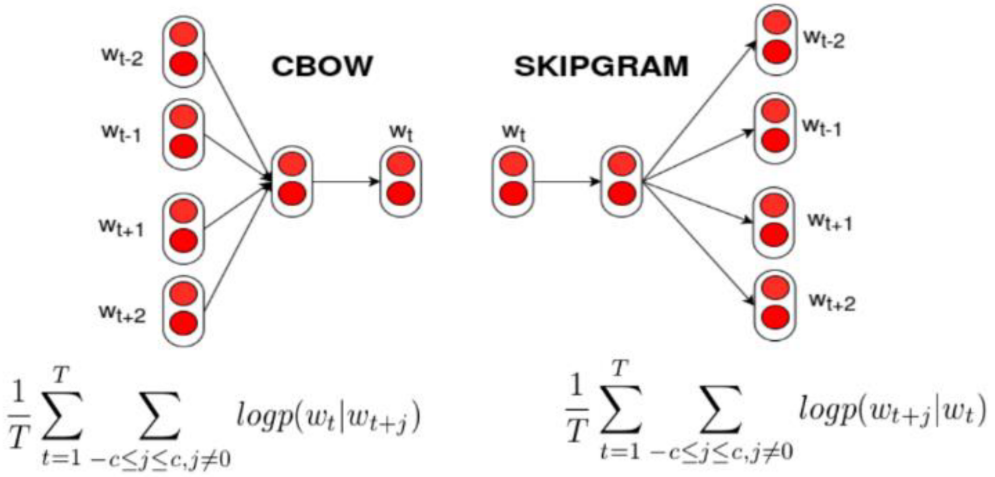


Figure 2.1 Illustration du modèle word2vec [1]

Finalement, FastText est une extension du modèle Word2Vec qui, au lieu de traiter chaque mot dans un corpus comme la plus petite unité, considère chaque mot comme une composition de n-grammes caractères (par exemple, le mot "pomme" est une composition de "po" "pom" "pomm", "om" "omm" "omme" "mm" "mme" "me" pour un modèle FastText utilisant des bi-gramme comme minimum et 4-grammes comme maximum), le vecteur représentant le mot est donc la somme des représentations de chaque n-gramme caractères qui le compose.

1. Tiré de `https://fr.wikipedia.org/wiki/Word2vec`

Les représentations compactes des vecteurs denses comparées aux vecteurs clairsemés permettent d'éviter la malédiction de la dimension (*curse of dimensionality*) [30] qui survient en intelligence artificielle lorsque l'on cherche à analyser ou organiser des données dans des espaces de grande dimension.

La représentation par vecteur dense permet aussi d'incorporer les notions de mots synonymes, et de mots similaires qui manquait dans la représentation par vecteur clairsemé.

En somme, la représentation de séquence peut se faire soit à l'aide d'un vecteur clairsemé ou bien à l'aide de vecteurs denses notamment à l'aide de modèles comme GloVe où Word2Vec. Dans le cas de représentation par vecteur dense, diverses techniques existent pour combiner les vecteurs de chaque mot. En effet, ils peuvent être combiné en en faisant la moyenne de chaque vecteur [31], où encore la moyenne pondérée par un TF-IDF [32]. Il existe aussi des techniques utilisant des réseaux de neurones récurrents (ce genre de réseau est décrit dans la prochaine section) qui apprennent à combiner les vecteurs de mots dans le but d'accomplir une tâche spécifique [33–35]

Ainsi, il est possible de projeter une séquence de jetons dans un espace vectoriel ce qui nous permet d'utiliser différents réseaux de neurones afin d'accomplir différentes tâches du TALN.

2.2 Réseau de neurones récurrents

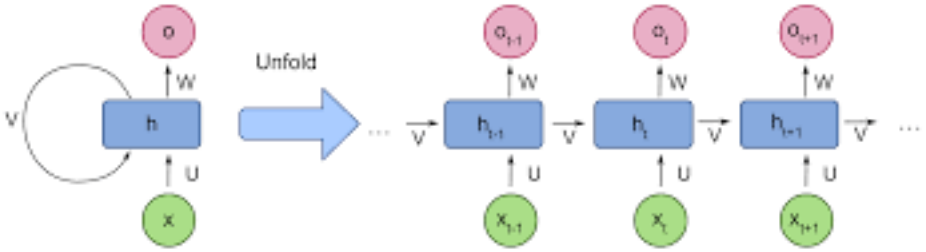


Figure 2.2 Schéma d'un RNN simple [2]

Un des premiers réseaux qui ont contribué à l'essor récent du TALN est appelé un réseau de neurones récurrents (*recurrent neural network*, RNN). Un RNN est un réseau de neurones qui contient un cycle. C'est-à-dire un réseau de neurones où la valeur d'un neurone dépend directement ou indirectement d'une valeur de sortie d'un neurone précédent comme l'illustre

2. Tiré de `https://commons.wikimedia.org/wiki/Artificial_neural_network`

la Figure 2.2 d'un RNN simple aussi appelé *Elman network* [36]. De façon similaire à un réseau de neurones classique, nous calculons la sortie o de la façon suivante :

$$h_t = g(Vh_{t-1} + Ux_t)$$
$$o_t = f(Wh_t)$$

où U, V et W sont des matrices de poids apprises et partagées à travers le temps, h est une couche cachée (*hidden layer)* calculée à chaque étape de temps, tout comme o. Dans le traitement du langage naturel, l'unité de temps est en fait un jeton de la séquence d'entrée et on avance dans le temps au fur et à mesure que le modèle lit la séquence d'entrée. La couche cachée permet une forme de mémoire, ou de contexte, qui encode l'information précédente afin que le réseau puisse prendre une décision avec plus d'informations. Aussi, cette forme de récurrence permet aux dernières couches cachées d'avoir de l'information sur toute la séquence que le réseau vient de traiter. Toutefois, plus on avance dans le temps, plus l'information des premières couches cachées est diluée par rapport à celles plus récentes. Finalement, g et f sont des fonctions d'activation. Dans le cas de classification, la fonction f est en fait un softmax qui permet d'obtenir une distribution de probabilité normalisée sur toutes les classes possibles.

2.3 Réseau de neurones récurrents à mémoire court-terme et long-terme

Les réseaux de neurones récurrents à mémoire court terme et long terme (*Long Short-Term Memory*, LSTM) [16, 37] est une autre forme de RNN. Ils ont été créés pour pallier le fait que les RNN donnent plus d'importance au contexte proche plutôt qu'au contexte pertinent. Les LSTMs sont en fait une forme de RNN, mais plus sophistiquée. L'intuition derrière ces modèles est de résoudre le problème de gestion de contexte en le divisant en deux sous-problèmes : a) enlever l'information qui n'est plus nécessaire du contexte et b) ajouter l'information qui est nécessaire pour une prise de décision ultérieure. Et c'est le rôle du LSTM d'apprendre comment gérer le contexte. Pour ce faire, les LSTM ajoutent une couche de contexte en plus de la couche cachée. Celle-ci est contrôlée par trois différentes portes qui permettent de maîtriser l'information qui est retenue. La porte de remise à zéro (*forget gate*), F_t, permet au réseau de supprimer l'information du contexte qui n'est plus nécessaire. Cette porte est calculée avec la somme pondérée de l'état caché précédent et l'entrée courante et passe à travers un sigmoïde. Ce masque est ensuite multiplié terme par terme avec la couche de contexte de

l'état précédent c_{t-1} afin d'enlever l'information qui n'est plus nécessaire.

$$F_t = \sigma(U_F ht - 1 + W_F x_t + b_F)$$
$$k_t = c_{t-1} \odot F_t$$

Avec h_{t-1} l'état caché précédent et x_t l'entrée courante. Ensuite, il faut calculer l'information à extraire du précédent état caché et de l'entrée courante grâce à la porte d'entrée (*input gate*), I_t. Pour ce faire, il faut tout d'abord calculer le masque de la porte, qui est aussi un sigmoïde de la somme pondérée de l'état caché et de l'entrée courante, laquelle est ensuite multipliée terme à terme avec l'information provenant de l'état caché et de l'état courant passé à travers une fonction d'activation (couramment on utilise une tangente hyperbolique) :

$$I_t = \sigma(U_i h_{t-1} + W_i x_t + b_I)$$
$$g_t = \tanh(U_c h_{t_1} + W_c x_t + b_c)$$
$$j_t = g_t \odot I_t$$

Afin d'actualiser le contexte, il faut additionner les deux étapes précédentes :

$$c_t = j_t + k_t$$

Finalement, la dernière porte, la porte de sortie (*output gate*) décide quelle information est requise par l'état caché courant :

$$O_t = \sigma(U_o h_{t-1} + W_o x_t + b_o)$$
$$h_t = O_t \odot \tanh(c_t)$$

Tout ce mécanisme est résumé dans la Figure 2.3. À noter que toutes les matrices W, U ainsi que b sont des matrices apprises par le modèle LSTM.

Les LSTM se déclinent aussi en réseaux bidirectionnels (Bi-LSTM) soit un réseau qui connecte deux couches cachées qui vont dans des directions opposées à la même sortie. Ainsi la couche de sortie peut avoir de l'information sur les états passés (de gauche à droite) et sur le futur (de droite à gauche) simultanément [38]. Cette architecture permet d'augmenter l'information sur l'entrée disponible au modèle et permet de combler une des limites des RNNs qui n'ont accès qu'à l'information disponible au temps t. Au contraire, avec un BRNN, on peut avoir accès autant à l'information sur le passé que sur le futur.

4. Tiré de `https://commons.wikimedia.org/wiki/Artificial_neural_network`

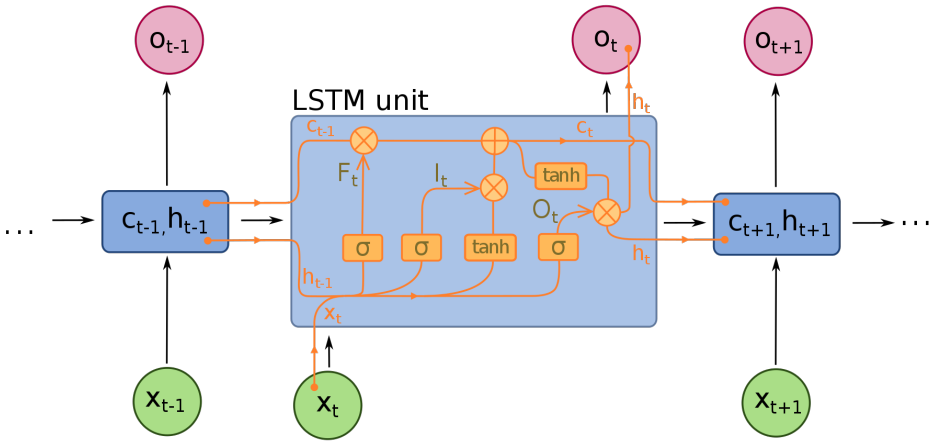


Figure 2.3 Schéma d'un neurone d'un modèle LSTM avec les opérations effectuées à l'étape t. les poids ne sont pas indiqués afin d'alléger le schéma [4]

En somme, les LSTM sont des RNN qui sont mieux armés pour le traitement des séquences, car ils gardent en mémoire du traitement des mots vus précédemment et intègrent un mécanisme de contrôle de l'information importante grâce à l'utilisation de portes pour générer un vecteur de contexte au temps t et l'utilisent pour générer son état caché. Bien que ce mécanisme permette d'améliorer la conservation d'informations importantes jusqu'à la fin de la séquence, il ne règle pas tout à fait le problème des RNN. Ainsi, tout comme les RNN plus simples, ce genre de modèle est obligé de traiter l'information séquentiellement, ce qui implique un temps de traitement relativement long et une difficulté à bien traiter les plus longues séquences. Ce sont à ces limites que les transformateurs s'attaquent.

2.4 Transformateurs

Outre les limites présentées plus tôt pour les réseaux récurrents, la motivation des Transformateurs (*Transformers*) [12, 39] vient spécifiquement de la tâche de la traduction. En effet, pour ce genre de tâche, l'architecture se basant sur les LSTM consiste en deux parties. Un encodeur, qui est un réseau LSTM qui va lire l'entrée et produire un encodage de celle-ci. Un décodeur qui va produire une sortie à partir de l'encodeur qui est la traduction de l'entrée initiale [40]. Cela présente certaines limites, entre autres que le dernier état de l'entrée représente la totalité du texte d'entrée et que ce genre de réseau gère assez difficilement les longues phrases.

Pour pallier ces limites, les LSTM ont été enrichis du mécanisme d'attention [41,42]. L'attention est en fait un lien entre les couches cachées de l'encodeur et le décodeur. Ainsi le décodeur peut construire la traduction en utilisant la totalité des états intermédiaires de l'encodeur. Spécifiquement, au lieu d'utiliser seulement le dernier état caché h_n, le décodeur utilise une

combinaison linéaire de tous les états $\sum_{i=1}^{n} a_i h_i$ où a est le vecteur unitaire d'attention qui est calculé à chaque étape de la production de la sortie. Bien que ce mécanisme améliore les performances des LSTM, il n'en reste pas moins que ces modèles sont fastidieux à entraîner de par le fait qu'ils sont lents à l'entraînement, que la rétro-propagation du gradient doit se faire sur toute la séquence et surtout, du fait qu'il est très difficile de paralléliser le traitement qui doit être séquentiel (vu que ce sont des RNNs).

Les Transformateurs [12] parent à toutes ces lacunes. L'idée derrière le transformateur est que le mécanisme d'attention permet déjà de relier des séquences de mots, et que donc on peut se passer du mécanisme de récurrence et n'utiliser que l'attention.

Dans les faits, le Transformateur est composé de deux composantes principales : un ensemble d'encodeurs suivi d'un ensemble de décodeurs [12]. Chaque encodeur a pour fonction de traiter le vecteur d'entrée pour générer un encodage qui contient l'information sur des parties des entrées qui sont pertinentes entre elles. Chaque encodeur donne à son prochain encodeur les encodages générés comme entrée. Chaque décodeur fait l'inverse, il prend en entrée les encodages et le traite en utilisant l'information contextuelle pour générer une séquence de sortie. Pour y arriver, chaque encodeur et décodeur utilise le mécanisme d'attention, qui pour chaque entrée, pondère la pertinence de toutes les entrées et garde l'information en conséquence quand la sortie est produite. On appelle ce mécanisme de l'auto-attention. Aussi, chaque décodeur a un second mécanisme d'attention, plus traditionnel, qui prend de l'information à partir de la sortie des décodeurs précédents. Les décodeurs utilisent cette information avant d'utiliser l'information de l'auto-attention. Finalement, les encodeurs et les décodeurs se terminent par un réseau de neurones à propagation avant pour des traitements additionnels de la sortie.

L'architecture que nous allons présenter est résumée dans la Figure 2.4

2.4.1 Produit scalaire de l'attention mis à l'échelle

Le concept de base dans l'architecture des Transformateurs est l'unité produit scalaire de l'attention mise à l'échelle (*Scaled dot-product attention*). Une illustration de l'attention dans les transformateurs est présentée à la Figure 2.5. Lorsqu'une séquence passe à travers un modèle de Transformateur, les poids d'attention sont calculés entre chaque jeton de la séquence simultanément. La couche d'attention produit ainsi un encodage pour chaque jeton dans le contexte qui contient de l'information sur lui-même, mais aussi une combinaison pondérée de tous les autres jetons pertinents qui sont mis à l'échelle selon leur poids d'attention.

Concrètement, pour chaque unité d'attention dans un modèle Transformateur, le modèle

apprend trois matrices de poids : les poids de requête (*Query*) W_q, les poids de clé (*Key*) W_k, et les poids de valeur (*Value*) W_v. Pour chaque jeton i, le plongement du mot x_i est multiplié avec chacune des matrices de poids pour produire un vecteur de requête q, un vecteur de clé k et un vecteur de valeur v :

$$q_i = x_i W_q$$
$$k_i = x_i W_k$$
$$v_i = x_i W_v$$

Le poids d'attention est calculé en utilisant le vecteur de requête et le vecteur de clé : le poids d'attention a_{ij} du jeton i sur le jeton j est obtenu en calculant le produit scalaire entre q_i et

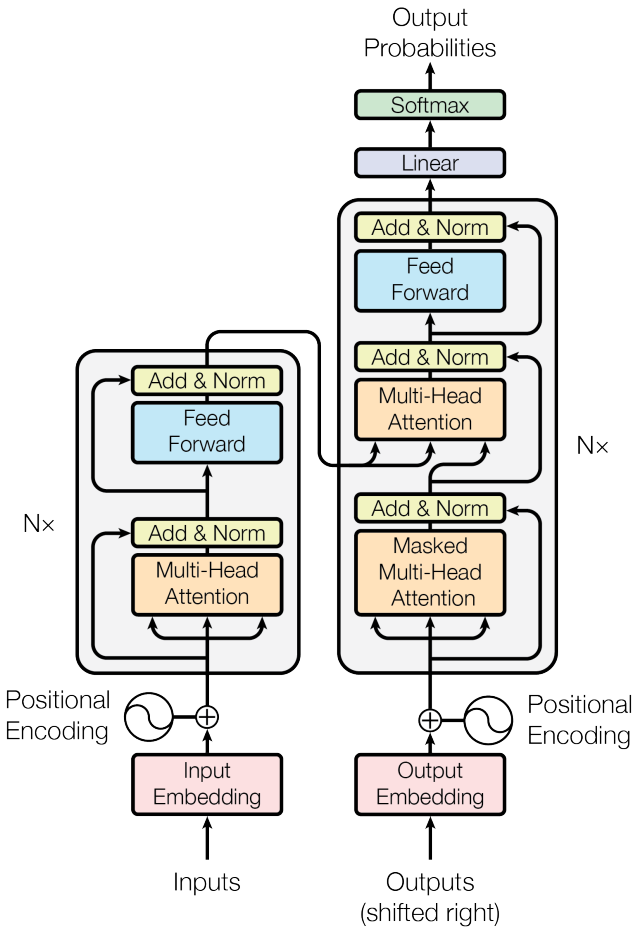
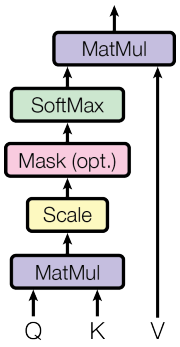


Figure 2.4 Schéma d'un Transformateur tiré de [12]

Produit scalaire de l'attention mis à l'échelle



Attention multi-tête

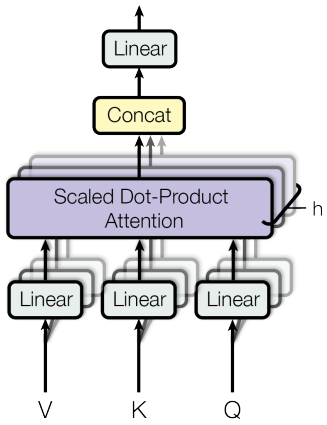


Figure 2.5 (Gauche) Produit scalaire de l'attention mis à l'échelle. (Droite) L'attention multi-têtes comporte plusieurs couches d'attention en parallèle. Tous deux tirés de [12]

k_j. Les poids d'attention sont ensuite divisés par la racine carrée de la dimension du vecteur de clé $\sqrt{d_k}$, afin de stabiliser le gradient durant l'entraînement. Le tout passe finalement à travers un softmax pour normaliser les poids pour que leurs sommes soient de 1. Le fait que les matrices W_q et W_k soient différentes permet que l'attention soit non-symétrique : si le jeton i est important pour le jeton j (donc $q_i \cdot k_j$ est grand) cela ne veut pas forcément dire que le jeton j est important pour i (donc $q_j \cdot k_i$ est grand). La sortie de l'unité d'attention pour le jeton i est la somme pondérée des vecteurs de valeur de tous les jetons, pondérée par a_{ij}, l'attention du jeton i sur chaque jeton.

Le calcul de l'attention pour tous les jetons peut être exprimé en tant qu'une seule matrice de calcul. Les matrices Q, K et V sont définies comme les matrices où les i^e lignes sont respectivement les vecteurs q_i, k_i et v_i :

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

2.4.2 Attention multi-têtes

Un ensemble de (W_q, W_k, W_v) est appelé une tête d'attention et chaque couche d'auto-attention dans un modèle de Transformateur a plusieurs têtes d'attention. Alors qu'une tête d'attention cherche les jetons qui sont pertinents pour chaque jeton, avec plusieurs têtes d'attention, le modèle peut apprendre cette attention pour différentes définitions de pertinence. En effet,

les recherches montrent que plusieurs têtes d'attention dans un Transformateur permettent d'encoder une relation de pertinence qui est similaire à celle des humains. Par exemple, certaines têtes vont, pour chaque jeton, trouver une relation de pertinence avec le jeton suivant alors que d'autres têtes vont encoder une relation entre un verbe et son complément d'objet direct [43]. Étant donné que les Transformateurs ont plusieurs têtes d'attention, ils ont ainsi la possibilité de capturer différents niveaux de relation de pertinence. Les sorties de chaque tête sont alors concaténées ensemble et passées à la couche de réseau de neurones à propagation avant.

2.4.3 L'encodeur

Chaque encodeur d'un Transformateur est composé de deux composantes : une couche d'auto-attention, et un réseau de neurones à propagation en avant (*Feedforward neural network*). L'encodeur utilise une connexion résiduelle [44] autour de ses deux couches et est tout le temps suivi par une couche de normalisation [45] pour faciliter l'optimisation de l'encodeur.

La couche d'auto-attention multi-têtes prend en entrée un ensemble d'encodage (un encodage par jeton de la séquence d'entrée) et applique de l'auto-attention multi-têtes pour générer un nouvel encodage pour chaque jeton en sortie. Par la suite, le réseau de neurones à propagation en avant est appliqué à chaque jeton indépendamment afin de produire un encodage pour chaque position dans une représentation de plus haut niveau [46]. Ces encodages de sorties sont alors passés au prochain encodeur comme une entrée.

Le premier encodeur prend en entrée l'information sur la position ainsi que le plongement de chaque jeton de la séquence d'entrée plutôt que des encodages. Étant donné qu'il n'y a pas de récurrence qui permet au modèle de savoir où il est rendu dans la séquence, l'information sur la position est cruciale. Ainsi, pour chaque jeton, l'information sur la position est calculée à l'aide d'une fonction sinusoïdale. Les Transformateurs utilisent cette approche plutôt qu'un vecteur appris afin de diminuer le nombre total de poids à apprendre ainsi que d'avoir, virtuellement, la possibilité d'avoir une longueur de séquence maximale infinie [12], la seule limite étant d'ordre matériel.

L'information sur la position de chaque jeton est critique pour le Transformateur afin de pouvoir utiliser l'information sur l'ordre des mots dans la séquence et donc pouvoir différencier un même mot qui se répète dans une même séquence d'entrée [12].

2.4.4 Le décodeur

Chaque décodeur d'un Transformateur est composé de trois composantes : une couche d'auto-attention, une couche d'auto-attention multi-têtes sur le dernier encodage produit par la pile d'encodeurs et finalement un réseau de neurones à propagation en avant. Tout comme pour l'encodeur, le décodeur possède des connexions résiduelles avec une couche de normalisation autour de chaque composante. Le décodeur fonctionne de façon similaire à celle de l'encodeur, mais l'ajout de l'attention sur la représentation du dernier encodeur lui permet aussi de tirer des informations pertinentes sur la séquence.

Tout comme le premier encodeur, le premier décodeur prend l'information de position et les plongements de la séquence de sortie finale (qu'il construit au fur et à mesure) comme entrée plutôt que des encodages. Comme le Transformateur ne doit pas utiliser l'état courant ou futur pour prédire une sortie, l'auto-attention est appliquée seulement sur les jetons qui précèdent le jeton à prédire. Cela est fait en masquant les autres jetons de la séquence de sortie. Cela permet de prévenir un flux d'information vers la gauche [12]. Le dernier décodeur est suivi d'une couche linéaire complètement connectée qui projette le vecteur produit par l'ensemble des décodeurs sur un seul vecteur de la taille du vocabulaire du modèle qui contient le score que le i^e mot soit la prédiction. Ce vecteur passe ensuite à travers un softmax qui va alors distribuer les probabilités que chaque mot soit le mot choisi, le mot avec la plus grande probabilité est alors ajouté à la séquence de sortie.

2.5 Transformateur-XL

Une variante des Transformateurs, les Transformateurs-XL (*Transformer-XL*) [47] s'attaque aux limites des Transformateurs de [39], qui traitent une séquence au niveau du caractère (et non du jeton). Ceux-ci possèdent une limite majeure : ils possèdent une longueur de segment maximale (ici, la séquence peut être infinie et le transformateur la traite en la découpant en segment) ainsi, ils sont obligés de séparer la séquence en plusieurs segments afin de pouvoir les traiter. De fait, l'attention se fait ainsi seulement à l'intérieur d'un même segment [47].

Le Transformateur-XL utilise un mécanisme de récurrence capable d'utiliser de l'information provenant de segments précédents, ce mécanisme est accompagné du concept de position relative [47]. Ainsi un Transformateur-XL est capable d'encoder la position d'une entrée en prenant en compte dans quel segment il est. Cela permet au modèle de ne pas perdre de cohérence temporelle et permet au mécanisme de récurrence de pouvoir utiliser l'information des segments précédents afin de pouvoir utiliser l'attention des segments déjà traités [47]. Ces mécanismes permettent au Transformateur-XL d'accéder à n'importe quel état caché dont il

a besoin afin de produire un encodage de la séquence d'entrée.

Ces différentes architectures, et plus spécifiquement les Transformateurs et Transformateurs-XL sont les fondements des modèles de langues modernes. Les modèles de langue sont les fondements du TALN, car ils permettent de mieux représenter les interactions entre les mots d'une séquence. Ainsi, les réseaux de neurones utilisant des modèles de langue se basent sur leur compréhension de la langue et du monde pour accomplir des tâches spécifiques.

2.6 Modèle de langue

Le but d'un modèle de langue est d'estimer la probabilité d'une séquence de mots dans une langue. Il permet ainsi d'assigner une probabilité à une séquence de mots. Ces modèles de langue sont statistiques c'est-à-dire qu'ils ne sont pas définis par des règles formelles, mais sont appris sur un large corpus.

Dans la section suivante, nous allons présenter les trois modèles de langue majeurs à savoir les modèles de langue N-gramme, ceux obtenus à l'aide d'un RNN et ceux utilisant les Transformateurs.

2.6.1 Modèle de langue N-gramme

Ces modèles sont basés sur les chaînes de Markov. La probabilité d'une séquence de mots étant le produit des probabilités de chaque mot sachant les mots précédents, on introduit une simplification en prenant uniquement en compte les n mots précédents. Autrement dit, le modèle permet de prédire les mots suivants étant donné les n mots précédents.

L'idée est de calculer $P(m|c)$, la probabilité du mot m sachant son contexte, soit tous les mots avant c. Pour un modèle bigramme, cela revient à approximer la probabilité d'un mot donné sachant tous les mots précédents en utilisant seulement la probabilité conditionnelle du mot précédent :

$$P(m_n|m_1^{n-1}) \approx P(m_n|m_{n-1})$$

Et de manière générale pour un modèle N-gramme :

$$P(m_n|m_1^{n-1}) \approx P(m_n|m_{n-N+1}^{n-1})$$

Pour obtenir ces probabilités, les modèles utilisent un grand corpus et calculent la fréquence pour chaque mot et séquence de mots, ainsi pour obtenir la probabilité du bigramme $<$

$m_1, m_2 >$:

$$P(m_1|m_2) = \frac{f(m_1, m_2)}{m_1} \tag{2.1}$$

où $f()$ retourne la fréquence du mot ou de la séquence de mots.

Ces modèles possèdent certaines limites : étant donné que les modèles N-grammes s'apparentent à une technique de glissage de fenêtre (*Sliding window*), ils ne sont pas capables de capter les dépendances entre les mots qui sont éloignés dans la séquence. Aussi, bien que des techniques comme le lissage de Good-turing [48] atténuent ce problème par l'introduction d'un jeton spécial, les modèles n-grammes ne sont pas capables de bien gérer des mots qui ne font pas partie de leur vocabulaire.

2.6.2 Modèle de langue obtenu avec un RNN

Un modèle de langue obtenu avec un RNN utilise souvent l'architecture de LSTM ou de Bi-LSTM. Ceux-ci sont spécialement utilisés pour le traitement de séquences, car ils gardent la mémoire du traitement des mots vus précédemment [16]. Ils intègrent de cette façon le mot à prédire dans son contexte. Cette architecture est donc tout indiquée pour la tâche de modélisation de langue.

Dans ce genre de réseaux de neurones, les séquences de mots sont premièrement transformées en vecteurs de mots denses à l'aide de modèles comme GLoVe avant d'être utilisées comme entrée du réseau. Ainsi, chaque mot est projeté dans un espace de dimension d, c'est ce qu'on appelle le plongement lexical ou plongement de mot (*word embedding*) dans lequel il est représenté par un vecteur de sorte que deux mots similaires seront représentés par des vecteurs relativement proches selon une mesure de distance.

Pour créer un modèle de langue en utilisant un RNN, il suffit d'ajouter en aval du modèle de représentation de mot en vecteur dense une couche récurrente, que ce soit un RNN simple, un LSTM ou un Bi-LSTM suivi d'un softmax. Le modèle de langue doit alors apprendre à prédire la probabilité du prochain jeton sachant son historique.

Dans le cas de ELMO [49], ils utilisent un bi-LSTM suivi d'un softmax. Durant la passe en avant (*forward pass*), l'historique contient les mots avant le jeton à prédire et durant la passe

arrière (*backward pass*), l'historique contient les mots après le jeton à prédire, respectivement :

$$p(x_1, ...x_n) = \prod_{i=1}^{n} (x_i | x_1, ..., x_{i-1})$$
$$p(x_1, ...x_n) = \prod_{i=1}^{n} (x_i | x_{i+1}, ..., x_n)$$

La dernière couche concatène les états cachés dans chaque direction et le passage à travers un softmax pour obtenir les probabilités normalisées de chaque jeton.

Le modèle est entraîné pour minimiser l'opposée du log-vraisemblance (*Negative log-likelihood, NLL*) des mauvais mots (donc, maximiser le log-vraisemblance des bons mots) dans chaque direction.

Ce faisant, ELMO apprend une représentation contextuelle profonde des mots (*Deep contextualized word representations*) [50]. C'est l'un des premiers modèles de langue pré-entraîné. Il suffit alors de le spécialiser dans une tâche en ajoutant un réseau de neurones à propagation en avant qui va utiliser la représentation des mots obtenus de ELMO et affiner le système au complet afin de le spécialiser sur une tâche du TALN.

Les représentations compactes et l'intégration du concept de similarité introduites avec les vecteurs de mots denses favorisent la généralisation du modèle, c'est-à-dire la capacité du modèle à faire une prédiction correcte pour un cas qu'il a peu, voire pas rencontré lors de l'entraînement.

2.6.3 Modèle de langue basé sur les Transformateurs

En se basant sur les Tranformateurs [12], de nouveaux modèles de langues basés sur le transfert d'apprentissage (*Transfer learning*) et l'affinement (*fine-tuning*) ont été créés. L'idée derrière ces modèles de langue est de leur faire apprendre une tâche générale puis, par la suite de l'affiner sur une tâche plus spécifique afin de tirer parti des apprentissages de la tâche générale. Ces modèles sont entraînés sur la tâche de modélisation de la langue, c'est-à-dire de prédire le prochain mot dans une phrase sachant le reste de celle-ci. L'idée de transfert d'apprentissage est très logique dans le cas des modèles de langue. En effet, en TALN il est primordial d'avoir une compréhension générale du monde et de la langue avant de se pencher sur une tâche spécifique comme la tâche de question-réponse. Afin d'acquérir cette connaissance générale, un modèle de langue a besoin d'une quantité phénoménale de données ainsi, l'apprentissage par transfert est idéal, car il nous permet de faire cet apprentissage une seule fois et ensuite le modèle de langue peut être rapidement spécialisé sur une tâche spécifique. Nous allons maintenant présenter les deux modèles de langues basés sur l'architecture des Transformateurs

que nous avons utilisés dans nos expériences.

BERT

Une des révolutions les plus récentes en TALN qui a engendré la plupart des résultats de l'état de l'art actuel provient de BERT [13], un modèle de langue basé sur l'architecture des Transformateurs [12].

Dans BERT, seule la partie encodeur avec mécanisme d'auto-attention est utilisée, cependant l'information sur la position est apprise ce qui limite le modèle de langue final à une longueur de séquence maximale déterminée à l'entraînement (à savoir 512 jetons). Profitant de l'architecture des encodeurs, et plus précisément de l'auto-attention, ce modèle de langue utilisé est bidirectionnel, c'est-à-dire que lorsque le modèle détermine la représentation d'un mot dans un contexte, il prend en compte autant l'historique du contexte, c'est-à-dire les mots à gauche, mais aussi la suite du contexte, soit les mots à droite. BERT profite donc de tous les avancements récents en matière d'attention afin de produire des plongements de mots qui contiennent l'information nécessaire à la tâche du modèle.

Pour aider BERT à apprendre, des jetons spéciaux sont ajoutés dans la séquence d'entrée, à savoir le jeton [CLS] qui indique le début d'une séquence et le jeton [SEP] qui indique la fin d'une séquence. Comme BERT peut prendre jusqu'à deux séquences en entrée, il est possible d'avoir deux jetons [SEP].

Pour entraîner ce modèle de langue, Google propose l'approche suivante. BERT est un modèle de langue entraîné de façon non-supervisée sur un très grand corpus comme Wikipédia et se fait sur 2 tâches. La première tâche permet au modèle d'apprendre les relations entre les mots (*Masked Language Model* MLM). Pour ce faire, le modèle apprend à prédire un mot masqué dans une séquence, comme le montre l'exemple suivant où le [MASK] remplace le mot *président*. En somme, 15% du corpus d'entraînement est masqué et c'est à BERT de retrouver le bon mot correspondant. Cela permet au modèle de produire des représentations de mots pertinentes pour tous les mots de la phrase.

> Exemple : Barack Obama was the first African-American
> [MASK] of the United States.

La seconde tâche permet au modèle d'apprendre les relations entre les séquences. Pour ce faire, le modèle est entraîné avec deux phrases en entrée. Le modèle de langue doit alors apprendre à prédire si une séquence A est suivie d'une séquence B ou non, comme l'illustrent les deux exemples suivants. Dans le premier cas, la séquence B est la suite de la séquence A

mais pas dans le second cas. Cela permet au modèle d'apprendre des bases de compréhension multi-phrases et aussi des bases d'implication textuelle (*textual entailment*).

Exemple 1 : The man went to the store. He bought a gallon of milk.
Exemple 2 : The man went to the store. Penguins are flightliess birds.

Avec cet entraînement, nous obtenons un modèle pré-entraîné auquel il faut ajouter une couche supplémentaire pour l'affiner sur une tâche spécifique de TALN. Ainsi, un modèle utilisant BERT peut être spécialisé sur des tâches de classification aussi bien au niveau de la phrase comme l'inférence textuelle (*natural language inference*) [51, 52] qu'au niveau du jeton comme la tâche de question-réponse ou la reconnaissance d'entités nommées [9, 53] . BERT, du fait de ses deux tâches d'entraînement distinctes, se prête particulièrement à de l'affinement sur une tâche comme de la question-réponse. En effet, le fait d'être entraîné à apprendre des relations entre deux séquences semble idéal pour une tâche où l'on fournit une question et une séquence. Aussi la tâche d'apprentissage de relations entre les mots assure une bonne représentation de chaque mot que ce soit dans la question ou dans la séquence.

Le modèle de langue pré-entrainé BERT se décline en deux versions majeures, une version de base qui compte 12 couches d'encodeurs, 12 têtes d'attentions et pas moins de 110 millions de paramètres. La version large compte 24 couches d'encodeurs, 16 têtes d'attentions et 340 millions de paramètres.

XLnet

XLnet [54] est un modèle de langue adaptant l'architecture des Transformateurs-XL [47]. Spécifiquement, XLnet reprend le mécanisme récurrent ainsi que le mécanisme d'information positionnelle relative introduite dans les Transformateurs-XL afin de pouvoir gérer des séquences de longueur arbitraire et théoriquement infinie (la limite étant d'ordre matériel) alors que BERT à une longueur de séquence maximale de 512 jetons fixés lors de son pré-entraînement. Cela donne un avantage considérable à XLnet sur des tâches impliquant des séquences très longues. Aussi XLnet remplace la tâche d'apprentissage de relations entre mots par une tâche d'apprentissage de relations entre mots par permutations (*Permutation language modeling)* [54]. Cette tâche consiste à prédire un jeton spécifique d'une séquence en traitant les autres dans un ordre différent. Par exemple, prenons la phrase :

Je mange un gâteau au chocolat.

Où XLnet doit prédire le mot "gâteau", durant l'entraînement, au lieu de masquer le mot comme le fait BERT, il analyse la séquence dans un ordre aléatoire, par exemple "chocolat",

"mange", "un", "Je", "au" et doit prédire le mot "gâteau". Ainsi l'ordre de prédiction n'est pas de gauche à droite, mais il est aléatoire. Cela force XLnet à être bidirectionnel et à apprendre une meilleure relation entre les mots, en faisant donc un meilleur modèle de langue.

Cependant, XLnet est un modèle très coûteux et difficile à entraîner sans des ressources matérielles importantes.

RoBERTa

RoBERTa [55] est un modèle de langue directement inspiré de BERT [13]. Les différences majeures sont que RoBERTa est entraîné sur énormément plus de données (161GB vs 16GB pour BERT). Spécifiquement, BERT est seulement entraîné avec le BookCorpus [56] alors que RoBERTa est aussi entraîné avec CC-NEWS [55, 57], OpenWebText [58] et Stories [59]. RoBERTa utilise une stratégie de masquage dynamique, alors que BERT utilise une stratégie statique. Comparé à BERT, RoBERTa n'est pas entraîné sur la tâche de prédiction de prochaine phrase et est entraîné avec des séquences plus longues.

Pour ce qui est de la tâche de MLM, BERT applique un masque une seule fois sur tout le jeu de données d'entraînement alors que RoBERTa duplique le jeu de données d'entraînement dix fois et applique des masques différents sur chaque duplicata. En ce qui concerne la suppression de la seconde tâche d'apprentissage de BERT, RoBERTa compare quatre façons de faire afin d'évaluer si cette tâche est réellement nécessaire :

- *SEGMENT-PAIR* : la même approche que celle de BERT, Chaque entrée est une paire de segments et chaque segment peut contenir plusieurs phrases.
- *SENTENCE-PAIR* : une paire de phrases, soit à partir d'une portion continue d'un document ou de documents séparés. Chaque exemple contient donc exactement deux phrases. Ici aussi on garde la tâche de prédiction de prochaine phrase.
- *FULL-SENTENCES* : le corpus d'entraînement est prétraité pour former des exemples avec une longueur de séquence aussi proche que possible que la longueur de séquence maximale de RoBERTa (512 jetons) avec des phrases d'un ou plusieurs documents. Quand on atteint la fin d'un document, on ajoute les phrases du prochain document, des exemples peuvent donc être composés de phrases qui ne se suivent pas. Dans ce cas, on ne garde que la tâche d'apprentissage de relations entre mots.
- *DOC-SENTENCES* : La même chose que l'approche *FULL SENTENCES*, mais on ne mélange jamais plusieurs documents. On ne garde pas la tâche de prédiction de prochaine phrase.

Après l'évaluation de ces quatre approches, les deux approches qui ne gardent pas la tâche de

prédiction de prochaine phrase obtiennent des résultats légèrement meilleurs sur MNLI-m, une tâche d'implication textuelle qui demande de classifier si une première séquence appuie, contredit ou est neutre par rapport à la seconde [60].

Finalement, les auteurs de RoBERTa comparent l'incidence de la taille de lot (*batch size*) sur les performances, il s'avère que RoBERTa avec une taille de lot de 2000 et entraîné durant 125000 étapes avec un taux d'apprentissage de 7×10^{-4} performe mieux que s'il était entraîné avec les hyperparamètres de BERT, soit une taille de lot de 256 durant $1M$ d'étapes avec un taux d'apprentissage de 1×10^{-4}.

Bref, RoBERTa se veut une version plus robuste et mieux entraînée que BERT entre autres en modifiant les hyperparamètres du modèle, mais aussi en modifiant les tâches d'apprentissage. Aussi, le fait que le modèle pré-entraîné RoBERTa ne soit pas entraîné sur la tâche de prédiction de prochaine phrase lui permet d'améliorer les performances sur une tâche d'implication textuelle.

ALBERT

ALBERT [61], quant à lui, est un modèle de langue qui diffère par son architecture et ses tâches de pré-apprentissage. Premièrement, après avoir observé que dans tous les modèles existants comme RoBERTa, BERT ou XLnet, la taille de la couche cachée est directement liée à celle des plongements de mots utilisés, ALBERT propose de factoriser les matrices de plongement en deux matrices plus petites. Une matrice s'occupe d'apprendre à représenter le plongement du mot indépendamment de son contexte, et la seconde apprend à représenter le contexte du mot. Ce faisant, ALBERT a un meilleur contrôle sur la taille de sa couche cachée.

De plus, à l'inverse de BERT, ALBERT propose de partager les paramètres à apprendre entre toutes les couches. Ces deux modifications à l'architecture de base permettent d'obtenir un modèle de langue avec beaucoup moins de paramètres, ALBERT large en compte 18 fois moins que dans BERT large accompagné d'un temps de calcul réduit. Finalement, ALBERT modifie aussi la tâche de prédiction de prochaine phrase. En effet, cette tâche ne semblait pas améliorer la qualité du modèle de langue au vu des performances de RoBERTa. Ainsi, cette tâche est remplacée par la tâche de prédiction d'ordre de phrase illustrée à la Figure 2.6. Au lieu de deux phrases de documents aléatoires lors de la création d'exemples négatifs, cette tâche construit un exemple négatif en prenant un exemple positif et en inversant l'ordre des phrases pour le rendre négatif. Cette nouvelle tâche oblige le modèle de langue à construire une forte représentation de chaque séquence afin de pouvoir faire des inférences sur l'ordre des phrases.

De manière générale, ALBERT se veut être un modèle de langue capable de représenter des séquences plus efficacement et plus rapidement.

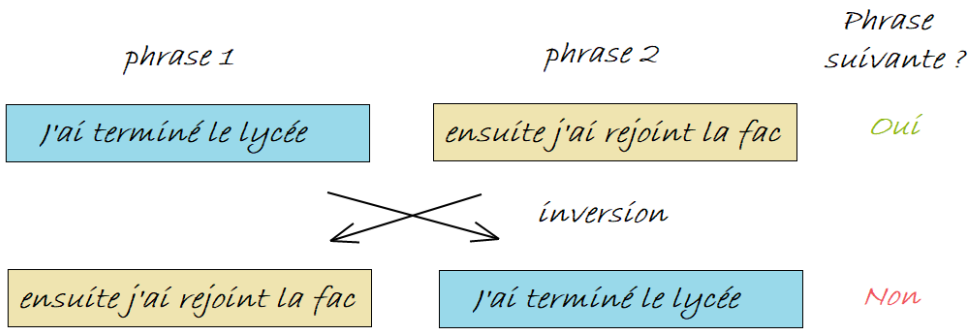


Figure 2.6 Illustration de la tâche de prédiction d'ordre de phrase [5]

Grâce à ces différents modèles de langue, il nous est possible d'accomplir diverses tâches du TALN comme la tâche de *question-réponse dans le domaine ouvert*. Cette tâche, qui se divise en plusieurs sous-tâches, fait appel à plusieurs types de modèles de langues afin d'être résolue.

2.7 La tâche de question-réponse (Question Answering)

Dans ce mémoire, ainsi que nous l'avons précédemment mentionné, nous visons à développer un système de question-réponse dans le domaine ouvert, c'est-à-dire capable de retrouver une réponse à une question en langue naturelle dans un vaste corpus de documents. Cette tâche se base sur les différentes architectures et modèles de langue présentés dans les sections précédentes. Elle implique également le recours à deux sous-tâches : la tâche d'implication textuelle dans le domaine de la question-réponse qui permet d'identifier un document qui contient la réponse à la question, et la tâche d'ordonnancement de documents qui est nécessaire pour classer les documents selon leur pertinence pour la question donnée. Dans les sections suivantes, nous présentons un aperçu de l'état de l'art sur chacune de ces sous-tâches ainsi que sur les systèmes de question-réponse desquels nous nous inspirons et auxquelles nous nous comparons.

2.7.1 L'implication textuelle

Dans nos expériences, l'implication textuelle appliquée à la tâche de *question-réponse dans le domaine ouvert* est la fondation sur laquelle nous nous appuyons afin d'obtenir un ordonnan-

5. Tiré de `https://lbourdois.github.io/blog/nlp/ALBERT/`

cement efficace des documents selon leur pertinence pour une question donnée.

La reconnaissance de l'implication textuelle (*Recognizing Textual Entailment* (RTE)) permet d'avoir une formule générale d'inférence entre deux textes [62]. Plus précisément, la tâche est de reconnaître si la déclaration textuelle (appelée hypothèse H) peut être déduite (inférée) d'un autre texte (appelé le texte-T) [62, 63]. Cette relation s'obtient entre des fragments de texte lorsqu'un fragment dans un certain sens implique l'autre [64]. L'automatisation de la reconnaissance de l'implication textuelle peut prendre en charge une grande variété de tâches basées sur le texte, y compris la recherche d'information, l'extraction d'information, les réponses à des questions, la synthèse de textes et la traduction automatique [64]. C'est donc une tâche qui nous intéresse particulièrement dans le cadre d'un système de questions-réponses.

Cependant, le défi principal de la reconnaissance de l'implication textuelle est la variabilité de la langue, c'est-à-dire que différents fragments de textes peuvent avoir la même signification [62].

Un dérivé de l'implication textuelle, la tâche relative au jeu de données QNLI (*Question-answering Natural Language Inference*) créée pour le standard de référence GLUE [60], cherche uniquement à identifier si une phrase contient oui ou non la réponse à une question plutôt que d'extraire la réponse même. Nous appelons cette tâche la classification de paire de phrases dans la question-réponse. Cette tâche permet donc de supprimer l'exigence que le modèle sélectionne la réponse exacte (comme dans la tâche de question-réponse), mais permet également de supprimer les hypothèses simplificatrices selon lesquelles la réponse est toujours présente dans l'entrée [60]. Cela n'est en effet pas le cas dans le cadre d'un système de question-réponse dit ouvert.

Une des premières approches fructueuses sur la tâche de QNLI, ERNIE (*Enhanced language Representation with Informative Entities*) [65], est en fait un modèle de langue qui utilise le web sémantique pour améliorer ses représentations. Le web sémantique est un standard qui préconise un format de donnée afin que l'information soit compréhensible par l'ordinateur en utilisant le modèle Ressource Description Framework (RDF) [66, 67]. Grâce au RDF, il est possible de représenter des phrases et même des documents en entier sous la forme de graphes de connaissance (*Knowledge graphs*, KG) et d'utiliser des bases de connaissances comme DBPedia [3].

ERNIE est un système basé sur l'architecture des Transformateurs qui a été proposé afin de surmonter deux défis reliés à l'incorporation de connaissances externes dans les modèles de représentation linguistique comme BERT. Il y a deux principaux défis, le premier est l'encodage des connaissances structurées [65]. Pour un texte donné, la manière d'extraire et encoder ces faits informatifs dans des KGs pour les modèles de représentation linguistique comme BERT est un problème important. Le deuxième défi est la fusion des informations

hétérogènes. La procédure de pré-entraînement pour la représentation linguistique est très différente de la procédure de représentation des connaissances. Cela conduit à deux espaces vectoriels individuels. Afin de surmonter ces défis, ERNIE préconise un modèle pour la modélisation linguistique pré-entraîné autant sur de larges corpus de texte non structurés que sur des KGs. Pour pallier au premier problème, ERNIE commence par reconnaître les entités nommées dans la séquence d'entrée et les aligne avec des entités de KGs. Par la suite, le graphe de la séquence est transformé en plongement en utilisant un algorithme de plongement de graphe comme TransE [68]. Ce sont ces plongements qui sont utilisés comme entrée à ERNIE. Pour le second défi, le modèle de langue ERNIE a aussi pour objectif de pré-entraînement les tâches de MLM et de prédiction de prochaine phrase. Aussi, il ajoute une tâche de pré-entraînement en masquant des entités nommées dans la séquence d'entrée, l'objectif étant pour le modèle de deviner la bonne entité nommée.

En aval du modèle de langue ERNIE, on ajoute un simple réseau de neurones à propagation en avant afin de prédire si la séquence contient ou non la réponse à la question. Ainsi, ERNIE obtient une précision de 91.3% sur le jeu de données de test de QNLI.

Les autres modèles de langues qui ont permis d'avoir de meilleurs résultats sur la tâche d'implication textuelle dans le domaine de question-réponse sont XLnet [54], RoBERTa [55] et finalement ALBERT [61].

Pour cette tâche, XLnet, RoBERTa et ALBERT, utilisent un réseau de neurones à propagation en avant afin de prédire si la phrase fournie présente ou non la réponse à la question à partir de leur modèle de langue respectif. Spécifiquement, la couche linéaire prend en entrée l'encodage du premier jeton de la séquence d'entrée (le jeton [CLS]) et le projette afin d'obtenir une classification [13].

XLnet, grâce à sa meilleure représentation d'une séquence et de son contexte obtient de bons résultats sur QNLI avec une précision de 94.4%.

Quant à RoBERTa et ALBERT, deux modèles de langue largement inspirés de BERT, ils sont les deux meilleurs modèles de langue pour cette tâche avec, respectivement, 98.8 et 99.2% de précision sur le jeu de données de test de QNLI. Comme mentionné plus tôt, ces deux modèles utilisent un réseau de neurones à propagation en avant qui prend l'encodage du premier jeton afin de déterminer si oui ou non la séquence peut répondre à la question.

Ainsi, ces derniers, de par leurs performances sur QNLI semblent tout indiqués pour la tâche d'inférences textuelles. Dans l'approche proposée dans ce mémoire, nous utilisons la classification de paire de phrase dans la question-réponse comme base pour développer un ordonnanceur en recherche d'information.

2.7.2 Ordonnancement en recherche d'information

L'ordonnancement de documents en recherche d'information est une étape clé dans un système de *question-réponse dans le domaine ouvert*. Pour ce faire, un système doit apprendre à évaluer la pertinence d'un document à partir d'une question. Cette tâche est appelée l'apprentissage à l'ordonnancement (*Learning to rank*). Plus précisément, c'est la tâche qui cherche à construire automatiquement un modèle d'ordonnancement en utilisant des données d'entraînement, de façon à obtenir un modèle capable de trier des documents selon leur degré de pertinence [69].

Les premiers modèles, les modèles les plus simples étaient des modèles booléens [70]. Ceux-ci étaient simplement capables de dire si un document était pertinent ou non. Afin de pouvoir évaluer le degré de pertinence d'un document, les premiers modèles d'ordonnancement se sont appuyés sur la représentation de document par modèle Vectoriel [70]. Dans ce genre de modèle, la question et les documents sont tous représentés sous forme de vecteur clairsemé dans un espace euclidien. Ainsi, il était possible d'évaluer la pertinence d'un document en calculant son produit scalaire avec le vecteur représentant la question. Cette approche, bien que basique reste tout de même utilisée actuellement [5].

La plupart des approches modernes de la tâche d'apprentissage à l'ordonnancement peuvent être divisées en trois sous-catégories : l'approche par point (*pointwise*), l'approche par paire (*pairwise*) et l'approche par liste (*listwise*) [69]. Ces trois types d'approches diffèrent par leur façon de traiter le processus d'apprentissage à l'ordonnancement. En effet, chaque approche a un format d'entrée et de sortie qui lui est particulier, repose sur des hypothèses qui sont différentes et utilise des fonctions de perte différentes.

Approche par point

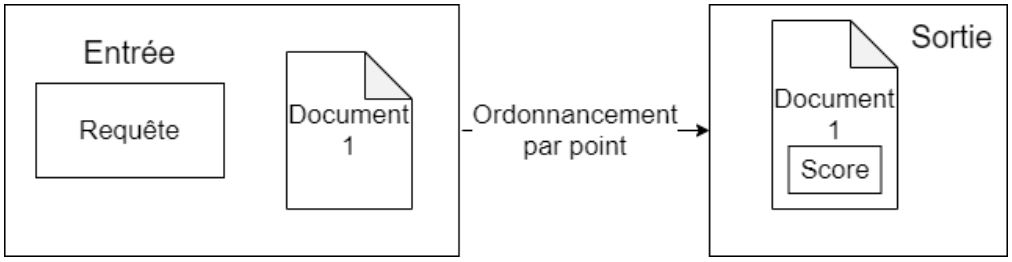


Figure 2.7 Schématisation de l'approche d'ordonnancement par point

Comme illustré à la figure 2.7, l'idée générale de cette approche est que chaque paire de question-document obtient un score évaluant la pertinence du document par rapport à la question. Le problème d'ordonnancement se résume alors soit à un problème de régression,

soit un problème de classification. Dans le premier cas, cela implique que le modèle apprend à prédire un score, et donc utilise un jeu de données d'entraînement qui associe déjà un score à une paire de question-document. Dans le second cas, le modèle apprend à prédire une étiquette pour la paire question-document [69]. Dans les deux cas, l'entrée est donc une paire de question-document.

Une des premières approches de l'apprentissage à l'ordonnancement [71] utilise la fonction de régression des moindres carrés comme fonction de perte pour apprendre la fonction d'attribution de score. Cette approche est limitée par le fait que dans le cas de bonne prédiction la fonction de perte devrait évaluer exactement 0 perte ; or si le modèle fait une prédiction trop forte (par exemple dans le cas d'une régression sur deux classes la fonction d'attribution de score donne (2, 0) alors que l'on s'attend à (1,0)) alors la fonction de perte va calculer une perte non nulle alors que la prédiction était bonne. [69].

Les approches abordant le problème de classification essaient de généraliser la régression et d'apprendre seulement à prédire une étiquette pour une paire. La pertinence du document revient alors au score de confiance du modèle dans sa prédiction. Dans ce genre d'approche, les réseaux de neurones à propagation en avant sont utilisés, car la fonction de décision est un simple produit scalaire entre une matrice de poids apprise et un vecteur de caractéristiques (*features vector*) représentant la paire question-document [72]. Ce vecteur peut être généré par un système en amont qui est aussi entraîné avec le réseau de neurones à propagation en avant ou alors par un système indépendant.

Approche par paire

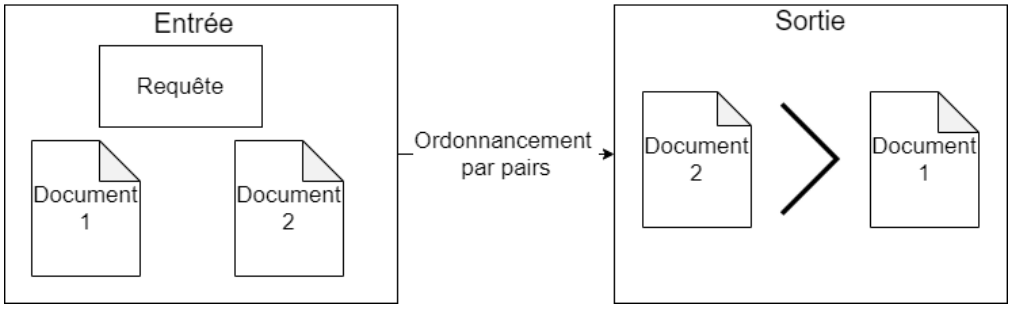


Figure 2.8 Schématisation de l'approche d'ordonnancement par paire

Utilisée par des algorithmes assez répandus comme SVM [73] ou XGBoost [74], l'approche par paire, utilise en entrée une paire de documents, tous représentés sous la forme de vecteurs de caractéristiques.Comme le montre la figure 2.8, un modèle utilisant cette approche doit trouver, parmi ces deux documents, lequel est le plus pertinent pour la requête. Ainsi ici on

ne s'intéresse pas à l'étiquette prédite, mais plutôt à la différence de score relatif [75]. Cette approche permet d'éloigner les documents qui ont une différence de score relatif élevé, mais elles ont un problème : toutes les paires de documents sont traitées avec la même importance, mais ce n'est pas ce à quoi l'on est confronté en réalité. Ainsi cette technique est généralement capable d'améliorer un top 50, mais au dépend d'un top 10 moins précis [76].

Aussi, ce genre d'approche nécessite aussi un nombre d'exemples d'entraînement grandement supérieur au nombre d'exemples nécessaires pour l'approche par point ce qui implique aussi un temps d'entraînement plus long.

Approche par liste

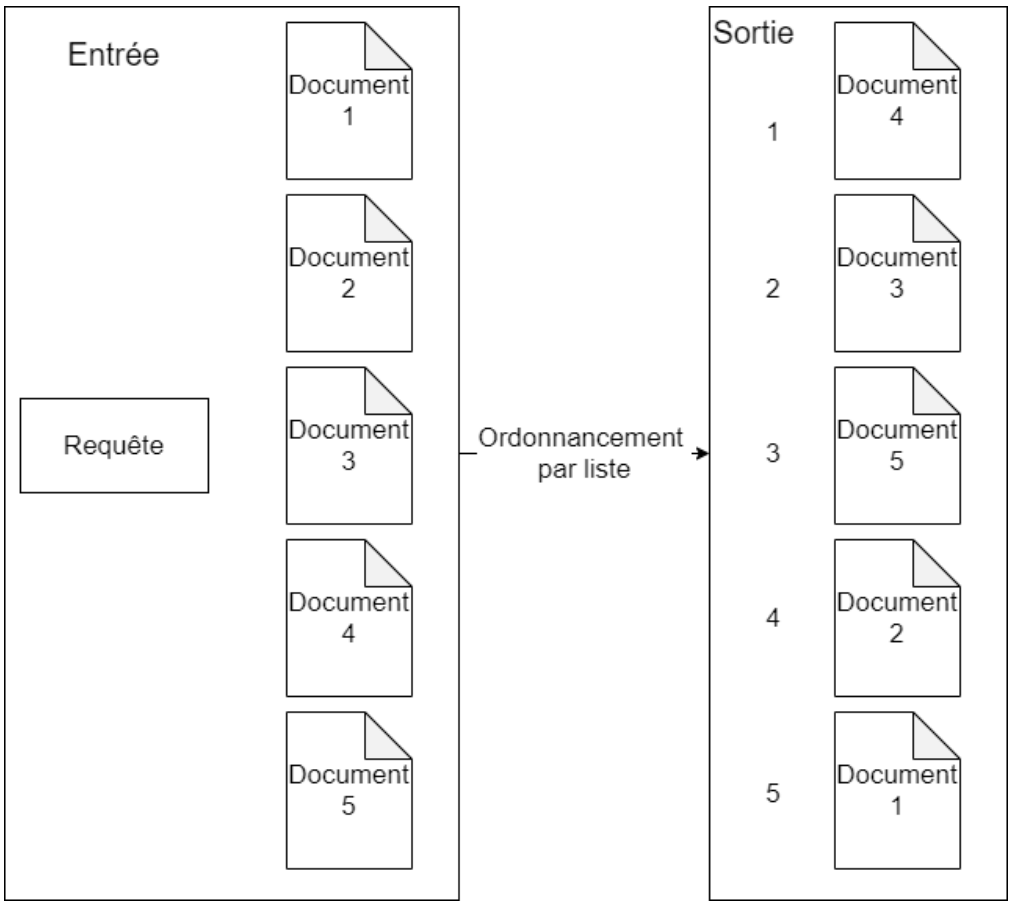


Figure 2.9 Schématisation de l'approche d'ordonnancement par liste

La dernière approche est la seule approche qui prend en compte que la tâche d'ordonnancement se fait sur une liste de documents [77]. Comme l'illustre la figure 2.9, cette approche traite une requête avec une liste de documents et le but est de retourner cette liste de documents, mais de façon ordonnée selon leur pertinence par rapport à la requête [72]. Les approches

récentes comme dans FastAP [78] cherchent à optimiser la mesure de précision moyenne de l'ordonnancement en utilisant une approximation dérivée de la quantification de distance [78] pour apprendre la représentation profonde des documents à ordonner.

Cette approche est, elle aussi, très difficile à mettre en place et difficile à entraîner [72], aussi, dans le domaine de *question-réponse dans le domaine ouvert*, il est difficile de concevoir un jeu de donnée où l'on ordonnerait des documents selon leur pertinence par rapport à une question. En effet, si, par exemple, plusieurs documents contiennent la réponse à la question alors il serait difficile de déterminer lequel des documents est le plus pertinent.

2.7.3 Question-réponse fermée

Les systèmes de question-réponse fermés actuels sont beaucoup influencés par les techniques utilisées pour la tâche de compréhension en lecture automatique. Le MRC est une tâche compliquée où un modèle doit être capable de répondre à n'importe quelle question sur un ou plusieurs documents. Cela implique que les modèles soient capables d'obtenir une bonne représentation du ou des documents qu'ils traitent. À son tour, la tâche de question-réponse fermée est la principale source d'inspiration de la tâche de *question-réponse dans le domaine ouvert* qui adapte les derniers avancements de la tâche de question-réponse afin qu'ils puissent fonctionner pour plusieurs documents par question.

Dans la tâche spécifique de la question-réponse fermée, on fournit au modèle une question et un document, et le modèle doit se charger de trouver la réponse dans le document [4].

Une des premières approches fructueuses présentées par Chen & al. [5] est d'utiliser un Bi-LSTM en aval d'un modèle de plongement de mots comme GLoVE [25] et de le faire suivre de deux classificateurs indépendants. Ces classificateurs ont pour but de calculer pour chaque jeton la probabilité que le jeton soit respectivement le début ou la fin de la réponse à la question [5]. Le bi-LSTM est responsable, à partir de la question et du document tous deux transformés en vecteurs de caractéristiques à l'aide du modèle de plongement de mots, de produire un encodage pour chaque jeton du document. Ces encodages sont alors donnés aux deux classificateurs.

Plus récemment, avec l'apparition des modèles de langues basés sur les Transformateurs, une panoplie de modèles [13, 54, 55, 61] ont vu le jour et sont tous basés sur le même genre d'architecture en aval du modèle de langue. Celui-ci consiste en un réseau de neurones à propagation en avant responsable de classifier chaque jeton du document fourni. Le classificateur va donner deux scores bruts (*raw logit*) indiquant si le jeton est le début ou la fin de la réponse. Ensuite, le jeton avec le plus grand score brut de début de réponse est prédit comme étant le début de

la réponse et similairement pour le jeton de fin.

Certains jeux de données compliquent la tâche initiale en ajoutant des paires de question-document où la réponse n'est pas présente dans le document, ou où la question est impossible [11] et demandent aux modèles d'être capables de dire qu'une paire de question-document est impossible en répondant "<No Answer>".

Là encore, les systèmes les plus performants sont ceux utilisant un modèle de langue basé sur les Transformateurs en amont [54, 55, 61, 61, 79]. Les premières approchent utilisent presque tous la même architecture que celle décrite juste avant avec différents modèles de langue, à savoir l'ajout d'un classificateur en aval du modèle de langue. Cependant ils ajoutent que le score de prédiction de paire impossible est le résultat de l'addition des deux scores bruts du jeton spécial [CLS] qu'ils ajoutent au début du document fourni comme mentionné dans la section 2.6.3. Finalement, une des approches les plus fructueuses met en place le mécanisme de lecture rétrospective (*retrospective reading*) [79] qui intègre deux étapes de lecture automatique : la première étape, une lecture rapide, qui émet un premier jugement sur la paire question-document. Cette lecture se focalise sur les interactions générales entre la question et le document. La seconde étape, une lecture en profondeur, a pour but de vérifier la réponse de la première étape et de donner une prédiction finale. Cette approche [79] établit un nouvel état de l'art dans la tâche de question-réponse fermée sur les jeux de données SQuAD2.0 [11] et NewsQA [80].

Cependant, des travaux ont prouvé que les meilleurs systèmes utilisés pour cette tâche étaient très sensibles aux entrées dites trompeuses - un document qui ne contient pas la réponse à la question, mais qui possède un vocabulaire proche de la question - et ceux-ci ne fonctionnent pas bien lorsque l'on utilise de trop longs documents [15]. Ces deux limites sont autant voire même encore plus présentes dans la tâche de *question-réponse dans le domaine ouvert*. En effet, étant donné que les systèmes pour cette tâche traitent de multiples documents pour une même question, ils ont encore plus de chance de tomber sur des documents trompeurs.

2.7.4 Question-réponse dans le domaine ouvert

Les systèmes de *question-réponse dans le domaine ouvert* peuvent être catégorisés en systèmes de question-réponse utilisant des bases de connaissances ou des graphes de connaissances et ceux utilisant la recherche d'information et des corpus de textes uniquement. Dans le premier cas, les systèmes doivent construire une représentation sémantique de la question et l'utiliser afin de faire une requête à une base de connaissances comme DBpedia [3, 4]. Dans le second cas, les systèmes de question-réponse basés sur la recherche d'information s'appuient sur des documents textuels afin de répondre à une question. La tâche de question-réponse s'appuyant

sur la recherche d'information est assez complexe et peut être divisée en deux sous-tâches, la compréhension de langage naturel et la recherche d'information. Un des défis est de fournir seulement des documents pertinents qui seront utilisés pour répondre à la question.

Une des premières approches, Pauchok, (qui a aussi inspiré ce mémoire) proposait de raffiner l'étape de recherche d'information [81]. Pour ce faire, le système récupère premièrement plusieurs documents à partir d'un vaste corpus. Ces documents sont ensuite segmentés en passages afin d'effectuer une nouvelle étape de recherche d'information. Cette seconde étape de recherche d'information a pour but d'identifier les segments qui ont le plus de chance de contenir la réponse à la question. Nous identifions plusieurs points intéressants dans cette approche, le système de question-réponse est conçu premièrement de manière à rétrécir l'espace de recherche d'un vaste corpus de documents à quelques documents. Par la suite, il effectue un classement des passages générés par la première étape. Finalement, il extrait la réponse d'un des passages. Dans notre approche, nous utilisons le même principe de raffinement de recherche d'information, mais avec des technologies de TALN plus modernes.

Notre pipeline s'inspire aussi de DrQA [5], qui est en fait le système que nous avons tenté d'améliorer. Ce système de *question-réponse dans le domaine ouvert* basé sur la recherche d'information est l'un des plus populaires dans les systèmes de *question-réponse dans le domaine ouvert*. DrQA propose un pipeline en deux étapes. La première étape, le Chercheur de documents est construit par-dessus le corpus de Wikipedia indexé par pages et a pour but de retourner les pages pertinentes à la question donnée. La seconde étape est l'extraction de réponses. DrQA utilise un réseau de neurones Bi-LSTM basé sur le mécanisme d'attention [82, 83] pour extraire les candidats des différentes pages et finalement sélectionne le meilleur candidat. Dans l'architecture de DrQA, il n'y a pas de raffinement après la première étape, aussi, l'extracteur de réponses est responsable de sélectionner la meilleure réponse possible.

Afin de pallier à la principale lacune de DrQA qui est la sélection de la bonne réponse, Wang et al. [6] propose R^3, une approche innovante proposant un module entre le Chercheur de documents et l'extracteur de réponse qui a pour but de classer les phrases générées par les documents trouvés à la première étape. Ce classeur, basé sur l'apprentissage par renforcement, est entraîné conjointement avec le module d'extraction de réponses. R^3 est ainsi le premier système de *question-réponse dans le domaine ouvert* incorporant un module de classement de phrases basé sur l'apprentissage machine. Bien que ce pipeline introduise une approche basée sur l'apprentissage machine dans le but de raffiner l'étape de recherche d'information, R^3 obtient des résultats assez similaires à ceux de DrQA.

Dans une approche similaire, Lee et al. [84] proposent un raffinement par le classement fait sur les paragraphes - plutôt que les phrases - générés par les documents récupérés lors de la

recherche d'information afin d'améliorer le rappel du système de *question-réponse dans le domaine ouvert* en utilisant l'apprentissage au classement [69]. L'approche proposée compare trois fonctions différentes de classement et s'appuie sur un Bi-LSTM pour encoder la question et chaque paragraphe. En fin de compte, la meilleure fonction de classement est le produit scalaire entre les deux encodages.

Une autre approche basée sur une architecture de type *Transformateur* [12] et utilisant des techniques de *pré-entraînement* [49] a inspiré BERTserini [7], un système de *question-réponse dans le domaine ouvert* utilisant un réseau de neurones basé sur BERT [13] pour leur module d'extraction de réponses précédé par Anserini [85,86], une bibliothèque logicielle libre de droits pour la recherche d'information qui utilise le moteur de recherche Lucene [87]. BERTserini est aussi le premier pipeline de *question-réponse dans le domaine ouvert* qui utilise le corpus de Wikipedia indexé par paragraphes au lieu de pages. À l'instar de DrQA, BERTserini ne comporte aucune étape de classement entre la recherche d'information et l'extraction de réponses.

SPARC [88] quant à lui s'appuie sur DenSPI [89], un système de *question-réponse dans le domaine ouvert* qui se base sur BERT [13] afin de pré-traiter son corpus : Wikipédia indexé par phrase. Dans SPARC, on représente chaque phrase avec un encodage dense obtenu à partir de BERT et un encodage contextualisé clairsemé (*Contextualized Sparse Representations*) en utilisant le mécanisme d'auto-attention pour assigner un poids pour chaque n-gramme du vecteur. SPARC représente la question de façon similaire, ainsi la tâche de *question-réponse dans le domaine ouvert* se résume à une recherche de la représentation qui maximise la similarité des vecteurs clairsemés et celle des vecteurs denses afin de trouver la phrase qui contient la réponse qui obtient le meilleur score. C'est système de *question-réponse dans le domaine ouvert* le plus rapide, entre autres grâce à son pré-traitement colossal et il nécessite beaucoup de ressources physiques afin de pouvoir fonctionner.

Pour finir, inspiré par BERT, Multi-passage BERT [8] propose un pipeline en trois étapes. Premièrement, ils effectuent une recherche d'information en utilisant un corpus de Wikipédia indexé par passages comptant au plus 100 mots pour trouver les 100 passages les plus pertinents à l'aide de ElasticSearch [90] et l'algorithme BM25 [91]. Leur choix d'utiliser des passages plutôt que des pages rejoint les conclusions de [14] qui montre qu'un contexte de quelques phrases est nécessaire pour qu'un système de *question-réponse dans le domaine ouvert* puisse répondre à une question donnée. Par la suite, ils utilisent un ordonnanceur de paragraphe qui utilise BERT pour garder le top-30 passages. Finalement un second système basé sur BERT est utilisé pour extraire les réponses candidates des 30 passages. Après un traitement des réponses similaires trouvées dans différents passages, ils appliquent la technique de normalisation

globale [92] afin d'obtenir des probabilités qui sont comparables entre les différents passages. Dans notre approche, nous nous inspirons de Multi-passage BERT mais nous utilisons des passages de plus grande taille grande afin de garder un maximum de contexte. Aussi, comme notre sujet s'intéresse à l'intégration du module d'extraction de réponse, nous n'utilisons pas de technique de normalisation globale sur les candidats de réponse. D'autre part, nous mettons en place un ordonnanceur de passages basé sur RoBERTa plutôt que BERT car RoBERTa est plus efficace dans la tâche d'inférence textuelle reliée à la tâche de question-réponse.

2.8 Conclusion

En somme, l'approche que nous présentons dans ce mémoire est largement inspirée de l'état de l'art dans le TALN et consiste en trois étapes.

Tout d'abord, nous proposons une recherche d'information fortement inspirée de DrQA [5]. Cependant, suivant les recommandations de [14] nous utilisons un corpus de Wikipédia indexé par passages de plusieurs phrases afin de conserver l'information contextuelle nécessaire pour répondre à une question.

Ensuite, afin de raffiner la première étape, nous mettons en place un ordonnanceur de passages qui a pour but de classer les passages de notre première étape et de n'en garder qu'une poignée pour notre dernier module. Cette ordonnanceur utilise une approche par point [69, 72] afin d'utiliser un modèle d'inférence textuelle basé sur RoBERTa [55], l'un des meilleurs modèles de langue pour ce genre de tâche selon le standard de référence GLUE [60].

Finalement, notre dernière étape est l'extraction de réponses. Pour cette étape nous utilisons un modèle basé sur BERT [13] et qui est aussi capable d'identifier une paire de question-passage qui est impossible [11]. Avec cette vérification en plus de l'étape précédente, nous nous assurons que notre système est capable de détecter un passage qui pourrait être trompeur [15].

Afin de mettre en place notre système, nous sommes partis d'un système déjà existant, DrQA [5]. Le prochain chapitre présente ce système en détail.

CHAPITRE 3 DrQA

Notre système, Re-BERT OQA, est basé sur DrQA [5] et est conçu pour répondre à des questions sur des faits en anglais en utilisant Wikipédia comme source de connaissances. Dans ce chapitre, nous présentons le pipeline de DrQA. Ensuite, nous discutons de ses limites auxquelles nous apportons une solution avec notre architecture Re-BERT OQA.

Comme nous pouvons le voir dans la Figure 3.1, DrQA est constitué de deux sous-systèmes appelés le sélecteur de page et le lecteur de paragraphe. Le lecteur de paragraphe est responsable de l'extraction de la réponse d'une des pages Wikipédia retournées par le sélecteur de page.

3.1 Le Sélecteur de page (*Retriever*)

La première étape du pipeline est le sélecteur de page. Ce module est responsable, avec une question donnée, de réduire l'espace de recherche en sélectionnant une liste des top-5 documents dans son corpus. Ces documents proviennent d'un corpus de Wikipédia indexé par pages qui compte plus de 5 millions de pages.

Dans le but de sélectionner le top-5 documents, les pages et les questions sont représentées et comparées en tant que vecteurs TF-IDF pondérés de sac de mots. Afin d'obtenir de meilleures performances, DrQA a un vocabulaire de taille fixe de 2^{24}. À l'aide d'un analyseur lexical, chaque page et chaque question sont analysées afin d'en extraire des bi-grammes. Ensuite, en utilisant le hachage MurmurHash3 [93], chaque jeton est converti en un entier non signé qui est utilisé comme indice pour faire correspondre chaque bi-gramme à un des 2^{24} sacs selon la technique de hachage de Weinberger et al. [94].

Afin de représenter toutes les pages du corpus de Wikipédia, DrQA crée une matrice creuse (*sparse matrix*) où chaque ligne est un vecteur TF-IDF pondéré de sac de mots représentant une page du corpus. Quand une question est donnée au sélecteur de page, celle-ci est aussi transformée en un vecteur TF-IDF pondéré de sac de mots. DrQA obtient un score de pertinence pour chaque page du corpus en calculant le produit scalaire entre le vecteur de la question et chaque ligne de la matrice. Finalement, le système retourne les 5 pages associées aux 5 plus grands scores.

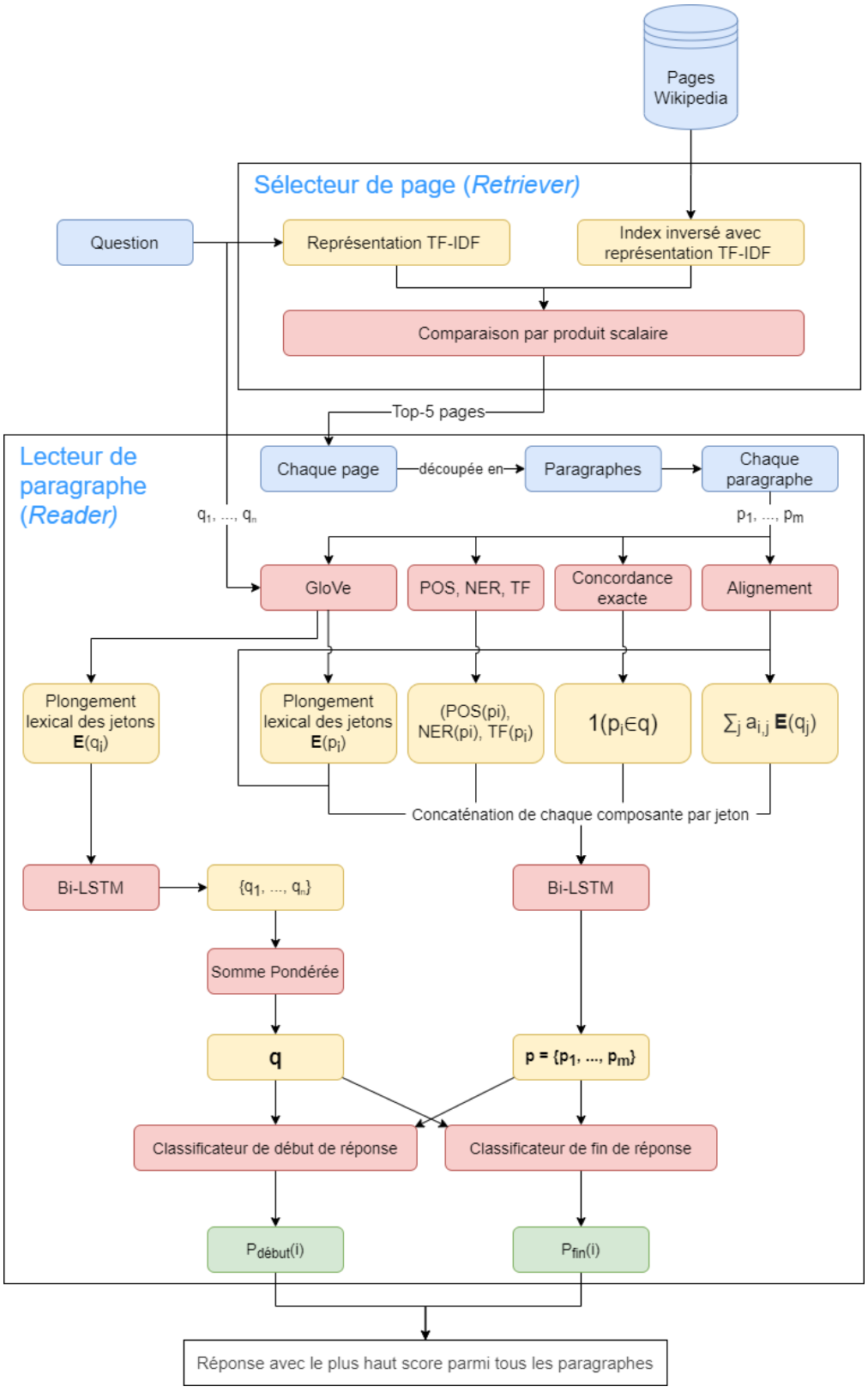


Figure 3.1 Architecture de DrQA

3.2 Le Lecteur de paragraphe (*Reader*)

Le lecteur de paragraphe de DrQA est basé sur un algorithme neuronal de compréhension de lecture qui utilise un passage p de m jetons unigrammes (*tokens*) $p_1, ..., p_m$ et une question q de n jetons $q_1, ..., q_n$ tous deux obtenus grâce à l'analyseur CoreNLP de Stanford [95] . Le but du lecteur de paragraphe est de calculer pour chaque jeton p_i la probabilité $P_{\text{début}}(i)$ et $P_{\text{fin}}(i)$ que p_i soit le début ou la fin de la réponse à la question.

Comme la plupart des systèmes utilisant une approche neuronale pour la compréhension de lecture, le lecteur de paragraphe construit un plongement (*embedding*) pour la question et pour chaque jeton du paragraphe. Ensuite, le lecteur de paragraphe calcule la similarité entre la question et chaque jeton dans le paragraphe et utilise finalement le score de similarité entre chaque paire question-jeton pour décider où est le début et la fin de la réponse dans le passage.

Plus précisément, afin de calculer le plongement du passage $\{\mathbf{p}_1, ..., \mathbf{p}_m\}$, le lecteur de paragraphe commence par créer une représentation $\tilde{p} = \{\tilde{\mathbf{p}}_1, ..., \tilde{\mathbf{p}}_m\}$ en concaténant quatre caractéristiques (*features*) :

1. Un plongement lexical de chaque jeton $\mathbf{E}(p_i)$ provenant du modèle de plongements lexicaux pré-entrainés GLoVE duquel seul les plongements lexicaux des 1000 mots les plus fréquents (calculé à partir des exemples d'entraînement) sont affinés durant l'entraînement du module. Cet affinement a pour but d'améliorer la représentation des mots clés des questions comme *what, how, where, which, many.*
2. Des caractéristiques de jeton : Chaque paragraphe est passé à travers un étiqueteur morpho-syntaxique (*POS*) qui permet de retrouver les étiquettes grammaticales des mots p_i. Les paragraphes sont aussi passés à travers un étiqueteur de reconnaissance d'entités nommées (*NER*) pour donner à chaque jeton un étiquette d'entité n_i. Finalement, DrQA calcule la fréquence normalisée de chaque jeton (notée TF) par rapport à son paragraphe.
3. Une caractéristique de concordance exacte qui indique si le jeton p_i du passage est aussi présent dans la question : $\mathbb{1}(p_i \in q)$.
4. Un alignement du plongement de la question (*Aligned question embedding*) [96] : en plus de la précédente composante, le lecteur de paragraphe utilise une mesure de similarité plus sophistiquée : $\sum_j a_{i,j}\mathbf{E}(q_j)$ où $a_{i,j}$ est un score d'attention qui capture la similarité entre le jeton p_i et chaque jeton de la question. Spécifiquement, $a_{i,j}$ est le produit scalaire entre chaque plongement lexical de mot passé à travers une couche dense avec une unité de rectification linéaire (*Relu*) comme fonction d'activation :

$$a_{i,j} = \frac{\exp\left(\alpha(\mathbf{E}(p_i)) \cdot \alpha(\mathbf{E}(q_j))\right)}{\sum_{j'} \exp\left(\alpha(\mathbf{E}(p_i)) \cdot \alpha(\mathbf{E}(q_{j'}))\right)}$$

Avec $\alpha(\cdot)$ la couche dense avec une unité de rectification linéaire (*Relu*) comme fonction d'activation. Comparé à la caractéristique de concordance exacte, cette caractéristique permet de faire un alignement moins strict en donnant un score non nul sur des mots synonymes, comme, par exemple, entre les mots *voiture* et *véhicule*.

Ces quatre caractéristiques ont pour but d'aider le lecteur de paragraphe à apprendre à trouver le début et la fin de la réponse. Une fois la représentation $\tilde{p}$ obtenue, elle passe à travers un Bi-LSTM afin d'obtenir l'encodage final de chaque jeton p_i :

$$\{\mathbf{p}_1, ..., \mathbf{p}_m\} = \text{RNN}(\{\tilde{\mathbf{p}}_1, ..., \tilde{\mathbf{p}}_m\})$$

Ce Bi-LSTM apprend à produire un encodage qui va aider le reste du module à déterminer le début et la fin de la réponse.

Pour ce qui est de la question, elle est représentée par un simple plongement $\mathbf{q}$, qui est la somme pondérée du plongement lexical de chaque jeton q_i de la question. Ce plongement final est calculé en passant la série de représentations $\mathbf{E}(q_1), ..., \mathbf{E}(q_n)$ des jetons de la question dans un Bi-LSTM qui ne partage pas les poids du Bi-LSTM des paragraphes. La représentation obtenue $\{\mathbf{q}_1, ..., \mathbf{q}_n\}$ est alors combinée par une somme pondérée :

$$\mathbf{q} = \sum_j b_j \mathbf{q}_j$$

Où le poids b_j est le score de pertinence de chaque jeton de la question et repose sur le vecteur de poids appris $\mathbf{w}$:

$$b_j = \frac{\exp\left(\mathbf{w} \cdot \mathbf{q}_j\right)}{\sum_{j'} \exp\left(\mathbf{w} \cdot \mathbf{q}_\mathbf{j}^{'}\right)}$$

Grâce aux méthodes décrites plus haut, DrQA obtient donc un simple plongement $\mathbf{q}$ pour la question ainsi qu'une représentation de chaque jeton du passage $\{\mathbf{p}_1, ..., \mathbf{p}_m\}$.

Afin de trouver la réponse dans le passage, DrQA entraîne deux classificateurs linéaires indépendants qui prédisent respectivement le début et la fin de la réponse dans le passage. Plus spécifiquement le lecteur de paragraphe capture la similarité entre p_i et q en utilisant une couche d'attention bi-linéaire afin d'obtenir la probabilité que chaque jeton soit le début

ou la fin de la réponse :

$$\begin{aligned} P_{\text{début}}(i) &\propto \exp\left(\mathbf{p}_i \mathbf{W}_s \mathbf{q}\right) \\ P_{\text{fin}}(i) &\propto \exp\left(\mathbf{p}_i \mathbf{W}_e \mathbf{q}\right) \end{aligned}$$

DrQA est donc un système qui peut être entraîné pour la tâche de *question-réponse fermée* afin de répondre à une question avec un passage. Pour choisir la meilleure réponse dans le passage, DrQA choisit la réponse qui maximise le score $P_{\text{début}}(i) \times P_{\text{fin}}(i')$ et qui respecte la contrainte $i \leq i' \leq i + 15$ afin de limiter la taille de la réponse à 15 jetons dans le passage.

Le lecteur de paragraphe est entraîné à l'aide de différents jeux de données. Le premier, SQuAD1.1, est un jeu de données contenant des paires de question-passage avec la réponse à la question. Le second, un jeu de données personnalisé appelé *Distantly Supervised* (DS) est construit à partir des jeux de données *CuratedTREC* [97], WebQuestions [98] et WikiMovies [99] car ces jeux de données n'ont que des questions et des réponses mais pas de passage associé comme dans SQuAD1.1. À l'aide du sélecteur de page décrit précédemment, les auteurs de DrQA récupèrent un passage contenant la réponse et associent ce passage avec la question afin de l'ajouter au jeu de donnée. Ce jeu de données spécialisé a été conçu dans le but de ressembler au format de SQuAD1.1 afin de pouvoir entraîner leur module sur la tâche de *question-réponse fermée*.

En utilisant ces deux jeux de données, trois modèles différents sont entraînés selon différentes méthodes afin d'obtenir le modèle le plus performant pour la tâche de *question-réponse dans le domaine ouvert* :

- SQUAD READER : un modèle entraîné uniquement sur le jeu de données SQuAD1.1.
- FINE-TUNED READER : un modèle pré entraîné sur le jeu de données SQuAD1.1 et ensuite affiné sur le jeu de données DS.
- MULTITASK READER : un modèle entraîné conjointement sur les jeux de données SQuAD1.1 et DS.

In fine, bien que le SQUAD READER soit le modèle le plus performant dans la tâche de *question-réponse fermée*, le MULTITASK READER s'avère être le modèle le plus performant pour la tâche de *question-réponse dans le domaine ouvert*.

Une fois incorporé dans le pipeline de DrQA, le lecteur de paragraphe doit être capable de gérer des pages entières de Wikipédia alors qu'il a été conçu pour recevoir des paragraphes. Pour ce faire, chaque page est scindée en paragraphes qui sont ensuite passés dans le lecteur de paragraphe. Ainsi, le lecteur de paragraphe va faire une prédiction pour chaque paragraphe. Afin de pouvoir comparer les scores entre les différents passages, DrQA applique une fonction

exponentielle non normalisée sur tous les scores et la réponse associée au plus grand score est retenue comme réponse finale du pipeline.

3.3 Limites

DrQA est ainsi capable d'obtenir de bons résultats avec ses deux modules individuellement [5]. En effet, leur sélecteur de page, dans ses paramètres finaux, est capable de trouver au moins une page contenant la réponse pour 77.8% des questions. Pour ce qui est du lecteur de paragraphe, il obtient un score F1 de 78.8% selon le standard de référence de SQuAD1.1. Cependant, le pipeline complet ne fait pas aussi bonne figure car il parvient à trouver la bonne réponse pour seulement 29.8% des questions. Nous avons identifié deux limitations majeures qui pourraient être la source de ces faibles performances :

1. À l'étape de la recherche d'information, le sélecteur de page sélectionne des pages complètes de Wikipédia, alors que le lecteur de paragraphe est conçu pour utiliser des paragraphes. Notre hypothèse est que le fait que les deux sous-systèmes ne travaillent pas avec la même unité de longueur de document introduit du bruit dans le système. En effet, cette différence de longueur de documents fait en sorte certains paragraphes utilisés n'ont aucun rapport avec la question.
2. À l'étape de l'extraction de la réponse, le lecteur de paragraphe doit scinder les pages en paragraphes et doit rassembler et calculer un score en prenant en compte toutes les prédictions de tous les paragraphes. Or cela introduit aussi énormément de bruit, car les prédictions provenant de paragraphes qui n'ont pas de rapport avec la question sont en compétitions avec la prédiction de la bonne réponse.

Dans notre architecture, nous nous attaquons à ces deux limites. Premièrement, nous proposons de travailler directement avec des passages depuis l'étape de la recherche d'information jusqu'à l'extraction de la réponse. Deuxièmement, nous ajoutons une étape de classement entre l'étape de recherche d'information et celle de l'extraction de la réponse qui a pour but d'identifier, pour chaque question, les passages les plus pertinents et de les passer ensuite à notre extracteur de réponse tout en diminuant le nombre de passages considérés par question.

CHAPITRE 4 Re-BERT OQA

Comme mentionné plus tôt, nous avons développé un système de *question-réponse dans le domaine ouvert* Re-BERT OQA, capable de répondre à des questions factuelles en utilisant un corpus de connaissances non structuré, Wikipédia. Re-BERT OQA est composé de trois sous-systèmes : le *sélecteur de passage* (*Passage Retrieval*), l'*ordonnanceur de passage* (*Passage Ranker*) et le *lecteur de passage* (*Passage Reader*).

Avec Re-BERT OQA, nous proposons plusieurs améliorations qui viennent directement résoudre les limites de DrQA présentées à la Section 3.3 :

1. Nous avons changé l'indexation du corpus pour utiliser une indexation par passage (versus une indexation par page Wikipédia complète) pour pallier au changement de taille de document que fait DrQA entre sa recherche d'information et son extraction de réponse.
2. Nous avons ajouté une étape de classement entre le module de recherche d'information et celui d'extraction de réponses. Ce module de classement s'appuie sur l'état de l'art du TALN. L'objectif est de diminuer la charge de notre module d'extraction de réponse afin de mieux l'intégrer à notre pipeline.
3. Nous avons entraîné notre module d'extraction de réponses à reconnaître les questions auxquelles on ne peut pas répondre à l'aide d'un passage afin de permettre à notre système d'ignorer les paires de questions-passages qui ne sont pas reliées. Cela permet à notre module d'extraction de réponse de juger de la pertinence de chaque passage et augmente les chances d'éliminer un passage trompeur [15].
4. Nous avons développé notre module d'extraction de réponses en utilisant les techniques les plus récentes de l'état de l'art de la tâche de *question-réponse fermée* [13]. Ainsi, nous avons un module d'extraction de réponse qui rivalise avec la performance humaine [11, 13] sur la tâche de *question-réponse fermée*.

Notre architecture détaillée est présentée à la Figure 4.1

4.1 Le Sélecteur de passage

Dans Re-BERT OQA, le *sélecteur de passage* (Figure 4.2) est la première étape du pipeline. C'est aussi la seule étape qui n'est pas basée sur des algorithmes d'apprentissage machine. Tout comme dans le sélecteur de page de DrQA, le but de ce module est, à partir d'une

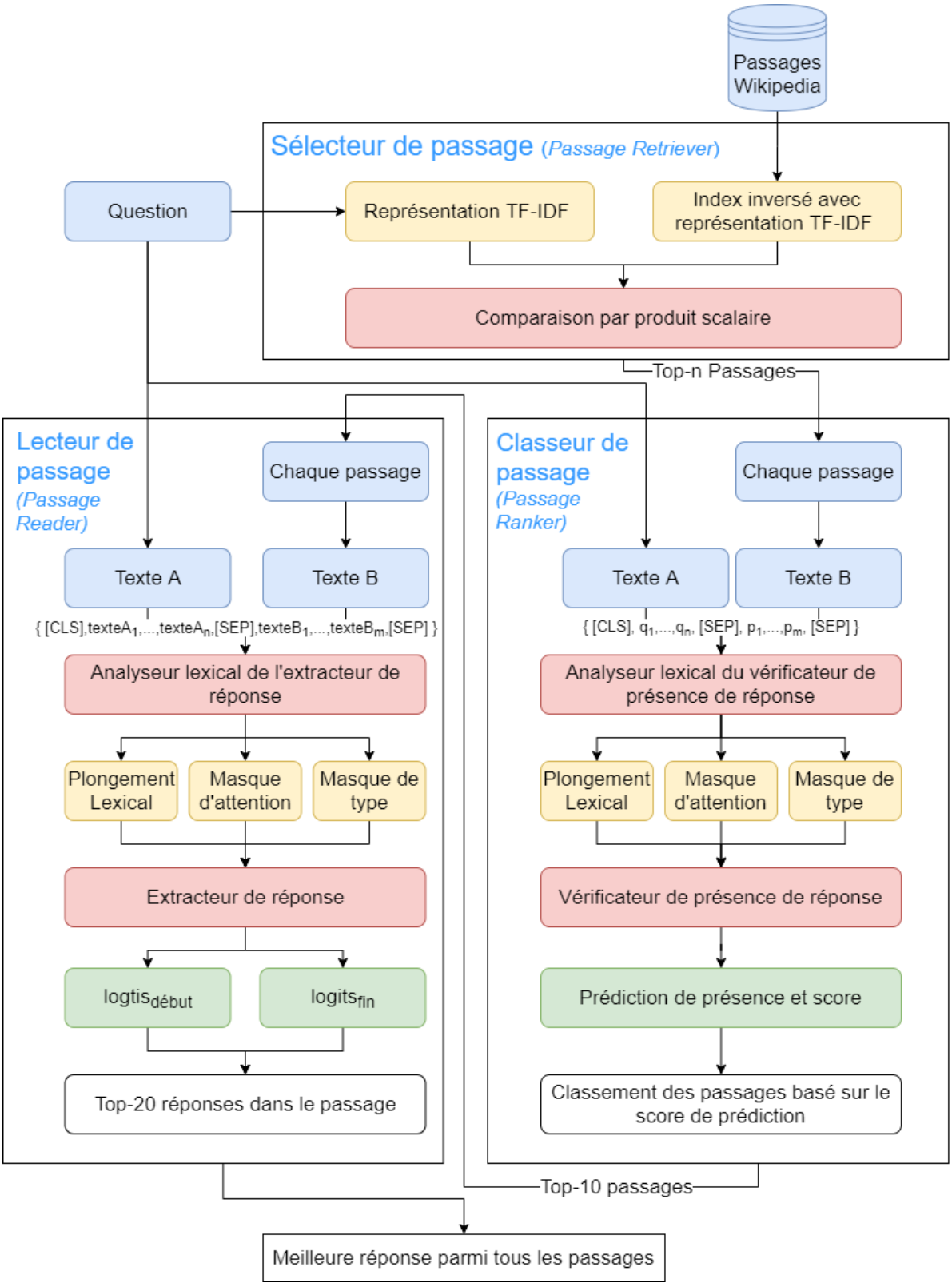


Figure 4.1 Architecture de Re-BERT OQA

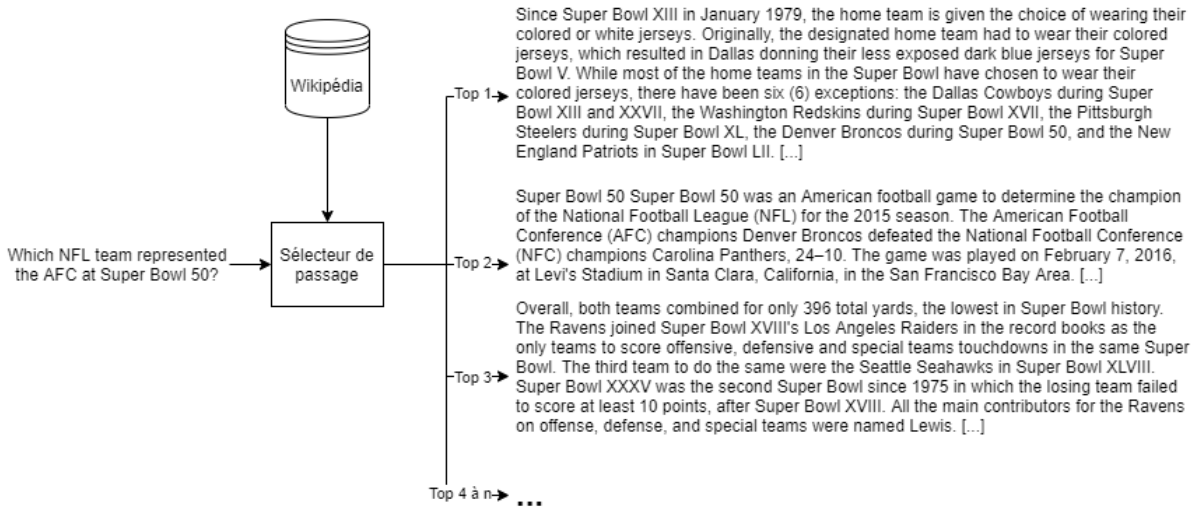


Figure 4.2 Le sélecteur de passage

question, de trouver les top-n documents pertinents dans le corpus du système, soit Wikipédia indexé par passages.

La différence majeure avec le sélecteur de page de DrQA est que notre module est paramétré pour sélectionner des passages de Wikipédia et non des pages. Afin de pouvoir préciser la notion de passage, nous définissons tout d'abord un paragraphe comme une séquence de phrases qui se termine par un ou plusieurs caractères de fin de ligne. Afin de créer un passage, nous divisons chaque article Wikipédia en paragraphe, nous regroupons les paragraphes séquentiellement en commençant du début de la page Wikipédia tant que nous ne dépassons pas un certain nombre de jetons appelé la longueur maximale de séquence. Ce paramètre est exploité par nos modules d'ordonnancement et d'extraction de réponse dans les étapes subséquentes. L'algorithme d'extraction de passages est répété jusqu'à ce que toute la page Wikipédia soit traitée. Il est important de noter qu'aucun paragraphe n'est coupé dans le but de respecter la longueur maximale de séquence, ainsi, si un paragraphe seul est plus grand que notre valeur de séquence maximale, il est ajouté tel quel à notre corpus. Ce choix, ainsi que celui de travailler avec des passages plutôt que des phrases est motivé par notre besoin d'information contextuelle [14] pour les prochaines étapes de notre pipeline.

Pour déterminer notre longueur maximale de séquence, nous avons utilisé le jeu de données SQuAD2.0 [11]. Ce dernier contient plus de 150k paires de question-passage. Plus précisément, nous analysons la longueur de tous les passages du jeu de données. Notre but est de déterminer une longueur de séquence maximale aussi proche que possible de celle utilisée dans une tâche de *question-réponse fermée*, car nos modules suivants utilisent une architecture basée sur le modèle de langue BERT qui à aussi une longueur de séquence maximale et qui est l'état

de l'art sur la tâche de *question-réponse fermée*. Ainsi, si nous arrivons à avoir des passages de taille similaire à ceux de la tâche de *question-réponse fermée*, cela facilitera l'intégration de nos modules suivants dans notre pipeline. De plus, notre hypothèse est que les passages de taille similaire à ceux de la tâche de *question-réponse fermée* seront plus adéquats pour des architectures basées sur les transformateurs que nous utilisons dans Re-BERT OQA (vs DrQA qui, au début de son pipeline, utilise des pages Wikipédia qui sont ensuite séparées en paragraphes pour la fin du pipeline).

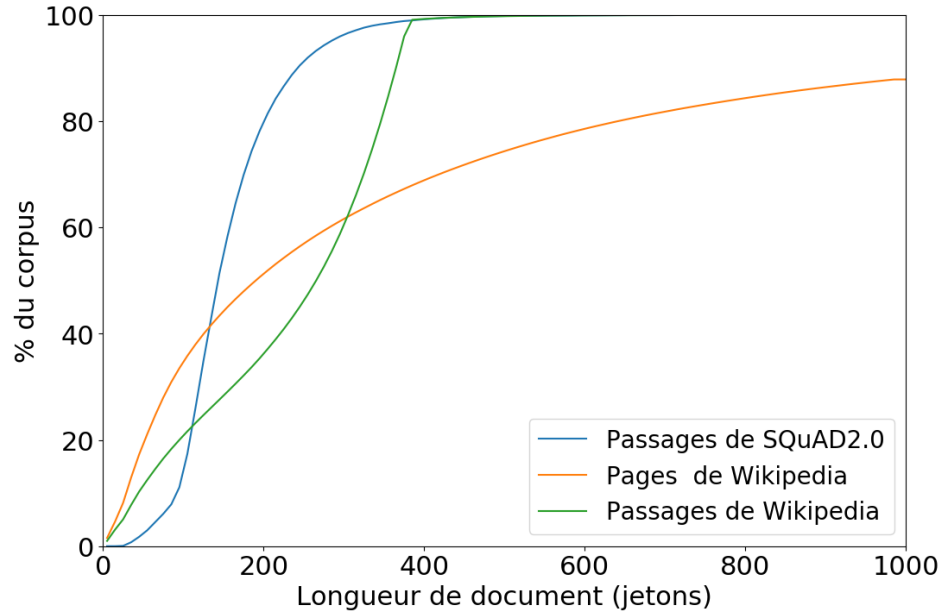


Figure 4.3 Distribution cumulative des passages du jeu de données SQuAD2.0 et des pages de Wikipédia (corpus de DrQA) comparé aux passages de Wikipédia (corpus de Re-BERT OQA) basé sur la longueur des documents.

Dans la Figure 4.3, nous étudions la longueur des documents qui composent le jeu de données SQuAD2.0 (le tracé bleu) et les pages Wikipédia (corpus de DrQA, tracé orange). La différence entre la distribution des documents dans le corpus de pages Wikipédia et la distribution attendue pour une tâche de *question-réponse fermée* selon le jeu de données de SQuAD2.0 est flagrante. Nous pouvons voir que la distribution du jeu de données de SQuAD2.0 est en dessous de 400 jetons alors que celle des pages Wikipédia a une longue traîne (*long-tailed distribution*) avec plus de 30% des documents ayant une longueur supérieure à 400 jetons. Ainsi, nous devons déterminer une longueur maximale de séquence assez grande afin que notre sélecteur de page ait toujours de bonnes performances et assez petite afin que la distribution de notre corpus soit similaire à celle du jeu de données de SQuAD2.0.

Basé sur ces observations, nous avons créé un corpus de passages de Wikipédia (tracé vert

dans la figure 4.3) avec une longueur maximale de séquence de 384 (plus de détails dans la détermination de cette valeur seront présentés à la section 5.3) générant ainsi plus de 12 millions de passages à partir des quelques 5 millions de pages Wikipédia. Comme nous pouvons le constater, la distribution de notre corpus est, en général, bien plus similaire à la distribution des passages du jeu de données SQuAD2.0 que de celle des pages de Wikipédia. En effet, nous notons que nos passages sont aussi tous principalement en dessous de 400 jetons.

Afin de récupérer le top-n passages, nous utilisons le même procédé que celui du sélecteur de page de DrQA décrit à la section 3.1. La seule différence est que chaque ligne de la matrice clairsemée représente un passage de notre corpus plutôt qu'une page Wikipédia.

4.2 L'Ordonnanceur de passage

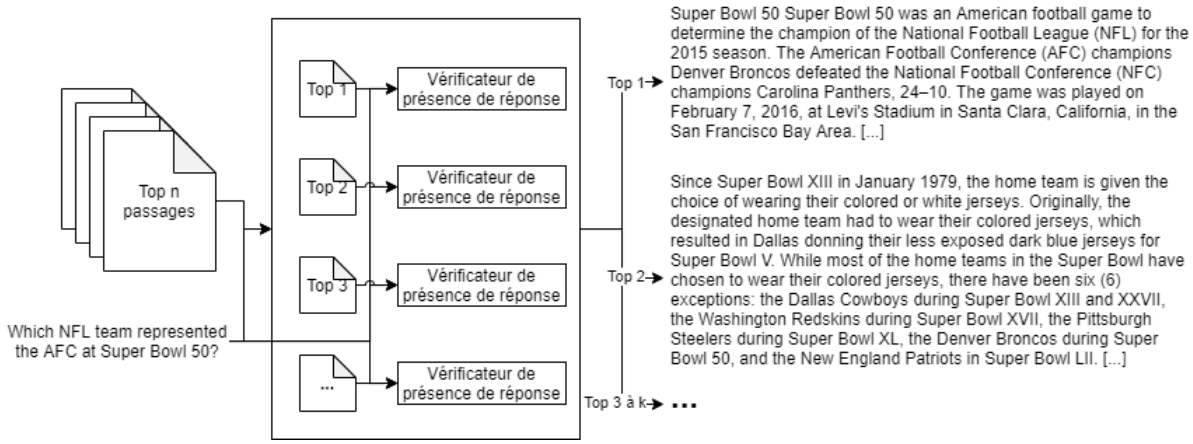


Figure 4.4 L'ordonnanceur de passage

L'*ordonnanceur de passage* est la seconde étape de notre pipeline et a pour but de classer la collection de passages obtenue par le *sélecteur de passage*. Nous avons opté pour une approche de classement par point (*pointwise ranking*) [69, 72]. Cette approche consiste à regarder un seul passage à la fois et de calculer un score en se basant sur ce passage et la question. Cette étape peut être décomposée en deux sous-modules. Le premier, le *vérificateur de présence de réponse* est responsable d'attribuer un score à une paire de question-passage. Le second, le *véritable classeur* est responsable de retourner un ensemble de k passages. Cette collection de passage est plus petite et plus pertinente que les top-n passages préalablement sélectionnés par le *sélecteur de passage*. Un résumé du rôle de l'*ordonnanceur de passage* est présenté à la Figure 4.4.

Le *vérificateur de présence de réponse* est en fait un classificateur qui est basé sur l'apprentissage

machine et prend comme entrée une question et un passage et qui retourne si oui ou non le passage contient la réponse à la question. Notre classificateur est basé sur RoBERTa [55], un des meilleurs modèles de langue pour développer un modèle spécialisé dans la classification de paires de phrases dans la tâche de question-réponse selon le standard de référence QNLI [55,60]. Ce *vérificateur de présence de réponse* est donc affiné sur une tâche de classification de paires de phrases dans le domaine de question-réponse. Nous avons construit un réseau de neurones de deux couches en aval du modèle de langue pré-entraîné RoBERTa. Ce réseau de neurones prend en entrée la sortie mise en commun (*pooled output*) de RoBERTa. Il consiste en une couche dense de la même dimension que la sortie et est suivi d'une couche de projection qui est responsable de donner la prédiction du réseau avec un score de confiance. Par la suite, nous nous appuyons sur le score de confiance afin d'évaluer la pertinence du passage selon la question.

Comme tous les modèles basés sur BERT ou RoBERTa [13, 55], notre modèle prend en entrée un vecteur de la forme $v = \{[CLS], \text{texteA}_1, ..., \text{texteA}_n, [SEP], \text{texteB}_1, ..., \text{texteB}_m, [SEP]\}$ où texteA_i et texteB_i sont le $i^{\text{ème}}$ jeton respectivement de la question de longueur n et du passage de longueur m. Afin d'uniformiser la dimension de notre entrée, nous avons fixé une longueur maximale de séquence de 384 (nous avons choisi cette valeur, car c'est le meilleur compromis que nous avons trouvé entre les performances et les limites matérielles). Tous les vecteurs d'entrée inférieurs à cette valeur seront remplis de 0 (*padding*) et tous les vecteurs supérieurs seront tronqués. Dans le cas des troncatures, nous comparons la longueur du texteA et du texteB et enlevons un jeton dans le plus grand des deux. Nous répétons ce processus jusqu'à ce que $|v| = 384$.

Une fois que nous avons ce vecteur, nous utilisons l'analyseur lexical du *vérificateur de présence de réponse* afin d'en extraire trois composantes qui seront utilisées par notre *vérificateur de présence de réponse* :

1. Le plongement lexical de chaque jeton. Cette représentation est obtenue à partir du vocabulaire du modèle pré-entraîné RoBERTa. Le vecteur résultant est lui aussi de taille 384, les jetons de remplissage sont représentés par des vecteurs nuls. À noter que durant l'entraînement du *vérificateur de présence de réponse*, l'analyseur lexical et donc les représentations des mots sont aussi affinés.
2. Le masque d'attention, un vecteur de taille 384 qui indique si le $i^{\text{ème}}$ jeton du plongement lexical est un jeton de remplissage ou non. Ce masque indique au modèle sur quels jetons du vecteur de plongement lexical il doit focaliser son attention.
3. Le masque de type, aussi un vecteur de taille 384. Ce vecteur indique si le $i^{\text{ème}}$ jeton du plongement lexical est un jeton du texteA, du texteB ou un jeton de remplissage. Ce

masque permet au modèle de différencier les deux textes en entrée durant l'entraînement et durant la prédiction. Ce masque permet donc d'aider le modèle à apprendre à faire des liens entre la question et le passage.

Durant l'affinement, ces trois vecteurs ainsi que l'étiquette de l'exemple sont donnés à notre *vérificateur de présence de réponse* et son but est de prédire l'étiquette donnée. Les 3 vecteurs d'entrée passent tout d'abord par le modèle pré-entraîné, celui-ci produit un encodage de chaque jeton de la séquence v à partir du plongement lexical de chaque jeton, de la place du jeton dans la séquence et du plongement lexical de la séquence. Nous prenons la sortie mise en commun (*pooling*) du dernier état caché du premier jeton (le jeton $[CLS]$), car nous voulons que le *vérificateur de présence de réponse* apprenne à placer les encodages importants pour la classification dans ce jeton. Par la suite, nous régularisons la sortie en appliquant un décrochage (*dropout*) pour éviter le surajustement (*overfitting*) de notre modèle et on le passe à travers une couche linéaire dense de même taille que celle de l'état caché puis dans une tangente hyperbolique. Finalement, nous appliquons encore un abandon pour ensuite projeter à travers une couche linéaire et obtenir les scores de classification (*logits*) de chaque classe :

$$\begin{aligned} in &= Pooled(RoBERTa(v)) \\ out &= Dense(Dropout(in[0])) \\ out &= tanh(out) \\ logits &= Projection(Dropout(out)) \end{aligned}$$

Le modèle s'affine en minimisant l'entropie croisée entre les logits et l'étiquette de l'exemple (qui est 1 lorsqu'il y a la réponse à la question dans le passage, 0 sinon). Pour ce faire, nous calculons le log Softmax des logits et calculons l'inverse de la log-vraisemblance que nous voulons minimiser :

$$\text{Loss}(\text{logits}, \text{e}) = \log\Big(\sum_i \exp\left(logits(i)\right)\Big) - \text{logits}(\text{e})$$

où e est l'index de la bonne prédiction (dans notre cas, 0 si le passage ne contient pas la réponse à la question, un sinon).

Durant la prédiction, nous appliquons le même réseau de neurones sauf que nous n'appliquons pas le décrochage. À partir des logits, nous appliquons un softmax pour avoir une probabilité associée à la prédiction.

Le *classeur* utilise le *vérificateur de présence de réponse* afin de sélectionner les top-k passages.

Il prend en entrée la question ainsi que les top-n passages provenant du *sélecteur de passage*. Afin d'effectuer la comparaison par point, le classeur utilise le score de confiance du *vérificateur de présence de réponse* pour attribuer un score à chaque passage par rapport à la question. Finalement, les passages associés avec les top-k meilleurs scores sont retournés. Ainsi, nous sommes capables de produire une liste ordonnée de passages basée sur leur pertinence à l'égard d'une question.

4.3 Le Lecteur de passage

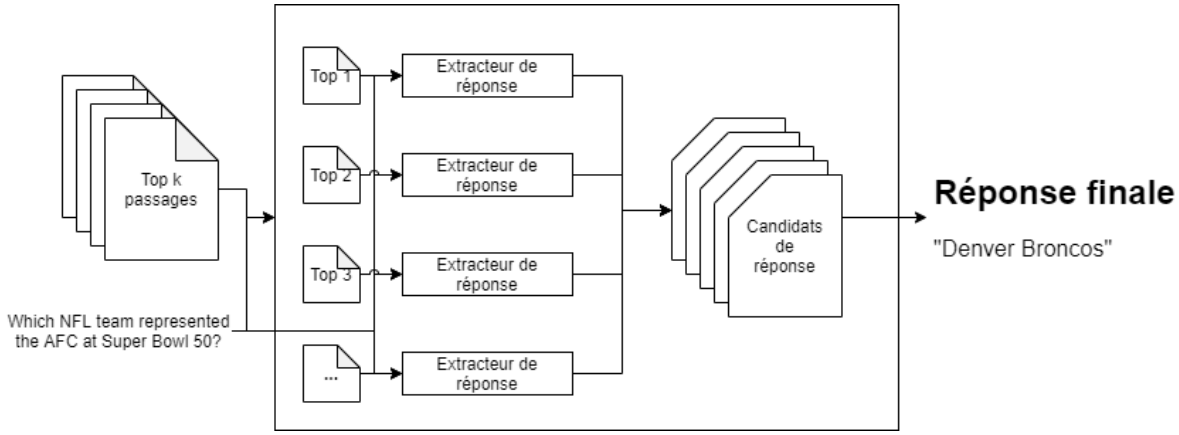


Figure 4.5 Le lecteur de passage

Le *lecteur de passage* est la dernière étape de notre pipeline. Ce module peut lui aussi être divisé en deux sous-modules. Le premier, l'*extracteur de réponse* est aussi basé sur des algorithmes d'apprentissage machine et vise à déterminer le début et la fin de la réponse dans un passage. Le second, le véritable *lecteur de passage*, utilise l'*extracteur de réponse* pour traiter les top-k passages retournés par notre *ordonnanceur de passage* à partir d'une question afin de sélectionner la réponse finale du système. Un aperçu du *lecteur de passage* est visible à la Figure 4.5.

Basé sur les récents succès dans la tâche de *question-réponse fermée* [13, 55, 61], nous avons construit un *extracteur de réponse*, c'est-à-dire un réseau de neurones au-dessus d'un modèle de langue de BERT [61] pré-entraîné pour la question-réponse et nous avons affiné notre système sur la tâche de *question-réponse fermée*. De plus, nous avons aussi affiné notre modèle afin qu'il soit capable de détecter des paires de question-passage sans réponse.

Notre *extracteur de réponse* reçoit, lui aussi, en entrée une question (texteA) et un passage (texteB). Ceux-ci sont alors combinés en une seule séquence et sont délimités par des jetons

spéciaux. Ainsi nous nous retrouvons avec une entrée de la forme

$$v = \{[CLS], q_1, ..., q_l, [SEP], p_1, ..., p_m, [SEP]\}$$

où $\{q_1, ..., q_l\}$ et $\{p_1, ..., p_m\}$ sont, respectivement, les jetons, obtenus via l'analyseur lexical de l'*extracteur de réponse*, de la question q de longueur l et du passage p de longueur m. Si $|v| < n$ où n est la valeur de la longueur maximale de séquence du système alors nous remplissons v de 0 (*padding*) de sorte que $|v| = n$. Au contraire, si $|v| > n$ nous tronquons q ou p en enlevant un jeton à la fois selon celui qui compte le plus grand nombre de jetons jusqu'à ce que $|v| = 384$.

À partir de cette séquence v, nous utilisons l'analyseur lexical de notre *extracteur de réponse* pour générer trois vecteurs de taille 384 de façon similaire à celle décrite pour notre *vérificateur de présence de réponse* (cf. section 4.2). Ces trois vecteurs sont les différentes composantes utilisées pour déterminer où est la réponse dans le passage. De la même façon que pour le *vérificateur de présence de réponse*, nous générons le plongement lexical pour chaque jeton, le masque d'attention et le masque de type. La seule différence est que, étant donné que le masque de type permet d'aider le modèle à apprendre à faire des liens entre la question et le passage, il permet aussi au modèle d'apprendre plus facilement que la réponse doit être trouvée exclusivement dans le p (le passage). Aussi, nous marquons le jeton $[CLS]$ comme provenant du passage, car c'est ce jeton que le système doit apprendre à prédire dans le cas d'une question sans réponse dans le passage.

Durant l'affinement, l'*extracteur de réponse* prend en entrée les trois vecteurs décris ci-dessus ainsi que les positions de début et de fin de la réponse dans la séquence v. À partir des trois composantes, le modèle de langue pré-entraîné BERT va produire un encodage pour chaque jeton de la séquence v. Cet encodage est construit à partir du plongement lexical en entrée, de la position du jeton dans la séquence, du plongement lexical de la séquence et des états cachés intermédiaires des transformateurs de BERT. L'encodage final est obtenu en utilisant la représentation du dernier état caché du modèle de langue. Celui-ci est ensuite projeté à travers une couche linéaire de classification. Cette couche calcule pour chaque jeton des scores de classification (*logits*) qui indiquent la probabilité que ce jeton indique le début (*d)* ou la fin (f) de la réponse :

$$in = BERT(v)$$
$$logits_d, logits_f = projection(in)$$

Ainsi, pour chaque paire de question-passage, l'*extracteur de réponse* produit deux tableaux

de taille 384. Le premier tableau contient les scores de classification que le $i^{\text{ème}}$ jeton soit le début de la réponse, réciproquement, le second tableau contient les scores de classification pour la fin de la réponse. De plus, nous utilisons les logits du premier index des deux tableaux comme score que la paire de question-passage soit sans réponse. L'objectif d'entraînement durant l'affinement est la moyenne du log-vraisemblance de la bonne position de début et de fin de réponse.

$$\text{Loss}_{\text{D}}(\text{logits}_{\text{d}}, \text{p}_{\text{d}}) = \log\Big(\sum_i \exp\left(logits_d(i)\right)\Big) - \text{logits}_{\text{d}}(\text{p}_{\text{d}})$$
$$\text{Loss}_{\text{F}}(\text{logits}_{\text{f}}, \text{p}_{\text{f}}) = \log\Big(\sum_i \exp\left(logits_f(i)\right)\Big) - \text{logits}_{\text{f}}(\text{p}_{\text{f}})$$
$$\text{Loss} = \frac{\text{Loss}_{\text{D}} + \text{Loss}_{\text{F}}}{2}$$

Avec p_d et p_f les positions respectives du début et de la fin de la bonne réponse dans le passage.

Durant la prédiction, nous utilisons les deux tableaux afin de récupérer le top-20 prédictions par passage. Pour ce faire, nous essayons toutes les combinaisons de début et de fin de réponse auxquelles nous associons un score de prédiction qui est en fait l'addition des scores de classification de début et de fin de la réponse. Ensuite, nous enlevons les prédictions qui sont incohérentes comme lorsque le début de la réponse est après la fin, ou encore lorsque le début de la réponse se trouve dans la question. Afin de normaliser les scores pour obtenir des probabilités à travers le passage, nous appliquons une fonction softmax sur tous les scores de prédiction du passage. La réponse avec la plus grande probabilité est notre prédiction finale. Cela nous sert, notamment, pour l'évaluation de notre *extracteur de réponse* dans une tâche de *question-réponse fermée*.

Pour fonctionner dans le pipeline de Re-BERT OQA, l'*extracteur de réponse* est utilisé par notre *lecteur de passage*. Pour ce faire, nous devons être capables de gérer les top-k passages provenant de notre *ordonnanceur de passage* pour chaque question. Nous utilisons donc notre *lecteur de passage* pour obtenir des prédictions avec les scores de prédiction associés pour chaque passage. Nous retirons toutes les prédictions provenant d'un passage où la meilleure prédiction pointe sur le jeton $[CLS]$, car cela indique que la paire est jugée sans réponse par l'*extracteur de réponse*. Ainsi, nous retirons toutes les prédictions faites sur un passage qui ne contient pas de réponse à la question selon notre *extracteur de réponse*. Ensuite, nous sélectionnons la prédiction associée au plus grand score de prédiction comme la réponse finale de Re-BERT OQA.

Nous avons décidé d'utiliser le score de prédiction plutôt que la probabilité de la prédiction,

Question : Where did Super Bowl 50 take place ?
Réponse : ["Santa Clara, California", "Levi's Stadium", "Levi's Stadium in the San Francisco Bay Area at Santa Clara, California."]
Prédictions du 1er passage :

1. "Denver Broncos", score de prédiction : 9.97, score normalisé : 92%
2. "the Denver Broncos", score de prédiction : 6.86, score normalisé : 4%
3. "Denver Broncos during Super Bowl 50", score de prédiction : 6.68, score normalisé : 3%

...

Prédictions du 2e passage :

1. "Santa Clara, California", score de prédiction : 14.28, score normalisé : 59%
2. "Levi's Stadium in Santa Clara, California", score de prédiction : 13.81, score normalisé : 37%
3. "Levi's Stadium", score de prédiction : 11.01, score normalisé : 2%

...

Figure 4.6 Exemple de question illustrant l'indépendance du score de prédiction des autres prédictions du même passage

car nous obtenons de meilleurs résultats ainsi. Cela peut s'expliquer par le fait que le score de prédiction est complètement indépendant du passage d'où il provient et donc des autres passages. En effet, comme mentionné plus tôt, la probabilité de prédiction est obtenue en appliquant un softmax sur l'ensemble des scores de prédictions, la rendant ainsi intrinsèque au passage d'où elle provient, alors que le score de prédiction est celui que l'*extracteur de réponse* apprend à maximiser, il est donc plus élevé plus le modèle est confiant en sa prédiction. Par exemple, considérons la question *Where did Super Bowl 50 take place ?* pour laquelle notre *extracteur de réponse* a retourné des prédictions provenant de plusieurs passages avec différents scores rapportés à la Figure 4.6. Pour alléger la charge du lecteur, nous avons seulement rapporté les scores des prédictions du passage qui aurait été choisi si on utilisait les probabilités et celui qui a été choisi en utilisant le score de prédiction. La réponse correcte est située dans le 2e passage. Comme nous pouvons le voir, les prédictions faites dans le 1er passage sont toutes erronées (elles ne correspondent pas à un élément de la réponse), mais la 1ère prédiction obtient quand même une probabilité de 92%. En revanche dans le second passage, toutes les prédictions sont bonnes ou correspondent bien à des co-références à la bonne réponse. Ce passage ainsi que ses prédictions sont donc un bien meilleur choix comparé au 1er. Cela résulte en un score de prédiction très fort, mais une probabilité plus faible due à la compétition à l'intérieur même du passage. Ainsi, le score de prédiction est une meilleure métrique dans le but de comparer des prédictions à travers plusieurs passages.

CHAPITRE 5 ÉVALUATION

Dans ce chapitre, nous allons présenter les différents jeux de données ainsi que les métriques utilisées pour l'évaluation des sous-systèmes et du pipeline complet. Finalement, nous allons présenter les résultats des évaluations de ces sous-systèmes seuls et des modules au sein du pipeline.

5.1 Jeux de données

Tableau 5.1 Statistiques sur les jeux de données utilisés durant l'entraînement et l'évaluation des différents modules de notre pipeline. Les paires positives sont des questions auxquelles nous pouvons répondre, les paires négatives ne le sont pas. Les valeurs données correspondent au nombre de paires question-passages dans chaque jeu de données

	SQuAD1.1 SQuAD$_{\text{Open}}$	PQNLI	SQuAD2.0	PSQuAD
Ensemble d'entraînement				
Paires positives	87,599	63,506	86,821	-
Paires négatives	-	63,506	43,498	-
Ensemble de test				
Paires positives	10,570	8,467	5,928	41,421
Paires négatives	-	8,467	5,945	-

Tableau 5.2 Jeux de données utilisés pour l'entraînement et l'évaluation du *vérificateur de présence de réponse* et du *lecteur de passage*

	PQNLI		SQuAD2.0		PSQuAD
	entraîn$^{\text{nt}}$	test	entraîn$^{\text{nt}}$	test	test
Vérificateur de présence de réponse	X	X			
Extracteur de réponse			X	X	X

Afin d'entraîner et d'évaluer nos différents modules et notre pipeline complet, nous utilisons plusieurs jeux de données. Dans le Tableau 5.1, nous présentons les statistiques descriptives de chaque jeu de données. Nous notons que certains jeux de données ne contiennent que des paires positives, ce choix à été fait pour rester comparable avec les différents systèmes

de *question-réponse dans le domaine ouvert*. De plus, nous résumons les cas d'usages de chaque jeu de données pour l'entraînement et l'évaluation des sous-systèmes dans la Table 5.2. Pour l'évaluation de nos modules et de Re-BERT OQA, nous utilisons le jeu de données $\text{SQuAD}_{\text{Open}}$ car c'est le jeu de données le plus répandu pour l'évaluation de cette tâche [5–8,88]. À noter que nous n'évaluons pas directement le *lecteur de passage* car son évaluation revient à l'évaluation de Re-BERT OQA. Cependant, afin d'analyser les performances de son sous-système, l'*extracteur de réponse*, nous évaluons aussi ce dernier sur PSQuAD afin de pouvoir le comparer à un autre module d'extraction de réponse. Les raisons du choix de ce jeu de données sont expliquées dans sa description à la section 5.1.2.

5.1.1 SQuAD1.1

La première version du jeu de données *Stanford Question Answering* (SQuAD1.1) [9] contient plus de 90k paires de question-passage, toutes avec réponse. Tous les passages sont tirés de Wikipédia. Ce jeu de données fournit aussi, pour chaque question, le segment du passage qui correspond exactement à la réponse. Ce jeu de données n'est pas utilisé directement dans notre système. Cependant, nous allons utiliser quelques jeux de données qui sont dérivés de celui-ci. À noter que l'ensemble de test de ce jeu de données n'est pas publiquement disponible. Nous utilisons donc l'ensemble de validation comme ensemble de test et c'est ce que nous désignons par "ensemble de test" dans la table 5.1. Dans les sections suivantes, nous présentons les jeux de données utilisés dans nos différents modules soit : le *sélecteur de passage*, l'*ordonnanceur de passage* et le *lecteur de passage*.

5.1.2 Jeux de données pour l'entraînement et l'évaluation des sous-systèmes

Notre *vérificateur de présence de réponse* et notre *extracteur de réponse* sont tous deux basés sur une approche d'apprentissage machine qui requiert donc un entraînement. Nous allons donc présenter les différents jeux de données que nous avons utilisés pour entraîner et valider ces deux sous-modules (cf. Table 5.2).

Passage question-réponse NLI (PQNLI)

Dans le but d'entraîner et de valider le *vérificateur de présence de réponse* de notre *ordonnanceur de passage*, nous avons créé un jeu de données portant sur la classification de paires de phrases dans la question-réponse inspirée du jeu de données QNLI dans le standard de référence GLUE [60]. QNLI est un jeu de données composé de paires question-phrase. La tâche reliée à ce jeu de données est alors de déterminer si la phrase contient ou non la réponse

à la question qui lui est associée. Dans notre cas, au lieu d'utiliser une simple phrase comme contexte comme dans QNLI, nous utilisons des passages. Cela augmente la complexité de la tâche d'origine, car le système doit comprendre un passage au complet plutôt qu'une seule phrase. Notre jeu de données consiste en des triplets $< Q, P, E >$ où Q est la question, P le passage et E l'étiquette indiquant si oui ou non le passage contient la réponse à la question. L'exemple 5.1 est tiré de notre jeu de données PQNLI.

Question : Of all their agreements with Hollywood producers in the 50s, which was the most iconic for ABC ?
Passage contenant la réponse : "Stanley Mouse filed a lawsuit against the producers of the film "Monsters, Inc." in 2002, alleging that the characters of Mike and Sulley were based on his drawings of "Excuse My Dust", which he unsuccessfully pitched to Hollywood producers in 1998. A Disney spokeswoman responded that the characters in Monsters, Inc were "developed independently by the Pixar and Walt Disney Pictures creative teams, and do not infringe on anyone's copyrights". Mouse now lives in Sonoma County, California where he continues to paint and create music."
Passage ne contenant pas la réponse : "Star Wars exceeded expectations in terms of profit, had a revolutionary effect on the film industry, and had an unexpected resonance as a cultural phenomenon. Lucas hoped to become independent from the Hollywood film industry, choosing to finance the sequel himself with $33 million from loans and the previous film's earnings—going against the principles of many Hollywood producers never to invest one's own money. Now fully in command of his "Star Wars" enterprise, Lucas chose not to direct the sequel because of his other production roles, including overseeing his special effects company Industrial Light & Magic (ILM) and handling the financing. Lucas offered the role of director to Irvin Kershner, one of his former professors at the USC School of Cinematic Arts. Kershner was known for smaller-scale, character-driven films, but had more recently directed the true-life drama "Raid on Entebbe" (1977) and the thriller "Eyes of Laura Mars" (1978). Kershner initially turned Lucas down, citing his belief that a sequel would never meet the quality or originality of "Star Wars". He called his agent, who immediately demanded that he take the job."

Figure 5.1 Exemple de question du jeu de données PQNLI

Tout comme dans le jeu de données QNLI, toutes les questions proviennent du jeu de données SQuAD1.1. Cependant les passages sont recueillis en utilisant notre *sélecteur de passage*

Question : What does AFC stand for ?
Passage provenant de SQuAD1.1 : Super Bowl 50 was an American football game to determine the champion of the National Football League (NFL) for the 2015 season. The American Football Conference (AFC) champion Denver Broncos defeated the National Football Conference (NFC) champion Carolina Panthers 24–10 to earn their third Super Bowl title. [...]

Figure 5.2 Exemple de question qui dépend de son passage dans SQuAD1.1

décrit à la section 4.1. Pour chaque question, nous récupérons le top-5 passages et sélectionnons au hasard un passage qui contient la bonne réponse et un qui ne la contient pas. Si, pour une question, nous ne trouvons pas de passage qui contienne la bonne réponse, nous ne l'utilisons pas. Cela nous permet de retirer les questions qui ne peuvent être comprises que grâce à leur contexte d'origine (qui est le passage qui est associé à la question dans le jeu de données SQuAD1.1) car ces questions sont dépendantes de leur passage [92] comme, par exemple, la question "*What does AFC stand for ?*". Une rapide recherche sur Wikipédia nous montre que AFC peut correspondre à "*Asian Football Confederation*" alors que si nous avions le passage associé avec la question, il est clair que la réponse est en fait la "*American Football Conference*" (voir la figure 5.2). Ainsi, pour chaque question, nous créons deux exemples, un positif et un négatif. Nos jeux de données d'entraînement et de test comptent respectivement plus de 125k et 16k paires et sont également équilibrés.

SQuAD2.0

Pour l'affinement et l'évaluation de l'*extracteur de réponse*, nous utilisons le jeu de données SQuAD2.0. Cette seconde version du jeu de données de SQuAD ajoute plus de 50k paires de question-passage sans réponse. La tâche est donc d'être capable de déterminer si la question peut être répondue ou non et d'en extraire la bonne réponse le cas échéant.

Au total, ce jeu de données contient plus de 140k paires de question-réponse étiquetées comme étant avec ou sans réponse, et là encore, toutes les paires avec réponse sont associées avec leur réponse. Tout comme pour le jeu de données SQuAD1.1, l'ensemble de test n'est pas publiquement disponible, et nous utilisons l'ensemble de validation comme ensemble de test. C'est ce que nous désignons par "ensemble de test" dans les tables 5.1 et 5.2

Nous avons décidé d'utiliser ce jeu de données pour affiner notre *lecteur de passage* pour plusieurs raisons. Premièrement, nous voulons que notre module soit capable de répondre à des questions dans n'importe quel domaine ; or, la pluralité des sujets abordés dans le jeu de données SQuAD2.0 est exceptionnelle car les questions sont posées sur différents passages provenant de plus de 500 articles différents [9]. La plupart des autres jeux de données dans la tâche de *question-réponse fermée* touchent soit un seul sujet comme WikiMovies [99] ou ne comptent pas assez d'exemples comme CuratedTREC [97] qui compte moins de 1k paires de question-passage dans son jeu de données d'entraînement. Secondement, tous les passages du jeu de données SQuAD2.0 proviennent de Wikipédia, qui est donc la même source que celle que nous utilisons dans Re-BERT OQA. Finalement, la tâche reliée à SQuAD2.0 introduit la notion de paires question-passage sans réponse et nous voulons que notre *lecteur de passage* soit capable de reconnaître ce genre de paires afin de les éliminer une fois incorporé dans

Re-BERT OQA. L'exemple 5.3 présente un passage tiré de Wikipédia avec une question où l'on trouve la réponse dans le passage (question provenant de SQuAD1.1) et une question que l'on ne peut pas répondre (question ajoutée par SQuAD2.0).

Passage : The principle of cross-cutting relationships pertains to the formation of faults and the age of the sequences through which they cut. Faults are younger than the rocks they cut ; accordingly, if a fault is found that penetrates some formations but not those on top of it, then the formations that were cut are older than the fault, and the ones that are not cut must be younger than the fault. Finding the key bed in these situations may help determine whether the fault is a normal fault or a thrust fault.
Question que l'on peut répondre : What principle relates to the formation of faults and the age of the sequences through which they cut ? (réponse : The principle of cross-cutting)
Question que l'on ne peut pas répondre : What principle helps find the difference between a normal or thrust fault ? (réponse : <No Answer>)

Figure 5.3 Exemple de question du jeu de donées SQuAD2.0

PSQuAD

Afin de vraiment évaluer un module d'extraction de réponse développé pour un système de *question-réponse dans le domaine ouvert* basé sur la recherche d'information, nous devons tout d'abord avoir un jeu de données qui contienne des exemples aussi proches que possible de ce à quoi ces modules vont être confrontés une fois incorporés dans leur pipeline.

Le jeu de données SQuAD1.1 ne serait pas un bon choix pour cette évaluation. En effet, ce jeu de données a été construit par des humains qui devaient lire des passages et créer des questions en rapport avec les passages. Ainsi, les questions sont très proches des passages pour lesquels elles ont été écrites. Cependant, dans un système de *question-réponse dans le domaine ouvert* basé sur la recherche d'information, la question est l'entrée unique du système et le système doit trouver un ou plusieurs passages pertinents parmi tout un ensemble qui peut ou non être relié à la question. En conséquence, les passages qu'un module d'extraction de réponse rencontre ne sont peut-être pas toujours les plus optimaux pour répondre à la question et sont donc plus compliqués à traiter que ceux que l'on trouve dans SQuAD1.1.

Dans la Figure 5.4, nous présentons une question avec sa réponse. De plus, nous avons pris le passage pour lequel cette question a été créée (le passage provenant de SQuAD1.1) ainsi qu'un passage que l'on pourrait rencontrer dans un système de *question-réponse dans le domaine ouvert* (le passage provenant de PSQuAD). Dans cet exemple, les mots qui correspondent exactement à des mots de la question sont indiqués en rouge, la réponse à la question est en vert, finalement, nous avons souligné en bleu les informations pertinentes pour déterminer

Question : Which NFL team won Super Bowl 50 ?
Réponse : Denver Broncos
Passage provenant de SQuAD1.1 : Super Bowl 50 was an American football game to determine the champion of the National Football League (NFL) for the 2015 season. The American Football Conference (AFC) champion Denver Broncos defeated the National Football Conference (NFC) champion Carolina Panthers 24–10 to earn their third Super Bowl title. The game was played on February 7, 2016, at Levi's Stadium in the San Francisco Bay Area at Santa Clara, California. [...]
Passage provenant de PSQuAD : After defeating the Pittsburgh Steelers in the AFC Divisional round and the New England Patriots in the AFC Championship, the Denver Broncos went on to appear in Super Bowl 50. On February 7, 2016, Harris recorded five solo tackles and a sack as the Broncos defeated the Carolina Panthers 24–10 to win Super Bowl 50. Harris didn't allow a touchdown reception in coverage from December 1, 2013 to December 15, 2015, a span that covered 35 games (including playoffs). [...]

Figure 5.4 Example de question qui est plus simple à répondre avec un passage défini dans SQuAD1.1 que dans PSQuAD

la réponse à la question. Dans l'exemple présenté, il est flagrant que le passage provenant de SQuAD1.1 est un passage beaucoup plus simple pour répondre à la question que celui provenant de PSQuAD. En effet, dans la phrase du passage provenant de PSQuAD qui répond à la question, on note la présence du nom *Broncos* qui est une coréférence à la bonne réponse *Denver Broncos* qui elle-même apparaît plus tôt dans le passage. Ainsi, le module d'extraction de réponse à besoin de résoudre cette coréférence afin de trouver la bonne réponse. Finalement, le passage provenant de PSQuAD est aussi un passage trompeur [15] car il parle aussi des accomplissements d'un des joueurs de l'équipe des Denver Broncos, qui doit être ignoré par le module d'extraction de réponse.

Pour ces raisons, et pour rendre notre modèle d'apprentissage plus robuste aux différents paragraphes, nous avons créé PSQuAD, un jeu de données de test qui nous permet d'évaluer notre *extracteur de réponse* indépendamment de tout le reste de notre système et de le comparer au lecteur de paragraphe (ou tout autre module d'extraction de réponse développé pour des systèmes de *question-réponse dans le domaine ouvert* basé sur la recherche d'information). Ce jeu de données est constitué de triplet $< Q, P, R >$ où Q est la question, P le passage et R la réponse à la question. Q et R sont tous deux tirés de l'ensemble de test du jeu de données SQuAD1.1. Pour ce qui est des passages, nous avons utilisé notre *sélecteur de passage* décrit à la section 4.1 pour trouver les top-50 passages pour chaque question. Ceux-ci sont filtrés pour éliminer les passages qui ne contiennent pas la bonne réponse et nous créons les triplets avec les passages restants. Nous obtenons alors un jeu de données de test avec 7883 questions uniques pour un total de plus de 40k triplets $< P, Q, R >$. La tâche ici est d'extraire la réponse

R de la question Q à l'aide du passage P. Comme mentionné plus tôt, ce jeu de données nous permet de nous comparer à d'autres extracteurs de réponse qui ont été développés pour un système de *question-réponse dans le domaine ouvert*, or aucun de ceux-ci n'a la capacité de juger impossible une paire question-passage. Aussi, nous avons décidé de ne pas ajouter de paires négatives pour ne pas nous donner un avantage injuste.

Bien que la tâche décrite soit celle de *question-réponse fermée*, la manière dont le jeu de données a été créé nous permet d'évaluer les performances d'un module d'extraction de réponse provenant d'un système de *question-réponse dans le domaine ouvert*. En effet, à l'inverse du jeu de données SQuAD1.1, dans PSQuAD ce sont les questions qui sont utilisées par un système de recherche d'information pour trouver un ou plusieurs passages contenant la bonne réponse.

5.1.3 SQuAD$_{\text{open}}$

Pour l'évaluation des performances de nos trois modules et de notre pipeline complet sur la tâche de *question-réponse dans le domaine ouvert*, nous utilisons SQuAD$_{\text{Open}}$.

Ce jeu de données [5] est basé sur SQuAD1.1 et permet l'évaluation d'un pipeline complet sur la tâche de *question-réponse dans le domaine ouvert*. Ainsi, dans notre recherche, il est utilisé pour l'évaluation de notre *sélecteur de passage*, notre *lecteur de passage* ainsi que notre système complet. Comme ce jeu de données est fait pour la tâche de *question-réponse dans le domaine ouvert*, il comporte uniquement les questions et les réponses tirées du jeu de données d'origine, SQuAD1.1 (Un exemple tiré de ce jeu de donnée est présenté à la figure 5.5. Plus précisément pour l'évaluation de notre *sélecteur de passage* et notre *ordonnanceur de passage*, la tâche est de trouver au moins un passage pertinent dans notre corpus pour chaque question donnée, c'est-à-dire un passage qui contient la réponse à la question. Pour l'évaluation de Re-BERT OQA, la tâche est de trouver un ou des passages pertinents pour chaque question et ensuite d'en extraire la bonne réponse.

Question : What was the population of Kenya in 2014 ?
Réponse : [45 million people, approximately 45 million]

Figure 5.5 Example de question provenant du jeu de données SQuAD$_{\text{Open}}$

5.2 Métriques & Systèmes de références

Afin d'entraîner et d'évaluer nos différents modules, nous utilisons plusieurs métriques que nous allons détailler dans la présente section. Par la suite, nous présentons aussi les systèmes auxquels nous nous comparons.

5.2.1 Métriques

Pour l'entraînement et l'évaluation, nous utilisons diverses métriques :

- Correspondance Exacte (*Exact Match* - EM) : Utilisée pour l'évaluation complète de notre pipeline ainsi que pour notre *extracteur de réponse*, cette métrique représente le pourcentage de réponses extraites qui correspondent exactement à la réponse du jeu de données.
- Rappel (*Recall*) : Utilisé pour l'évaluation du *sélecteur de passage* et de l'*ordonnanceur de passage*, cette métrique représente le pourcentage de questions pour lesquelles au moins un des passages retournés contient la réponse à la question. Dans l'évaluation du pipeline complet, cette métrique indique le pourcentage de questions pour lesquelles au moins un des passages retournés contient la réponse à la question avant que ces passages soient traités par le module d'extraction de réponse.
- (*Macro averaged*) F1-score : Cette métrique mesure le chevauchement moyen entre la réponse extraite et la vraie réponse. Nous traitons les deux réponses comme des sacs de jetons et calculons leur F1-score [9].
- Précision (*Accuracy* - Acc) Utilisé dans le développement et l'évaluation de notre *vérificateur de présence de réponse*, cette métrique représente le pourcentage de bonnes classifications des exemples du jeu de données PQNLI.

5.2.2 Systèmes de références

Afin d'évaluer et comparer les performances de nos modules et de Re-BERT OQA, nous utilisons différents systèmes.

DrQA

Étant donné que nous sommes partis de l'architecture de DrQA, il est logique que ce système, composé d'un sélecteur de page suivi d'un lecteur de paragraphe que nous avons présenté en détail au chapitre 3 soit utilisé comme système de référence (*baseline*). Dans le présent chapitre, nous l'utilisons comme point de comparaison pour notre *sélecteur de passage*, notre

lecteur de passage et finalement pour le système au complet. À noter que pour la comparaison du système final, nous prenons le système avec les paramètres recommandés, à savoir que le sélecteur de page retourne son top 5 pages Wikipédia qui sont ensuite utilisées par le lecteur de paragraphe pour extraire la réponse dans l'une des pages.

BERTserini

Ce système de *question-réponse dans le domaine ouvert* est le premier à utiliser BERT pour l'extraction de réponse. De par le fait qu'il utilise le modèle de langue BERT, c'est un système qui utilise des paragraphes de Wikipédia (plutôt que des pages). Nous utilisons ce système comme point de comparaison pour notre *ordonnanceur de passage*. En effet, BERTserini est le système qui utilise le sélecteur de document le plus performant. Finalement, nous nous comparons aussi à son système complet qui se décline en deux variantes : une qui traite 29 paragraphes et une qui traite 100 paragraphes.

Multi-Passage BERT

Finalement nous nous comparons aussi avec les deux systèmes de *question-réponse dans le domaine ouvert* proposés par [8] : BERT-RC et Multi-Passage BERT. Comme leurs noms l'indiquent, les deux systèmes utilisent le modèle de langue BERT comme base pour leur extraction de réponse. Leurs systèmes utilisent un corpus Wikipédia indexé par passages de 100 mots. Ils introduisent aussi une phase d'ordonnancement afin de réduire le nombre total de passages que leur extracteur de réponse prend en considération. Plus spécifiquement, BERT-RC est un modèle qui suit le même pipeline que Re-BERT OQA, les seules différences étant qu'il utilise des passages plus courts (seulement 100 mots) et utilise le score de leur phase d'ordonnancement comme un des paramètres pour aider à sélectionner la réponse finale. Multi-Passage BERT utilise le même principe que BERT-RC mais ajoute une étape de normalisation globale [92] sur tous les candidats de réponses. C'est grâce à l'ajout de ce mécanisme que Multi-Passage BERT obtient les meilleurs résultats de l'état de l'art pour la tâche de *question-réponse dans le domaine ouvert*.

Dans les prochaines sections, nous évaluons les modules individuellement et ensuite le pipeline complet.

5.3 Le sélecteur de passage

Notre *sélecteur de passage* utilise le corpus Wikipédia 20191101 [1] comme base de connaissance. Nous avons utilisé le script de WikiExtractor pour nettoyer le corpus. Nous avons traité chaque page Wikipédia pour en extraire les passages comme décrit à la section 4.1. Durant la création, nous séparons chaque page en paragraphes. Afin de respecter notre longueur de séquence maximale, nous comptons le nombre de jetons dans chaque paragraphe à l'aide de l'analyseur lexical pré-entraîné de BERT appelé "bert-large-whole-word-masking-finetuned-squad" qui est fourni dans la librairie *Transformers* de Hugging Face [100]. Cet analyseur lexical provient d'un modèle de BERT pré-entraîné avec des mots complètement masqués puis affinés sur le jeu de données SQuAD1.1. Nous avons choisi cet analyseur lexical, car il a été affiné sur une tâche de *question-réponse fermée*, une tâche relativement proche de la nôtre. De plus, cet analyseur lexical est assez proche de ceux utilisés dans le *vérificateur de présence de réponse* et l'*extracteur de réponse*, et donc sera capable de compter le nombre de jetons de façon similaire. Une fois que nous avons notre corpus indexé par passages, nous créons une matrice clairsemée de la même façon que pour le sélecteur de page de DrQA.

Notre première analyse porte sur l'impact du choix de la longueur de séquence maximale sur le rappel du module. Dans la Table 5.3, nous comparons les *sélecteurs de passage* avec différentes valeurs de longueur maximale de séquence variant de 300 à 500 sur le jeu de données $\text{SQuAD}_{\text{Open}}$. Nous évaluons ces différents *sélecteurs de passage* avec différents nombres de passages retournés. À première vue, nous pouvons noter qu'au-delà de 500 passages, les performances de tous les *sélecteurs de passage* sont similaires. Ce résultat indique que passés les 500 premiers passages, tous les passages pertinents pour la question on été trouvés. De plus, nous notons que nous ne gagnons pas beaucoup en performance avec le *sélecteur de passage* de longueur de séquence maximal de 500. Ce dernier résultat est des plus intéressants. En effet, le produit scalaire qui mesure la similarité entre la question et le passage que nous utilisons favorise les plus longs documents.Étant donné leur représentation en vecteur clairsemé, plus le passage est long, plus il a de chance d'avoir des termes non nuls dans le vecteur qui le représente. Ainsi, lors du produit scalaire avec le vecteur clairsemé représentant la question un passage plus long a plus de chance d'avoir un score non nul. Il semblerait donc logique qu'avec une représentation qui compte plus de jetons, le *sélecteur de passage* de longueur de séquence maximale de 500 soit beaucoup plus performant. Du fait que le gain en performance à cette étape avec un *sélecteur de passage* de longueur de séquence maximale de 500 est négligeable et aussi parce que tous nos systèmes subséquents sont basés sur l'apprentissage machine ont

1. Les corpus Wikipédia peuvent être trouvés à l'adresse suivante `https://dumps.wikimedia.org/backup-index.html`

aussi une longueur de séquence maximale de 384, nous avons décidé d'utiliser un *sélecteur de passage* paramétré avec une longueur de séquence maximale de 384.

Notre seconde analyse vise à comparer le rappel de notre *sélecteur de passage* avec le sélecteur de page de DrQA toujours sur le jeu de données SQuAD$_{\text{Open}}$. Dans la Table 5.4, nous rapportons le Rappel du sélecteur de page de DrQA et le comparons à notre *sélecteur de passage*. Comme le sélecteur de page de DrQA est paramétré pour sélectionner des pages, nous avons premièrement évalué les performances de ce dernier. Par la suite, nous avons calculé le nombre moyen de passages générés par les pages sélectionnées en utilisant notre processus décrit à la section 4.1 en utilisant une longueur de séquence maximale de 384 pour déterminer le nombre de passages que notre *sélecteur de passage* devait sélectionner. Comme nous pouvons le constater, notre *sélecteur de passage* surpasse le sélecteur de page de DrQA. En effet, lorsque l'on sélectionne 32 passages, soit 1 page Wikipédia, nous dénombrons 15% de questions en plus pour lesquelles au moins un des passages sélectionnés contient la bonne réponse. Aussi, lorsque nous comparons le sélecteur de page avec ses paramètres recommandés, soit 5 pages Wikipédia (ce qui correspond à 139 passages), notre *sélecteur de passage* surclasse son prédécesseur de 4 %. Cependant, lorsque le nombre de pages sélectionnées devient supérieur à 20 (soit 497 passages), la différence entre les deux systèmes devient négligeable.

Le fait que notre *sélecteur de passage* soit plus performant que le sélecteur de page de DrQA au début et que l'écart de performance s'amenuise avec l'augmentation du nombre de documents sélectionnés s'explique par la diversité des pages par rapport à la diversité des passages. En effet, lorsque le sélecteur de page sélectionne 1 page, toute l'information de son pipeline est limitée à cet unique article. En revanche, notre *sélecteur de passage* sélectionne 32 passages qui peuvent provenir, tout au plus, de 32 articles différents, ainsi, l'information que ses 32 passages contiennent est beaucoup plus diverse que celle d'une page Wikipédia unique. Cela explique donc le fait que l'écart entre les deux sélecteurs se réduise alors que le nombre de documents sélectionnés augmente jusqu'à ce que les deux sélecteurs de documents aient des

Tableau 5.3 Rappel des *sélecteur de passage* avec différentes valeurs de longueur séquence maximale lorsqu'on retourne entre 25 et 750 passages sur le jeu de données SQuAD$_{\text{Open}}$

Longueur de	**№ de passages sélectionnés**					
séquence max	25	75	100	250	500	750
300	66.07	76.09	78.24	83.95	87.06	88.72
384	68.09	77.80	80.02	85.01	88.04	89.42
500	69.82	79.33	81.34	86.12	88.81	90.12

performances similaires.

Dans le reste de nos expérimentations, nous rapporterons les performances de notre *sélecteur de passage* de longueur de séquence maximale de 384 paramétré pour sélectionner 32 ou 139 passages qui correspondent respectivement à 1 et 5 page Wikipédia. Nous avons fait ce choix pour rester comparables à DrQA et aux autres systèmes de *question-réponse dans le domaine ouvert*, soit : BERTserini et Multi-passage BERT.

Tableau 5.4 Comparaison du rappel entre le sélecteur de page de DrQA et notre *sélecteur de passage* de longueur de séquence maximale de 384 sur le jeu de données $\text{SQuAD}_{\text{Open}}$ lorsque le sélecteur de page sélectionne entre 1 et 40 pages Wikipédia, le nombre de passages sélectionnés par notre *sélecteur de passage* est indiqué entre parenthèses et correspond au nombre moyen de passages générés par les pages sélectionnées

Système	**Pages sélectionnées (№ moyen de passages générés)**					
	1(32)	5(139)	10(262)	20(497)	30(721)	40(938)
Sélecteur de page de DrQA	55.95	78.04	83.20	87.24	89.06	90.12
Sélecteur de passage (nous)	70.32	82.18	85.30	88.01	89.31	90.13

5.4 L'ordonnanceur de passage

La prochaine étape de notre pipeline est l'*ordonnanceur de passage*. Comme mentionné dans la Section 4.2, le but de ce module est d'ordonnancer les passages sélectionnés par notre *sélecteur de passage* en se basant sur leur pertinence par rapport à la question donnée.

Pour rappel, afin d'être capable de donner un score à chaque passage, nous avons dû développer et entraîner un classificateur, le *vérificateur de présence de réponse*. Ce modèle, basé sur RoBERTa, a été affiné à l'aide de notre jeu de données PQNLI que nous avons développé spécifiquement pour cette tâche. Durant l'entraînement, nous avons utilisé l'optimiseur de ADAM [101] avec les hyperparamètres suivant (aussi résumé dans la Table 5.5) : une longueur de séquence maximale de 384, un taux d'apprentissage (*learning rate*) de 5×10^{-6} durant 3 itérations (*epochs*) avec un échauffement (*warm-up*) linéaire de 4000 étapes et une taille de lot (*batch size*) de 8. Avec ces paramètres, notre classificateur obtient 80.6% de précision sur le jeu de données de test de PQNLI.

Par la suite, notre *vérificateur de présence de réponse* est utilisé par notre *ordonnanceur de*

Tableau 5.5 Hyperparamètres de notre *vérificateur de présence de réponse*

hyperparamètre	séquence maximale	taux d'apprentissage	itération	échauffement	taille de lot
Vérificateur de présence de réponse	384	5×10^{-6}	3	4000	8

passage dans le pipeline Re-BERT OQA afin que l'on puisse procéder à un classement par point. Nous utilisons le score de confiance du modèle pour attribuer un score à chaque passage afin d'obtenir une liste ordonnée de passages basée sur la pertinence de chaque passage par rapport à la question en entrée.

Pour évaluer l'effet de notre *ordonnanceur de passage* sur le rappel de notre système, nous utilisons encore une fois le jeu de données $\text{SQuAD}_{\text{Open}}$. Dans la Table 5.6, nous présentons les performances de notre *ordonnanceur de passage*. Tout d'abord, nous indiquons les performances de notre *sélecteur de passage* paramétré pour sélectionner le top 10 passages sans utiliser l'*ordonnanceur de passage*. Ce premier système est ici à titre indicatif comme système de référence. En effet, l'objectif ici est de rapporter les performances de notre système sans ordonnancement afin de pouvoir évaluer les performances de l'*ordonnanceur de passage*. Ensuite, nous comparons le rappel de notre *sélecteur de passage* en le paramétrant pour sélectionner le top 32 et le top 139 passages qui vont ensuite être traités par notre *ordonnanceur de passage* afin de retourner le top 10 passages. Nous avons choisi ces paramètres pour notre *sélecteur de passage* afin de toujours être comparables avec le sélecteur de page de DrQA (cf. Table 5.4). Finalement, nous indiquons aussi les performances du module de recherche d'information de BERTserini qui est le seul système publié qui rapporte aussi le rappel de son système.

Comme nous pouvons le constater, le sélecteur de passage sans ordonnancement a un rappel de 55.08%. On voit que notre *ordonnanceur de passage* améliore drastiquement la qualité de notre recherche d'information. En effet, nous gagnons jusqu'à 20% avec notre nouvelle étape d' *ordonnanceur de passage*. Il est aussi important de noter que les performances de notre *ordonnanceur de passage* sont limitées par les performances de notre *sélecteur de passage*. Ainsi, lorsque l'on doit classer 32 et 139 passages, nous ne pouvons pas excéder un rappel de 70.32% et 82.18% respectivement (cf. Table 5.4). Plus précisément, lorsque nous sélectionnons les top 10 passages à partir du top 32 préalablement sélectionné par notre *sélecteur de passage*, nous obtenons un rappel de 68.43% alors que notre maximum est de 70.32% soit moins de 2% d'écart. De façon similaire, à partir de 139 passages, pour obtenir les top 10 passages nous obtenons un rappel de 77.26% soit une diminution de moins de 5% du rappel que

l'on avait avec 139 passages par question. Cela implique que pour environ 5% des questions, l'*ordonnanceur de passage* juge le ou les passages contenant la réponse comme moins pertinents que d'autres qui ne la contiennent pas. Ainsi, nous pouvons dire que, même s'il n'est pas parfait, notre ensemble *sélecteur de passage* + *ordonnanceur de passage* est tout de même résilient au bruit introduit par des passages qui sont soit hors contexte ou trompeurs.

Finalement, lorsque l'on compare notre meilleur système de recherche d'information (*sélecteur de passage* top-139 + *ordonnanceur de passage*) à celui de celui de BERTserini [2] [7] paramétré pour retourné 10 passages, nous observons que, nous avons un meilleur rappel par plus de 7%. Ainsi, il est clair que l'ajout de l'*ordonnanceur de passage* à notre recherche d'information est bénéfique à la qualité des passages qui sont sélectionnés. Dans Re-BERT OQA, notre *ordonnanceur de passage* est paramétré pour passer le top 10 passages à notre *lecteur de passage* à partir des passages sélectionnés par le *sélecteur de passage*.

Tableau 5.6 Rappel des différentes combinaisons du *sélecteur de passage* suivi du *ordonnanceur de passage* sur le jeu de données $\text{SQuAD}_{\text{Open}}$ comparé au module de recherche d'information de BERTserini [4]. Pour chaque système, la sortie finale est le top 10 de passages par question

Système	Rappel (%)
Sélecteur de passage (Pas d'ordonnancement, top-10)	57.08
Sélecteur de passage top-32 + *Ordonnanceur de passage*	68.42
Sélecteur de passage top-139 + *Ordonnanceur de passage*	77.26
BERTserini top-10 paragraphes[2] [7]	70

5.5 Le lecteur de passage

Finalement notre dernier module est le *lecteur de passage*. Son but est d'extraire la réponse à la question donnée à partir des documents retournés par l'*ordonnanceur de passage*.

Comme indiqué dans la section 4.3, nous avons développé un module de lecture de passage qui s'appuie sur un *extracteur de réponse* basé sur l'état de l'art de la tâche de *question-réponse fermée*. Cet *extracteur de réponse* est en fait un simple réseau de neurones à propagation en avant (*Feedforward Network*) au dessus d'un modèle pré-entraîné de BERT [13] afin de classifier chaque jeton en un jeton de début de réponse ou de fin de réponse. Ce modèle a été affiné sur le jeu de données SQuAD2.0 pour lequel on utilise les métriques EM et F1-score.

2. Performances prise de l'article de BERTserini
4. BERTserini est le seul système qui a publié son rappel

Pour l'entraînement, nous utilisons l'optimiseur ADAM [101] avec un taux d'apprentissage de 3×10^{-5}, une longueur maximale de séquence de 384 et une taille de lot de 8 pour 2 itérations. Ces hyperparmètres sont rapportés à la Table 5.7. Avec ces hyperparamètres, notre *extracteur de réponse* obtient un score EM de 80.90 % et un F1-score de 84% sur l'ensemble de test de SQuAD2.0. Comme nous l'avons mentionné à la section 5.1.2, nous utilisons l'ensemble de validation comme ensemble de test car ce dernier n'est pas disponible. Cependant, nous obtenons des performances qui sont dans le même ordre de grandeur que celles que l'on retrouve sur le standard de référence [5]. Ces résultats indiquent que nous avons bien réussi à reproduire l'extracteur de réponse basé sur le modèle de langue BERT.

Tableau 5.7 Hyperparamètres de notre *extracteur de réponse*

hyperparamètre	séquence maximale	taux d'apprentissage	itération	taille de lot
Extracteur de réponse	384	3×10^{-5}	2	8

Afin d'évaluer et comparer notre *extracteur de réponse* au *multitask reader* de DrQA, nous avons évalué les deux systèmes selon la métrique EM sur le jeu de données PSQuAD décrit à la section 5.1.2. Les résultats de cette comparaison sont rapportés à la table 5.8. Nous observons que nous sommes plus performants que le *multitask reader* de DrQA de 5% avec la métrique EM. Cependant, nous notons que notre *extracteur de réponse* n'obtient pas des résultats aussi bons qu'on l'espérait. En fait, l'écart entre ces résultats et ceux que notre modèle a obtenu sur l'ensemble d'évaluation de SQuAD2.0 (un score EM de 80.90%) reflète aussi la difficulté qu'est l'extraction de réponse dans la tâche de *question-réponse dans le domaine ouvert*. Cela confirme ce que nous avions préalablement expliqué à la Section 5.1.2, c'est-à-dire, que le genre de passages obtenu par un module de recherche d'information est beaucoup plus difficile à traiter que les passages provenant du jeu de données SQuAD2.0. En effet, rappelons que les questions provenant du jeu de données SQuAD2.0 ont été créées par des humains après avoir lu un passage, alors que pour la tâche de *question-réponse dans le domaine ouvert*, les passages sont obtenus en utilisant la question comme entrée d'un module de recherche d'information. Cet écart de performance confirme également la nuance de tâches que nous avions mentionnée dans la Section 5.1.2 entre la tâche de *question-réponse fermée* et la tâche d'extraction de réponse pour les systèmes de *question-réponse dans le domaine ouvert* et s'explique par le fait que les deux modèles ont été entraînés sur une tache de *question-réponse fermée*.

5. comparé aux résultats de BERT (single model) -Google AI Language team- provenent du classement de SQuAD2.0's disponible à `https://rajpurkar.github.io/SQuAD-explorer/`

Étant donné que pour évaluer les performances de notre *lecteur de passage* sur la tâche de *question-réponse dans le domaine ouvert*, il nous faut au minimum un module de recherche d'information, l'évaluation de ce module revient en fait à l'évaluation de Re-BERT OQA, notre système complet.

Tableau 5.8 Comparaison de notre *extracteur de réponse* et du lecteur de paragraphe de DrQA sur le jeu de données PSQuAD

Système	EM (%)
multitask reader de DrQA	33.8
Extracteur de réponse (nous)	39.2

5.6 Système complet

Nos trois modules précédemment décrits ont été mis ensemble afin de former Re-BERT OQA. Cette partie correspond donc aussi à une évaluation externe de nos différents modules, puisqu'ils sont intégrés dans un système complet.

Dans la Table 5.9, nous comparons les performances des différents systèmes de *question-réponse dans le domaine ouvert* dont, notamment, Multi-Passage BERT [8], l'état de l'art actuel dans la tâche de *question-réponse dans le domaine ouvert*, avec notre pipeline Re-BERT OQA sur le jeu de données $\text{SQuAD}_{\text{Open}}$. Nous rappelons que DrQA utilise un corpus Wikipédia indexé par pages alors que BERTserini utilise un corpus Wikipédia indexé par des paragraphes, finalement Multi-Passage BERT et Re-BERT OQA utilisent un corpus Wikipédia indexé par des passages. Aussi, nous rapportons les performances de notre pipeline selon différents paramètres incluant le système sans l'*ordonnanceur de passage*. En plus du score EM du pipeline complet, nous spécifions (lorsque disponible) le rappel de chaque système avant le module d'extraction de réponse. Nous rapportons ces deux métriques, car l'écart entre le score EM et le rappel est un bon indicateur de l'efficacité du module d'extraction de réponse à l'intérieur de son pipeline.

Tout d'abord, lorsque l'on compare Re-BERT OQA avec et sans le module de classement, nous observons que notre *ordonnanceur de passage* améliore considérablement les performances de notre pipeline. Aussi, lorsque nous nous attardons sur la différence entre le score EM et le rappel, nous voyons que le fait d'ajouter notre *ordonnanceur de passage* dans le pipeline permet d'améliorer l'efficacité de notre *lecteur de passage*.

Ensuite, nous comparons les résultats des pipelines avec approximativement la même quantité

Tableau 5.9 Résultats sur le jeu de données SQuAD$_{\text{Open}}$

Modèle	Sélection de documents	Ordonnancement	EM (%)	Rappel (%)
DrQA [5]	5 pages	-	29.4	78.0
BERTserini [7]	29 paragraphes	-	36.6	75.8
BERTserini [7]	100 paragraphes	-	38.6	85.8
BERT-RC [8]	100 passages	30 passages	45.4	-
Multi-Passage BERT [8]	100 passages	30 passages	53.0	-
Re-BERT OQA	10 passages	(Pas d'ordonnancement)	30.5	57.1
Re-BERT OQA	32 passages	10 passages	40.7	68.4
Re-BERT OQA	139 passages	10 passages	46.7	77.2

de texte traité après la sélection de documents. Dans un premier temps, nous comparerons DrQA après la sélection de 5 pages, BERTserini avec 29 paragraphes et notre système avec 32 passages, par la suite, nous comparerons BERTserini après la sélection de 100 paragraphes, les deux modèles de Multi-Passage BERT avec 100 passages et notre système avec 139 passages.

En premier lieu, nous voyons que Re-BERT OQA obtient de meilleures performances que les systèmes de DrQA et BERTserini selon leur score EM tout en ayant un rappel plus bas que ces derniers. Cela indique que notre système est bien plus précis que les pipelines existants et que notre module d'extraction de réponse est plus efficient au sein de notre pipeline que les autres modules d'extraction de réponse au sein des leurs. En effet, nous améliorons BERTserini 29 paragraphes par 4% en termes de score EM. Le rappel plus bas de notre système indique que pour plus de questions, aucun des 10 passages ne contient la réponse. Rappelons que notre approche d'ordonnancement est limitée par la performance de notre *sélecteur de passage* qui a un rappel de 70.32% (voir Table 5.4 lorsqu'il sélectionne 32 passages). Ainsi, notre approche ne peut pas obtenir un rappel aussi grand que celui des autres systèmes, cependant nous avons beaucoup moins de passages qui vont être traités par notre *lecteur de passage* que dans les deux autres systèmes. Ainsi, il est assez logique que pour une plus petite quantité de texte nous ayons un rappel plus petit.

En second lieu, nous considérons les top-139 passages sélectionnés par notre *ordonnanceur de passage*, qui sont par la suite ordonnés et à partir desquels on ne garde que les top 10 passages pour notre *lecteur de passage*. Nous obtenons alors un score EM de 46.72% soit une amélioration de 8.1% par rapport à BERTserini avec une sélection de 100 paragraphes, et une légère amélioration de 1.1% par rapport à BERT-RC. Pour la même raison mentionnée plus tôt, notre rappel ne peut excéder celui de BERTserini. Cependant, notre rappel, qui est

8.6% inférieur à celui de BERTserini reste tout de même très compétitif contenu du fait que notre rappel ne se base que sur 10 passages alors que celui de BERTserini se base sur 100 paragraphes. Pour ce qui est de notre score EM, notre amélioration par rapport à BERTserini et BERT-RC indique clairement que notre pipeline, notamment grâce à l'*ordonnanceur de passage*, permet de sélectionner de meilleurs passages pour notre *lecteur de passage*. En fait, notre ensemble Re-BERT OQA qui n'utilise rien d'autre que les candidats de réponses pour sélectionner la réponse finale obtient un meilleur score EM que BERT-RC qui utilise un post-traitement pour fusionner les candidats identiques et qui prend en compte le score d'ordonnancement de chaque passage.

Finalement, notre système obtient de moins bonnes performances que Multi-Passage BERT. Or, Multi-Passage BERT est en fait le modèle BERT-RC auquel une seconde étape de post-traitement est ajoutée. Cette étape a pour but de calculer une normalisation globale [92] à travers tous les candidats de réponses de tous les passages. Ainsi, ce système n'obtient pas de meilleures performances parce qu'il est capable de fournir des passages plus pertinents à son module d'extraction de réponse mais parce qu'il utilise une façon de choisir sa réponse finale plus sophistiquée.

CHAPITRE 6 CONCLUSION

Dans ce dernier chapitre, nous résumons les différents travaux que nous avons accomplis afin de pouvoir mettre en place notre système, Re-BERT OQA. Par la suite, nous présentons les limites de notre système et finalement, nous concluons avec les futurs axes de recherches qui pourraient être potentiellement bénéfiques pour l'amélioration des performances de Re-BERT OQA.

6.1 Synthèse des travaux

En résumé, ce mémoire a pour but de présenter Re-BERT OQA, un système développé pour la tâche de *question-réponse dans le domaine ouvert*, c'est-à-dire un système capable de répondre à n'importe quelle question factuelle en langue naturelle en anglais.

Pour ce faire, nous avons développé un pipeline en trois étapes. Notre premier module, le *sélecteur de passage*, prend en entrée la question et cherche dans le corpus Wikipédia préalablement indexé par passages et représenté à l'aide de vecteurs clairsemés [17]. Le *sélecteur de passage* a pour but de de déterminer une centaine de passages qui pourraient être pertinents pour la question parmi le corpus Wikipédia.

Notre second module, l'*ordonnanceur de passage*, a pour but de raffiner la sélection de passage. En prenant en entrée la question ainsi que la centaine de passages de l'étape précédente, ce module s'appuie sur des techniques d'apprentissage pour effectuer l'ordonnancement des passages et implémente un ordonnanceur par point [69, 72]. Notre fonction de classification est obtenue grâce à notre *vérificateur de présence de réponse*, un classificateur qui s'appuie sur le modèle de langue Roberta [55] pour se spécialiser dans la tâche d'implication textuelle dans le domaine de la question-réponse. Le *vérificateur de présence de réponse* prend donc en entrée une question et un passage et assigne un score en fonction de la pertinence du passage par rapport à la question. L'*ordonnanceur de passage* utilise donc le *vérificateur de présence de réponse* pour donner un score à chaque passage et déterminer le top-10 passages par question.

Finalement, notre troisième et dernier module, le *lecteur de passage*, a pour but d'extraire la réponse d'un des 10 passages sélectionnés. Pour ce faire, il utilise notre *extracteur de réponse*, un modèle basé sur BERT [13] et affiné sur la tâche de question-réponse avec exemples négatifs [11]. L'*extracteur de réponse* prend en entrée un passage et une question et en extrait des candidats de réponses. Ensuite, le *lecteur de passage* considère tous les candidats de tous les passages et identifie parmi ceux-ci la réponse finale à la question.

Ainsi, nous avons conçu un système de *question-réponse dans le domaine ouvert* en 3 étapes qui nous permet d'améliorer la pertinence des passages traités par notre module d'extraction de réponse. Notre système, Re-BERT OQA, obtient ainsi de bonnes performances par rapport à celles de l'état de l'art, même si elles sont en-deça de celles d'un des systèmes les plus récents, Multi-Passage BERT [8]. Notre solution présente aussi des limites qui sont détaillées dans la section suivante.

6.2 Limites de la solution proposée

Bien que notre système obtienne de bonnes performances sur les jeux de données de référence, il présente quand même des limites.

Tout d'abord au niveau du *sélecteur de passage*, notre représentation des documents est faite avec un modèle vectoriel qui utilise un TF-IDF sur des bi-grammes afin d'obtenir un vecteur clairsemé. Bien que cette technique nous permette un assez bon rappel sur l'ensemble de notre système, il va de soi que ses performances ont des répercussions immédiates sur les performances du reste du système. Un meilleur rappel au niveau du *sélecteur de passage* permettrait au reste du pipeline d'obtenir de meilleurs résultats.

De plus, nos deux systèmes utilisant une approche neuronale sont basés sur des modèles de langue qui, au moment de la conception, étaient les meilleurs dans le domaine ; or aujourd'hui, ALBERT [61] semble être un meilleur choix de modèle de langue sur lequel se baser, que ce soit pour notre *vérificateur de présence de réponse* que pour notre *extracteur de réponse*. Cependant, par limite de temps et de ressources matérielles, nous n'avons pas pu développer de nouveaux modèles basés sur ce modèle de langue.

Finalement, beaucoup des techniques utilisées dans notre solution proviennent de recherches très récentes dans le domaine du TALN. Il nous a donc fallu changer d'approches pour certaines parties comme l'*extracteur de réponse* où l'utilisation d'un modèle de langue augmente grandement les performances du système. Ainsi, une partie du temps investi dans cette recherche a été dédié à comprendre et apprendre à utiliser ces nouvelles technologies.

6.3 Améliorations futures

Les limites présentées dans la section 6.2 démontrent qu'il y a plusieurs améliorations possibles que l'on peut apporter à notre solution.

Premièrement au niveau du *sélecteur de passage*, plusieurs axes de recherches peuvent être approfondis. Tout d'abord, il serait intéressant de voir l'impact de l'algorithme d'extraction de

passage sur le rappel du système. En effet, dans nos recherches nous construisons les passages à partir de paragraphes ; or il pourrait être bénéfique d'utiliser des phrases à la place. De plus, à la lumière des recherches de [88], une représentation contextualisée par vecteur clairsemé où le poids de chaque n-gramme est calculé dynamiquement plutôt qu'avec un traditionnel TF-IDF permettrait là aussi d'améliorer le rappel de la solution. Aussi, l'étape de sélection du document pourrait aussi tirer avantage des représentations par vecteur dense obtenu via un modèle de langue comme ALBERT [61] afin d'obtenir une meilleure représentation des passages et de la question et donc d'améliorer aussi le rappel de la solution.

Un autre axe de recherche qui serait intéressant serait l'amélioration de l'*extracteur de réponse*. En effet, bien que notre *extracteur de réponse* obtienne de bons résultats sur le standard de référence SQuAD2.0 [11], ses résultats chutent sur PSQuAD, un jeu de données conçu pour évaluer un système de question-réponse avec des paramètres de *question-réponse dans le domaine ouvert*. Cela s'explique par le fait que notre *extracteur de réponse* est entraîné sur un jeu de données de *question-réponse fermée* où un humain crée des questions sur un passage qu'il vient de lire. Ainsi le vocabulaire entre la question et le passage est généralement assez proche. Or dans un système de *question-réponse dans le domaine ouvert*, les passages sont trouvés par une recherche d'information, ainsi il arrive assez souvent que les passages ne soient pas aussi proches de la question (comme nous l'avions montré à la section 5.1.2). Ainsi, il pourrait être bénéfique d'avoir un jeu de données plus orienté sur la tâche de *question-réponse dans le domaine ouvert* afin d'entraîner un extracteur de réponse plus performant une fois intégré dans le pipeline.

Aussi, comme mentionné dans la Section 6.2, le modèle de langue ALBERT [61] semble le plus performant des modèles de langue pré-entraîné. Il serait donc pertinent de l'utiliser comme base autant dans notre *vérificateur de présence de réponse* que pour notre *extracteur de réponse*.

Finalement, la sélection de la réponse parmi les candidats provenant de différents passages est une sous-tâche qui commence à être explorée, notamment par [8]. Ainsi, un axe de recherche serait l'amélioration de la sélection de la réponse finale en utilisant la technique de normalisation globale [92] sur les scores des candidats de réponse.

RÉFÉRENCES

[1] B. F. Green Jr, A. K. Wolf, C. Chomsky et K. Laughery, "Baseball : an automatic question-answerer," dans *Papers presented at the May 9-11, 1961, western joint IRE-AIEE-ACM computer conference*, 1961, p. 219–224.

[2] W. A. Woods, "Lunar rocks in natural english : explorations in natural language question answering," 1977.

[3] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak et Z. Ives, "Dbpedia : A nucleus for a web of open data, in 'the semantic web', vol. 4825 of lecture notes in computer science," 2007.

[4] D. Jurafsky et J. H. Martin, "Speech and language processing (3rd ed. draft)," 2019, unpublished Manuscript, Available at https://web.stanford.edu/~jurafsky/slp3/ (2020/04/07).

[5] D. Chen, A. Fisch, J. Weston et A. Bordes, "Reading wikipedia to answer open-domain questions," *CoRR*, vol. abs/1704.00051, 2017. [En ligne]. Disponible : http://arxiv.org/abs/1704.00051

[6] S. Wang, M. Yu, X. Guo, Z. Wang, T. Klinger, W. Zhang, S. Chang, G. Tesauro, B. Zhou et J. Jiang, "R^3 : Reinforced reader-ranker for open-domain question answering," *CoRR*, vol. abs/1709.00023, 2017. [En ligne]. Disponible : http://arxiv.org/abs/1709.00023

[7] W. Yang, Y. Xie, A. Lin, X. Li, L. Tan, K. Xiong, M. Li et J. Lin, "End-to-end open-domain question answering with bertserini," *CoRR*, vol. abs/1902.01718, 2019. [En ligne]. Disponible : http://arxiv.org/abs/1902.01718

[8] Z. Wang, P. Ng, X. Ma, R. Nallapati et B. Xiang, "Multi-passage bert : A globally normalized bert model for open-domain question answering," *arXiv preprint arXiv :1908.08167*, 2019.

[9] P. Rajpurkar, J. Zhang, K. Lopyrev et P. Liang, "Squad : 100,000+ questions for machine comprehension of text," *arXiv preprint arXiv :1606.05250*, 2016.

[10] L. Hirschman, M. Light, E. Breck et J. D. Burger, "Deep read : A reading comprehension system," dans *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*. College Park, Maryland, USA : Association for Computational Linguistics, juin 1999, p. 325–332. [En ligne]. Disponible : https://www.aclweb.org/anthology/P99-1042

[11] P. Rajpurkar, R. Jia et P. Liang, "Know what you don't know : Unanswerable questions for squad," *arXiv preprint arXiv :1806.03822*, 2018.

[12] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser et I. Polosukhin, "Attention is all you need," dans *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan et R. Garnett, édit. Curran Associates, Inc., 2017, p. 5998–6008. [En ligne]. Disponible : http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf

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

[14] S. Min, V. Zhong, R. Socher et C. Xiong, "Efficient and robust question answering from minimal context over documents," dans *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1 : Long Papers)*. Melbourne, Australia : Association for Computational Linguistics, juill. 2018, p. 1725–1735. [En ligne]. Disponible : https://www.aclweb.org/anthology/P18-1160

[15] R. Jia et P. Liang, "Adversarial examples for evaluating reading comprehension systems," *CoRR*, vol. abs/1707.07328, 2017. [En ligne]. Disponible : http://arxiv.org/abs/1707.07328

[16] S. Hochreiter et J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, p. 1735–80, 12 1997.

[17] G. Salton, A. Wong et C. S. Yang, "A vector space model for automatic indexing," *Commun. ACM*, vol. 18, nº. 11, p. 613–620, nov. 1975. [En ligne]. Disponible : https://doi.org/10.1145/361219.361220

[18] U. Cambridge, "Introduction to information retrieval," 2009.

[19] K. S. Jones et P. Willett, *Readings in information retrieval*. Morgan Kaufmann, 1997.

[20] J. B. Lovins, "Development of a stemming algorithm," *Mech. Transl. Comput. Linguistics*, vol. 11, nº. 1-2, p. 22–31, 1968.

[21] J. Plisson, N. Lavrac, D. Mladenic *et al.*, "A rule based approach to word lemmatization," dans *Proceedings of IS*, vol. 3, 2004, p. 83–86.

[22] C. E. Shannon, "A mathematical theory of communication," *The Bell system technical journal*, vol. 27, nº. 3, p. 379–423, 1948.

[23] A. Rajaraman et J. D. Ullman, *Data Mining*. Cambridge University Press, 2011, p. 1–17.

[24] Z. S. Harris, "Distributional structure," *Word*, vol. 10, nº. 2-3, p. 146–162, 1954.

[25] J. Pennington, R. Socher et C. D. Manning, "Glove : Global vectors for word representation," dans *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, p. 1532–1543.

[26] T. Mikolov, K. Chen, G. Corrado et J. Dean, “Efficient estimation of word representations in vector space,” *arXiv preprint arXiv :1301.3781*, 2013.

[27] A. Joulin, E. Grave, P. Bojanowski, M. Douze, H. Jégou et T. Mikolov, “Fasttext. zip : Compressing text classification models,” *arXiv preprint arXiv :1612.03651*, 2016.

[28] M. Baroni, G. Dinu et G. Kruszewski, “Don’t count, predict ! a systematic comparison of context-counting vs. context-predicting semantic vectors,” dans *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1 : Long Papers)*. Baltimore, Maryland : Association for Computational Linguistics, juin 2014, p. 238–247. [En ligne]. Disponible : https://www.aclweb.org/anthology/P14-1023

[29] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado et J. Dean, “Distributed representations of words and phrases and their compositionality,” dans *Advances in neural information processing systems*, 2013, p. 3111–3119.

[30] R. E. Bellman, *Adaptive control processes : a guided tour*. Princeton university press, 2015, vol. 2045.

[31] J. Coates et D. Bollegala, “Frustratingly easy meta-embedding – computing meta-embeddings by averaging source word embeddings,” dans *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics : Human Language Technologies, Volume 2 (Short Papers)*. New Orleans, Louisiana : Association for Computational Linguistics, juin 2018, p. 194–198. [En ligne]. Disponible : https://www.aclweb.org/anthology/N18-2031

[32] Y. Liu, Z. Liu, T.-S. Chua et M. Sun, “Topical word embeddings,” dans *Twenty-ninth AAAI conference on artificial intelligence*, 2015.

[33] A. Muromägi, K. Sirts et S. Laur, “Linear ensembles of word embedding models,” *arXiv preprint arXiv :1704.01419*, 2017.

[34] D. Kiela, C. Wang et K. Cho, “Dynamic meta-embeddings for improved sentence representations,” *arXiv preprint arXiv :1804.07983*, 2018.

[35] M. Artetxe, G. Labaka, I. Lopez-Gazpio et E. Agirre, “Uncovering divergent linguistic information in word embeddings with lessons for intrinsic and extrinsic evaluation,” *arXiv preprint arXiv :1809.02094*, 2018.

[36] J. L. Elman, “Finding structure in time,” *Cognitive Science*, vol. 14, n°. 2, p. 179–211, 1990. [En ligne]. Disponible : https://onlinelibrary.wiley.com/doi/abs/10.1207/s15516709cog1402_1

[37] F. A. Gers, N. N. Schraudolph et J. Schmidhuber, “Learning precise timing with lstm recurrent networks,” *Journal of machine learning research*, vol. 3, n°. Aug, p. 115–143, 2002.

[38] M. Schuster et K. Paliwal, “Bidirectional recurrent neural networks,” *Signal Processing, IEEE Transactions on*, vol. 45, p. 2673 – 2681, 12 1997.

[39] R. Al-Rfou, D. Choe, N. Constant, M. Guo et L. Jones, “Character-level language modeling with deeper self-attention,” dans *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, p. 3159–3166.

[40] I. Sutskever, O. Vinyals et Q. V. Le, “Sequence to sequence learning with neural networks,” dans *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence et K. Q. Weinberger, édit. Curran Associates, Inc., 2014, p. 3104–3112. [En ligne]. Disponible : http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.pdf

[41] M.-T. Luong, H. Pham et C. D. Manning, “Effective approaches to attention-based neural machine translation,” *arXiv preprint arXiv :1508.04025*, 2015.

[42] D. Bahdanau, K. Cho et Y. Bengio, “Neural machine translation by jointly learning to align and translate,” *arXiv preprint arXiv :1409.0473*, 2014.

[43] K. Clark, U. Khandelwal, O. Levy et C. D. Manning, “What does BERT look at ? an analysis of BERT’s attention,” dans *Proceedings of the 2019 ACL Workshop BlackboxNLP : Analyzing and Interpreting Neural Networks for NLP*. Florence, Italy : Association for Computational Linguistics, août 2019, p. 276–286. [En ligne]. Disponible : https://www.aclweb.org/anthology/W19-4828

[44] K. He, X. Zhang, S. Ren et J. Sun, “Deep residual learning for image recognition,” dans *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, p. 770–778.

[45] J. L. Ba, J. R. Kiros et G. E. Hinton, “Layer normalization,” 2016.

[46] Y. Lu, Z. Li, D. He, Z. Sun, B. Dong, T. Qin, L. Wang et T.-Y. Liu, “Understanding and improving transformer from a multi-particle dynamic system point of view,” *arXiv preprint arXiv :1906.02762*, 2019.

[47] Z. Dai, Z. Yang, Y. Yang, J. G. Carbonell, Q. V. Le et R. Salakhutdinov, “Transformer-xl : Attentive language models beyond a fixed-length context,” *CoRR*, vol. abs/1901.02860, 2019. [En ligne]. Disponible : http://arxiv.org/abs/1901.02860

[48] S. F. Chen et J. Goodman, “An empirical study of smoothing techniques for language modeling,” *Computer Speech & Language*, vol. 13, n°. 4, p. 359–394, 1999.

[49] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee et L. Zettlemoyer, “Deep contextualized word representations,” *CoRR*, vol. abs/1802.05365, 2018. [En ligne]. Disponible : http://arxiv.org/abs/1802.05365

[50] ——, "Deep contextualized word representations," *arXiv preprint arXiv :1802.05365*, 2018.

[51] W. B. Dolan et C. Brockett, "Automatically constructing a corpus of sentential paraphrases," dans *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*, 2005.

[52] A. Williams, N. Nangia et S. R. Bowman, "A broad-coverage challenge corpus for sentence understanding through inference," *arXiv preprint arXiv :1704.05426*, 2017.

[53] E. F. Sang et F. De Meulder, "Introduction to the conll-2003 shared task : Language-independent named entity recognition," *arXiv preprint cs/0306050*, 2003.

[54] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov et Q. V. Le, "Xlnet : Generalized autoregressive pretraining for language understanding," dans *Advances in neural information processing systems*, 2019, p. 5753–5763.

[55] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer et V. Stoyanov, "Roberta : A robustly optimized bert pretraining approach," *arXiv preprint arXiv :1907.11692*, 2019.

[56] Y. Zhu, R. Kiros, R. Zemel, R. Salakhutdinov, R. Urtasun, A. Torralba et S. Fidler, "Aligning books and movies : Towards story-like visual explanations by watching movies and reading books," 2015.

[57] S. Nagel, "Cc-news," 2016.

[58] A. Gokaslan et V. Cohen, "Openwebtext corpus," 2019.

[59] T. H. Trinh et Q. V. Le, "A simple method for commonsense reasoning," *arXiv preprint arXiv :1806.02847*, 2018.

[60] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy et S. R. Bowman, "Glue : A multi-task benchmark and analysis platform for natural language understanding," *arXiv preprint arXiv :1804.07461*, 2018.

[61] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma et R. Soricut, "Albert : A lite bert for self-supervised learning of language representations," 2019.

[62] C. Lyu, Y. Lu, D. Ji et B. Chen, "Deep learning for textual entailment recognition," dans *2015 IEEE 27th International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE, 2015, p. 154–161.

[63] I. Dagan, O. Glickman et B. Magnini, "The pascal recognising textual entailment challenge," dans *Machine Learning Challenges Workshop*. Springer, 2005, p. 177–190.

[64] D. Z. Korman, E. Mack, J. Jett et A. H. Renear, "Defining textual entailment," *Journal of the Association for Information Science and Technology*, vol. 69, n°. 6, p. 763–772, 2018.

[65] Z. Zhang, X. Han, Z. Liu, X. Jiang, M. Sun et Q. Liu, "Ernie : Enhanced language representation with informative entities," *arXiv preprint arXiv :1905.07129*, 2019.

[66] T. Berners-Lee, J. Hendler et O. Lassila, "The semantic web," *Scientific american*, vol. 284, n°. 5, p. 34–43, 2001.

[67] F. Manola, E. Miller, B. McBride *et al.*, "Rdf primer," *W3C recommendation*, vol. 10, n°. 1-107, p. 6, 2004.

[68] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston et O. Yakhnenko, "Translating embeddings for modeling multi-relational data," dans *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani et K. Q. Weinberger, édit. Curran Associates, Inc., 2013, p. 2787–2795. [En ligne]. Disponible : http://papers.nips.cc/paper/5071-translating-embeddings-for-modeling-multi-relational-data.pdf

[69] T.-Y. Liu, "Learning to rank for information retrieval," *Foundations and Trends® in Information Retrieval*, vol. 3, n°. 3, p. 225–331, 2009. [En ligne]. Disponible : http://dx.doi.org/10.1561/1500000016

[70] R. Baeza-Yates, B. Ribeiro-Neto *et al.*, *Modern information retrieval*. ACM press New York, 1999, vol. 463.

[71] N. Fuhr, "Optimum polynomial retrieval functions based on the probability ranking principle," *ACM Transactions on Information Systems (TOIS)*, vol. 7, n°. 3, p. 183–204, 1989.

[72] A. Severyn et A. Moschitti, "Learning to rank short text pairs with convolutional deep neural networks," dans *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '15. New York, NY, USA : Association for Computing Machinery, 2015, p. 373–382. [En ligne]. Disponible : https://doi.org/10.1145/2766462.2767738

[73] R. Herbrich, T. Graepel et K. Obermayer, "Support vector learning for ordinal regression," 1999.

[74] T. Chen et C. Guestrin, "Xgboost : A scalable tree boosting system," dans *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. ACM, 2016, p. 785–794.

[75] W. Chen, T.-Y. Liu, Y. Lan, Z.-M. Ma et H. Li, “Ranking measures and loss functions in learning to rank,” dans *Advances in Neural Information Processing Systems*, 2009, p. 315–323.

[76] B. Mitra, N. Craswell *et al.*, “An introduction to neural information retrieval,” *Foundations and Trends® in Information Retrieval*, vol. 13, n°. 1, p. 1–126, 2018.

[77] Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai et H. Li, “Learning to rank : from pairwise approach to listwise approach,” dans *Proceedings of the 24th international conference on Machine learning*, 2007, p. 129–136.

[78] F. Cakir, K. He, X. Xia, B. Kulis et S. Sclaroff, “Deep metric learning to rank,” dans *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, p. 1861–1870.

[79] Z. Zhang, J. Yang et H. Zhao, “Retrospective reader for machine reading comprehension,” *arXiv preprint arXiv :2001.09694*, 2020.

[80] A. Trischler, T. Wang, X. Yuan, J. Harris, A. Sordoni, P. Bachman et K. Suleman, “Newsqa : A machine comprehension dataset,” *arXiv preprint arXiv :1611.09830*, 2016.

[81] S. Tellex, B. Katz, J. Lin, A. Fernandes et G. Marton, “Quantitative evaluation of passage retrieval algorithms for question answering,” dans *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, 2003, p. 41–47.

[82] D. Chen, J. Bolton et C. D. Manning, “A thorough examination of the CNN/daily mail reading comprehension task,” dans *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1 : Long Papers)*. Berlin, Germany : Association for Computational Linguistics, août 2016, p. 2358–2367. [En ligne]. Disponible : https://www.aclweb.org/anthology/P16-1223

[83] K. M. Hermann, T. Kocisky, E. Grefenstette, L. Espeholt, W. Kay, M. Suleyman et P. Blunsom, “Teaching machines to read and comprehend,” dans *Advances in neural information processing systems*, 2015, p. 1693–1701.

[84] J. Lee, S. Yun, H. Kim, M. Ko et J. Kang, “Ranking paragraphs for improving answer recall in open-domain question answering,” *CoRR*, vol. abs/1810.00494, 2018. [En ligne]. Disponible : http://arxiv.org/abs/1810.00494

[85] P. Yang, H. Fang et J. Lin, “Anserini : Enabling the use of lucene for information retrieval research,” dans *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR ’17. New York, NY, USA : Association for Computing Machinery, 2017, p. 1253–1256. [En ligne]. Disponible : https://doi.org/10.1145/3077136.3080721

[86] ——, “Anserini : Reproducible ranking baselines using lucene,” *J. Data and Information Quality*, vol. 10, n°. 4, oct. 2018. [En ligne]. Disponible : https://doi.org/10.1145/3239571

[87] A. Białecki, R. Muir, G. Ingersoll et L. Imagination, “Apache lucene 4,” dans *SIGIR 2012 workshop on open source information retrieval*, 2012, p. 17.

[88] J. Lee, M. Seo, H. Hajishirzi et J. Kang, “Contextualized sparse representations for real-time open-domain question answering,” 2019.

[89] M. J. Seo, J. Lee, T. Kwiatkowski, A. P. Parikh, A. Farhadi et H. Hajishirzi, “Real-time open-domain question answering with dense-sparse phrase index,” *CoRR*, vol. abs/1906.05807, 2019. [En ligne]. Disponible : http://arxiv.org/abs/1906.05807

[90] C. Gormley et Z. Tong, *Elasticsearch : the definitive guide : a distributed real-time search and analytics engine.* " O’Reilly Media, Inc.", 2015.

[91] S. E. Robertson et K. S. Jones, “Relevance weighting of search terms,” *Journal of the American Society for Information Science*, vol. 27, n°. 3, p. 129–146, 1976. [En ligne]. Disponible : https://asistdl.onlinelibrary.wiley.com/doi/abs/10.1002/asi.4630270302

[92] C. Clark et M. Gardner, “Simple and effective multi-paragraph reading comprehension,” dans *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1 : Long Papers).* Melbourne, Australia : Association for Computational Linguistics, juill. 2018, p. 845–855. [En ligne]. Disponible : https://www.aclweb.org/anthology/P18-1078

[93] A. Appleby, “Murmurhash3, 2012,” *URL : https ://github. com/aappleby/smhasher/blob/master/src/MurmurHash3. cpp*, 2012.

[94] K. Weinberger, A. Dasgupta, J. Langford, A. Smola et J. Attenberg, “Feature hashing for large scale multitask learning,” dans *Proceedings of the 26th annual international conference on machine learning*, 2009, p. 1113–1120.

[95] C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard et D. McClosky, “The Stanford CoreNLP natural language processing toolkit,” dans *Association for Computational Linguistics (ACL) System Demonstrations*, 2014, p. 55–60. [En ligne]. Disponible : http://www.aclweb.org/anthology/P/P14/P14-5010

[96] K. Lee, S. Salant, T. Kwiatkowski, A. Parikh, D. Das et J. Berant, “Learning recurrent span representations for extractive question answering,” *arXiv preprint arXiv :1611.01436*, 2016.

[97] P. Baudiš et J. Šedivý, “Modeling of the question answering task in the yodaqa system,” dans *Experimental IR Meets Multilinguality, Multimodality, and Interaction*, J. Mothe, J. Savoy, J. Kamps, K. Pinel-Sauvagnat, G. Jones, E. San Juan, L. Capellato et N. Ferro, édit. Cham : Springer International Publishing, 2015, p. 222–228.

[98] J. Berant, A. Chou, R. Frostig et P. Liang, “Semantic parsing on freebase from question-answer pairs,” dans *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, EMNLP 2013, 18-21 October 2013, Grand Hyatt Seattle, Seattle, Washington, USA, A meeting of SIGDAT, a Special Interest Group of the ACL*. Seattle, WA, USA : ACL, 2013, p. 1533–1544. [En ligne]. Disponible : https://www.aclweb.org/anthology/D13-1160/

[99] A. H. Miller, A. Fisch, J. Dodge, A. Karimi, A. Bordes et J. Weston, “Key-value memory networks for directly reading documents,” *CoRR*, vol. abs/1606.03126, 2016. [En ligne]. Disponible : http://arxiv.org/abs/1606.03126

[100] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz et J. Brew, “Huggingface’s transformers : State-of-the-art natural language processing,” *ArXiv*, vol. abs/1910.03771, 2019.

[101] D. P. Kingma et J. Ba, “Adam : A method for stochastic optimization,” dans *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio et Y. LeCun, édit., 2015. [En ligne]. Disponible : http://arxiv.org/abs/1412.6980