HAL
open science

Connaissances a priori pour la Recherche d'Information textuelle basée sur l'apprentissage profond

Jibril Frej

To cite this version:

Jibril Frej. Connaissances a priori pour la Recherche d'Information textuelle basée sur l'apprentissage profond. Recherche d'information [cs.IR]. Université Grenoble Alpes [2020-..], 2021. Français. NNT : 2021GRALM002 . tel-03322649

HAL Id: tel-03322649

https://theses.hal.science/tel-03322649v1

Submitted on 19 Aug 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Université Grenoble Alpes

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ GRENOBLE ALPES

Spécialité : **Informatique**

Arrêté ministériel : 25 mai 2016

Présentée par

Jibril FREJ

Thèse dirigée par **Jean-Pierre CHEVALLET**, UGA
et codirigée par **Didier SCHWAB**, UGA

préparée au sein du **Laboratoire d'Informatique de Grenoble (LIG), équipes MRIM et GETALP**
dans **l'École Doctorale Mathématiques, Sciences et Technologies de l'Information, Informatique (ED MSTII)**

Incorporation de Connaissances *a priori* pour la Recherche d'Information Textuelle Neuronale

Thèse soutenue publiquement le **5 février 2021**,
devant le jury composé de :

Mme Catherine Berrut
Professeur, Université Grenoble Alpes, Président
M. Mohand BOUGHANEM
Professeur, Université Paul Sabatier, Rapporteur
M. Jean-Pierre CHEVALLET
Maître de conférence, Université Grenoble Alpes, Directeur de thèse
Mme Christine LARGERON
Professeur, Université Jean Monnet, Examinateur
M. Didier Schwab
Maître de conférence, Université Grenoble Alpes, Invité
Mme Laure SOULIER
Maître de conférence, Sorbonne Université, Examinateur
M. Pierre ZWEIGENBAUM
Directeur de recherche, CNRS, Rapporteur

Résumé

Ce travail de thèse se situe dans les domaines de la recherche d'information (RI) textuelle et de l'apprentissage profond utilisant des réseaux de neurones. Les travaux effectués dans ce travail de thèse sont motivés par le fait que l'utilisation de réseaux de neurones en RI textuelle s'est révélée efficace sous certaines conditions mais que leur utilisation présente néanmoins plusieurs limitations pouvant grandement restreindre leur application en pratique.

Dans ce travail de thèse, nous proposons d'étudier l'incorporation de connaissances *a priori* pour aborder 3 limitations de l'utilisation de réseaux de neurones pour la RI textuelle : **(1)** la nécessité de disposer de grandes quantités de données étiquetées ; **(2)** les représentations du texte sont basées uniquement sur des analyses statistiques ; **(3)** le manque d'efficience.

Nous nous sommes intéressés à trois types de connaissances *a priori* pour aborder les limitations mentionnées ci-dessus : **(1)** des connaissances issues d'une ressource semi-structurée : *Wikipédia* ; **(2)** des connaissances issues de ressources structurées sous forme de ressources sémantiques telles que des ontologies ou des thésaurus ; **(3)** des connaissances issues de texte non structurées.

Dans un premier temps, nous proposons *WIKIR* : un outil libre d'accès permettant de créer automatiquement des collections de RI depuis *Wikipédia*. Les réseaux de neurones entraînés sur les collections créées automatiquement ont besoin par la suite de moins de données étiquetées pour atteindre de bonnes performances. Dans un second temps, nous avons développé des réseaux de neurones pour la RI utilisant des ressources sémantiques. L'intégration de ressources sémantiques aux réseaux de neurones leur permet d'atteindre de meilleures performances pour la recherche d'information dans le domaine médical. Finalement, nous présentons des réseaux de neurones utilisant des connaissances issues de texte non structurées pour améliorer la performance et l'efficience des modèles de référence de RI n'utilisant pas d'apprentissage.

Abstract

This thesis work is in the fields of textual information retrieval (IR) and deep learning using neural networks. The motivation for this thesis work is that the use of neural networks in textual IR has proven to be efficient under certain conditions but that their use still presents several limitations that can greatly restrict their application in practice.

In this thesis work, we propose to study the incorporation of prior knowledge to address 3 limitations of the use of neural networks for textual IR : **(1)** the need to have large amounts of labeled data, **(2)** a representation of the text-based only on statistical analysis, **(3)** the lack of efficiency.

We focused on three types of prior knowledge to address the limitations mentioned above : **(1)** knowledge from a semi-structured resource : *Wikipedia* ; **(2)** knowledge from structured resources in the form of semantic resources such as ontologies or thesauri ; **(3)** knowledge from unstructured text.

At first, we propose *WIKIR* : an open-access toolkit to automatically build IR collections from *Wikipedia*. The neural networks trained on the collections created automatically need less labeled data afterward to achieve good performance. Secondly, we developed neural networks for IR that use semantic resources. The integration of semantic resources into neural networks allows them to achieve better performance for information retrieval in the medical field. Finally, we present neural networks that use knowledge from unstructured text to improve the performance and efficiency of non-learning baseline IR models.

Table des matières

Résumé **iii**

Abstract **v**

1 Introduction **3**
1.1 Contexte général 3
1.2 Recherche d'information textuelle 4
1.3 Apprentissage profond pour la Recherche d'information textuelle . . 5
1.4 Motivations 5
1.5 Solutions proposées 6
1.6 Plan du manuscrit 8

I État de l'art **11**

2 Recherche d'Information **13**
2.1 Introduction 13
2.2 Système de RI textuelle 14
2.2.1 Pré-traitements linguistiques 14
2.2.2 Index inversé 16
2.2.3 Modèles basés sur la correspondance exacte 17
2.2.4 Évaluation 22
2.3 Disparité des termes 24
2.3.1 Expansion de requêtes 25
2.3.2 Ressources sémantiques 25
2.3.3 Apprentissage profond 26
2.4 Conclusion 27

3 Réseaux de neurones pour le TALN **29**
3.1 Introduction . . . 29
3.1.1 Perceptron Multicouche . . . 29
3.1.2 Apprentissage . . . 31
3.1.3 Différentiabilité . . . 32
3.2 Représentations continues de graphe relationnel . . . 33
3.2.1 Introduction . . . 33
3.2.2 Entraînement . . . 34
3.3 Représentations continues du texte . . . 35
3.3.1 Word2vec . . . 36
3.3.2 FastText . . . 39
3.4 LSTM pour le TALN . . . 40
3.4.1 Réseaux de neurones récurrents pour le TALN . . . 40
3.4.2 LSTM pour le TALN . . . 41
3.5 Couches de convolutions pour le TALN . . . 43
3.5.1 Couches de convolutions 1D . . . 43
3.5.2 Couches de convolutions 2D pour le TALN . . . 46
3.5.3 Fonction d'activation et couches de pooling . . . 47
3.6 Transformers . . . 48
3.6.1 Vue d'ensemble d'un encodeur *Transformers* . . . 48
3.6.2 Auto-attention . . . 48
3.6.3 Auto-attention multi-têtes . . . 51
3.6.4 Encodeur Transformers . . . 52
3.7 BERT . . . 54
3.8 Conclusion . . . 57

4 Recherche d'information neuronale supervisée **59**
4.1 Introduction . . . 59
4.2 Architectures symétrique et asymétrique . . . 60
4.2.1 Architecture symétrique . . . 60
4.2.2 Architecture asymétrique . . . 60
4.3 Représentation et interaction . . . 61
4.3.1 Architecture centrée sur la représentation . . . 61
4.3.2 Architecture centrée sur l'interaction . . . 61
4.4 Apprentissage . . . 62
4.4.1 Fonction objectif *pointwise* . . . 62
4.4.2 Fonction objectif *pairwise* . . . 63
4.4.3 Fonction objectif *listwise* . . . 65
4.5 Modèles . . . 66

4.5.1 ARC-I 66
4.5.2 ARC-II 66
4.5.3 MatchPyramid 68
4.5.4 DRMM 69
4.5.5 DUET 71
4.5.6 KNRM 72
4.5.7 CKNRM 74
4.6 BERT et modèles *Transformers* pré-entraînés pour la RI 77
4.6.1 Gérer la longueur des documents 77
4.6.2 Collection d'affinage 77
4.7 Limites des réseaux de neurones pour la RI 78
4.7.1 Quantité de données étiquetées 78
4.7.2 Ressources sémantiques 78
4.7.3 Incompatibilité avec l'index inversé 78
4.8 Conclusion 79

II Contributions 81

5 WikIR 83
5.1 Introduction 83
5.2 Un cadre général pour la création automatique de collection de RI . 85
5.2.1 Propriétés 85
5.2.2 Construction du jeu de données 86
5.2.3 Le cas de *Wikipédia* 87
5.3 Description de la boîte à outils WIKIR 88
5.3.1 *WIKIR* pour créer des collections de RI 89
5.3.2 WIKIR pour BM25 92
5.3.3 Reclassement neuronal avec *WIKIR* 92
5.3.4 Pré-entraînement des modèles NLTR avec *WIKIR* 93
5.4 Protocole expérimental 93
5.4.1 Collections de Recherche d'Information 93
5.4.2 Modèles de référence basés sur la correspondance exacte . . 95
5.4.3 Modèles NLTR 96
5.4.4 Évaluation 96
5.5 Résultats 96
5.5.1 Modèles NLTR sur les collections *Wikipédia* 96
5.5.2 Réseaux pré-entraînés sur TREC 100
5.6 Conclusions 101

6 Réseaux de Neurones Sémantiques pour la Recherche d'Information 105
6.1 Introduction . . . 105
6.2 Description des modèles . . . 106
6.2.1 Entrée des modèles . . . 106
6.2.2 Encodeurs de séquences . . . 107
6.2.3 Modèles basés sur la représentation . . . 107
6.2.4 Modèles basés sur l'interaction . . . 108
6.3 Protocole expérimental . . . 111
6.3.1 Collection NFcorpus . . . 113
6.3.2 Ressource sémantique dans le domaine médical : UMLS . . . 113
6.3.3 Ressource sémantique dans le domaine Général . . . 116
6.3.4 Vecteurs de mots . . . 116
6.3.5 Vecteurs de concepts . . . 117
6.3.6 Implémentation . . . 118
6.4 Résultats . . . 120
6.4.1 Modèles NLTR sémantiques dans le domaine médical . . . 120
6.4.2 Modèles NLTR sémantiques dans le domaine général . . . 123
6.5 Conclusions . . . 129

7 Réseaux de Neurones Efficients pour la Recherche d'Information 131
7.1 Introduction . . . 131
7.2 Apprentissage de valeur de discrimination . . . 133
7.2.1 RI traditionnelle différentiable . . . 133
7.2.2 Réseau de neurones peu profond pour l'apprentissage des TDV . . . 136
7.2.3 Apprendre les TDV avec de la RI différentiable . . . 136
7.3 Protocole expérimental . . . 139
7.3.1 Vecteurs de mots . . . 139
7.3.2 Nombre de paramètres . . . 139
7.3.3 Implémentation . . . 140
7.4 Résultats . . . 140
7.4.1 Performances des modèles NLTR efficients . . . 140
7.4.2 Accélération de la recherche . . . 142
7.4.3 Réduction de l'empreinte mémoire . . . 145
7.5 Conclusion . . . 146

8 Conclusion générale 149

A Publications **153**
A.1 Conférences internationales avec évaluation par les pairs 153
A.2 Conférences nationales avec évaluation par les pairs 153
A.3 Autres 154

B Définitions **155**

C Résultats supplémentaires **157**

D Bibliographie **169**

Table des figures

1.1 Représentation globale du fonctionnement d'un système de RI et de son interaction avec l'utilisateur. 4

2.1 Représentation simplifiée du fonctionnement d'un système de RI textuelle. Dans le domaine de la recherche d'information, le mot "indexation" désigne le procédé de construction de l'index inversé mais également l'ensemble des pré-traitements linguistiques appliqués aux documents de la collection. 14

2.2 Représentation simplifiée d'un index inversé associée à une collection de 3 documents. 16

2.3 Courbe de l'évolution du score BM25 en fonction du tf pour différentes valeurs de k_1 avec $b = 0$ et à idf constant 19

3.1 Architecture d'un MLP de $\mathbb{R}^2$ dans $\mathbb{R}$ à deux couches cachées de tailles $h_1 = 3$ et $h_2 = 4$. 31

3.2 Illustration des vecteurs TransE. 34

3.3 Illustration de la phase d'apprentissage du modèle *skip-gram* dans le cas d'un voisinage de taille 2. Afin de simplifier l'illustration, le détail du calcul du Softmax n'a pas été précisé. 38

3.4 Architecture d'un LSTM. Figure issue de https://colah.github.io/posts/2015-08-Understanding-LSTMs/ 43

3.5 Architecture simplifiée d'une couche de convolution 1D appliquée à une séquence de vecteur de mots. $X_{[1:3,:]}$ dénote les 3 premières lignes de la matrice X et $\odot$ dénote le produit élément par élément (ou produit de Hadamard) 45

3.6 Architecture simplifiée d'une couche de convolution 2D composée de k filtres de taille 3×3 appliquée à une Matrice. 46

3.7 Illustration de l'application d'une couche de max-pooling 1D de taille 2 et d'une couche de max-pooling 2D de taille 2. 47

3.8 Illustration du calcul de la matrice d'auto-attention. Pour plus de clarté, le calcul des matrices Q et K ainsi que la fonction softmax et le facteur d'échelle n'ont pas été représentés. Les valeurs représentés ont été calculés à l'aide du code disponible à cette adresse : https://nlp.seas.harvard.edu/2018/04/03/attention.html 50
3.9 Illustration des 4 matrices d'auto-attention produites par un encodeur Transformers à 4 têtes. Les valeurs représentés ont été calculés à l'aide du code disponible à cette adresse : https://nlp.seas.harvard.edu/2018/04/03/attention.html 52
3.10 *Heatmap* représentant des vecteurs de position de dimension 128 sur une séquence de longueur 100. 53
3.11 Architecture d'un encodeur *Transformers* ayant une séquence de mots en entrée 55
3.12 Architecture du modèle BERT 56

4.1 Architecture du modèle ARC-I. 67
4.2 Architecture du modèle ARC-II. 68
4.3 Architecture du modèle MatchPyramid. 69
4.4 Architecture du modèle DRMM. 70
4.5 Architecture du modèle DUET. 72
4.6 Architecture du modèle KNRM. 73
4.7 Architecture du modèle CKNRM. 75

5.1 Description du processus de construction d'une collection de RI par *WIKIR* en utilisant seulement deux articles. Dans cet exemple, les requêtes sont construites à partir du titre des articles. Les documents sont construits en utilisant le texte des articles sans le titre et sans la première phrase. 87
5.2 *json* file extracted from English *Wikipédia* dump using *WikiExtractor* 90

6.1 Architecture des modèles NLTR sémantiques basés sur la représentation 108
6.2 Histogrammes des similarités : produit scalaire, cosinus et noyau Gaussien entre un mot fixé et l'ensemble des mots du vocabulaire de la collection TREC *Robust04*. La flèche indique le score de similarité entre deux mots identiques. Figure traduite depuis l'article de Pang et al. (2016a) 109
6.3 Architecture des modèles NLTR sémantiques basés sur les interactions textuelle et conceptuelle 110

6.4 Architecture des modèles NLTR sémantiques basés sur une interaction globale 112
6.5 Illustration du concept C0004238 tiré du méta-thésaurus UMLS ainsi que des *terms*, *strings* et *atoms* qui lui sont associés. 114
6.6 Illustration d'un sous ensemble du graphe de relations entre concepts UMLS. 115
6.7 Illustration d'un sous ensemble du graphe relationnel DBpedia . . . 117

7.1 Architecture du modèle TDV-TF-IDF. Toutes les opérations sont différentiables et les gradients peuvent être rétro-propagés du score finale jusqu'à w et b. 138

Liste des tableaux

3.1 Comparaison entre les vecteurs de mots *skip-gram* et *fast*Text sur le jeux de données de similarité entre mots rares : *Rare Word Dataset* (Luong et al., 2013). La corrélation de Spearman (Spearman, 1904) est effectuée entre les jugements humain et la similarité cosinus entre vecteurs de mots. 40

4.1 Tableau récapitulatif des modèles NLTR de référence. Les modèles sont classés par ordre croissant de leur nombre de paramètres. . . . 76

5.1 Statistiques de plusieurs collections de RI *ad-hoc* en accès libre. Moy $\#d^+/r$ indique le nombre moyen de documents pertinents par requête. 84

5.2 Statistiques des collections wikIR78k et wikIRS78k. 94

5.3 Exemples de requêtes et d'un document pertinent associé provenant du NFCorpus. 95

5.4 Comparaison des performances des différents modèles sur wikIR78k. Les amélioration/dégradations statistiquement significatives par rapport à BM25-RM3 sont notées par (+/-) avec une p-value < 0.01. . . 98

5.5 Comparaison des performances des différents modèles sur wikIRS78k. Les amélioration/dégradations statistiquement significatives par rapport à BM25-RM3 sont notées par (+/-) avec une p-value < 0.01. . . 99

5.6 Performances sur les collections TREC des modèles de références, des réseaux de neurone, des réseaux de neurone pré-entraînés sur wikIR78k et des réseaux de neurone pré-entraînés sur wikIRS78k. Les amélioration/dégradations statistiquement significatives par rapport à BM25-RM3 sont notées par (+/-) avec une p-value < 0.01. . . 103

5.7 Gains de performance des modèles neuronaux pré-entraînés comparés aux modèles sans pré-entraînement 104

6.1 Exemples de requêtes et d'un document pertinent associé provenant du NFCorpus. Le corpus étant déjà pré-traité, les exemples ci-dessus ne contiennent ni majuscules, ni ponctuation. 113
6.2 Nombre de paramètres des modèles NLTR sémantiques. Les nombres de paramètres reportés dans ce tableaux correspondent aux valeurs des hyperparamètres donnant les meilleurs performances sur la collection NFCorpus. 118
6.3 Comparaison des performances de BM25-RM3, CKNRM et des modèles NLTR sémantiques avec les vecteurs de mots *word2vec* sur le NFCorpus. Les amélioration/dégradations statistiquement significatives par rapport à BM25-RM3 sont notées par (+/-) avec une p-value < 0.01. 120
6.4 Gains de performance obtenus avec l'incorporation de concepts sur la collection NFCorpus. 122
6.5 Comparaison de l'utilisation des vecteurs *bio-word2vec* (bw) avec les vecteurs BioBERT (BB) sur le NFCorpus. 123
6.6 Comparaison des performances des modèles NLTR sémantiques sur wikIR78k et wikIRS78k. Les amélioration/dégradations statistiquement significatives par rapport à BM25-RM3 sont notées par (+/-) avec une p-value < 0.01. 124
6.7 Gains de performance obtenus avec l'incorporation de concepts sur les WikIR78k et WikIRS78k. 125
6.8 Comparaison de l'utilisation des vecteurs *fast*Text (fT) avec les vecteurs BERT sur WikIR78k et WikIRS78k. 126
6.9 Comparaisons des performances sur les collections TREC de nos modèles pré-entraînés sur wikIR78k et wikIRS78k avec des modèles de référence. À l'exception du modèle ID-Rep$_{\text{BERT}_{\text{WikIRS}}}$ qui utilise les vecteurs de mots BERT, les résultats reportés sont ceux des modèles utilisant les vecteurs *fast*Text. Les amélioration/dégradations statistiquement significatives par rapport à BM25-RM3 sont notées par (+/-) avec une p-value < 0.01. 128

7.1 Nombre de paramètres des modèles NLTR efficients. 139
7.2 ndcg@5 des modèles NLTR efficients sur les collections TREC et *Wikipédia*. Les amélioration/dégradations statistiquement significatives par rapport au modèle basé sur la correspondance exacte associé sont notées par (+/-) avec une p-value < 0.01. 141
7.3 ndcg@5 des modèles NLTR efficients pré-entraînés sur les collections *Wikipédia* et affinés sur les collections TREC. 143

7.4 Comparaison du temps moyen (en millisecondes) de recherche par requête des modèles NLTR efficients sur les collections TREC. . . . 144
7.5 Comparaison du temps moyen (en millisecondes) de recherche par requête des modèles NLTR efficients sur les collections *Wikipédia*. . 144
7.6 Comparaison du temps moyen (en millisecondes) de recherche par requête des modèles NLTR efficients pré-entraînés sur les collections *Wikipédia* et affinés sur les collections TREC. 145
7.7 Réduction de l'empreinte mémoire de l'index inversé. 146
7.8 Réduction de l'empreinte mémoire de l'index inversé. 146

C.1 Comparaison des performances des modèles de référence et de nos contributions sur le NFCorpus. Les amélioration/dégradations statistiquement significatives par rapport à BM25-RM3 sont notées par (+/-) avec une p-value < 0.01. 158
C.2 Comparaison de l'utilisation des vecteurs *bio-word2vec* (bw) avec les vecteurs BioBERT (BB) sur le NFCorpus. 159
C.3 Comparaison des performances des différents modèles sur wikIR78k. Les amélioration/dégradations statistiquement significatives par rapport à BM25-RM3 sont notées par (+/-) avec une p-value < 0.01. . . 160
C.4 Comparaison des performances des différents modèles sur wikIRS78k. Les amélioration/dégradations statistiquement significatives par rapport à BM25-RM3 sont notées par (+/-) avec une p-value < 0.01. . . 161
C.5 Gains de performance obtenus avec l'incorporation de concepts sur la collection WikIR78k. 162
C.6 Gains de performance obtenus avec l'incorporation de concepts sur la collection WikIRS78k. 163
C.7 Comparaison de l'utilisation des vecteurs *fastText* (fT) avec les vecteurs BERT sur WikIR78k. 164
C.8 Comparaison de l'utilisation des vecteurs *fastText* (fT) avec les vecteurs BERT sur WikIRS78k. 165
C.9 Performances sur les collections TREC de nos modèles. Les amélioration/dégradations statistiquement significatives par rapport à BM25-RM3 sont notées par (+/-) avec une p-value < 0.01. 166
C.10 Performances sur les collections TREC de nos modèles pré-entraînés sur wikIR78k. Les amélioration/dégradations statistiquement significatives par rapport à BM25-RM3 sont notées par (+/-) avec une p-value < 0.01. 167

C.11 Performances sur les collections TREC de nos modèles pré-entraînés sur wikIRS78k. Les amélioration/dégradations statistiquement significatives par rapport à BM25-RM3 sont notées par (+/-) avec une p-value < 0.01. 168

Chapitre 1

Introduction

1.1 Contexte général

La recherche d'information (RI), qui étudie l'accès de documents répondant à un besoin d'information au sein d'une collection, est au cœur du fonctionnement de www.google.com, le site Web le plus populaire au monde [1]. Les utilisateurs d'un système de RI ont des besoins d'informations, exprimés sous forme de requête, qui doivent être satisfaits et effectuent donc des recherches dans des collections de documents non structurés afin de trouver les documents qui répondent à leurs besoins d'information.

Le fonctionnement global d'un système de RI ainsi que son interaction avec l'utilisateur sont illustrés sur la Figure 1.1. Étant donné un utilisateur avec un besoin d'information qu'il formule à l'aide d'une requête, le rôle d'un système d'information est de présenter à l'utilisateur des documents pertinents par rapport au besoin de l'utilisateur. Pour ce faire, le système de RI indexe les documents en une structure adaptée à la recherche et utilise un modèle de RI calculant la pertinence système des documents par rapport à la requête de l'utilisateur en affectant un score dit de pertinence aux documents et en classant ces derniers en utilisant ce score de pertinence. Il existe deux notions de pertinence en RI : la pertinence système que nous venons de mentionner qui consiste à comparer la requête aux documents à l'aide d'un modèle de RI et la pertinence utilisateur consistant à comparer directement le besoin d'information de l'utilisateur aux documents. Dans ce travail de thèse nous nous sommes principalement intéressés à la pertinence système. Par conséquent, dans la suite de ce manuscrit, l'utilisation du terme pertinence est à comprendre au sens de pertinence système. Dans ce travail de thèse, nous nous sommes concen-

1. Selon le classement Alexa https://www.alexa.com/topsites

trés sur les systèmes de recherche d'information textuels dans lesquels les requêtes sont exprimées sous forme de texte (phrase, mots-clés, question) et seul le contenu textuel des documents est pris en compte.

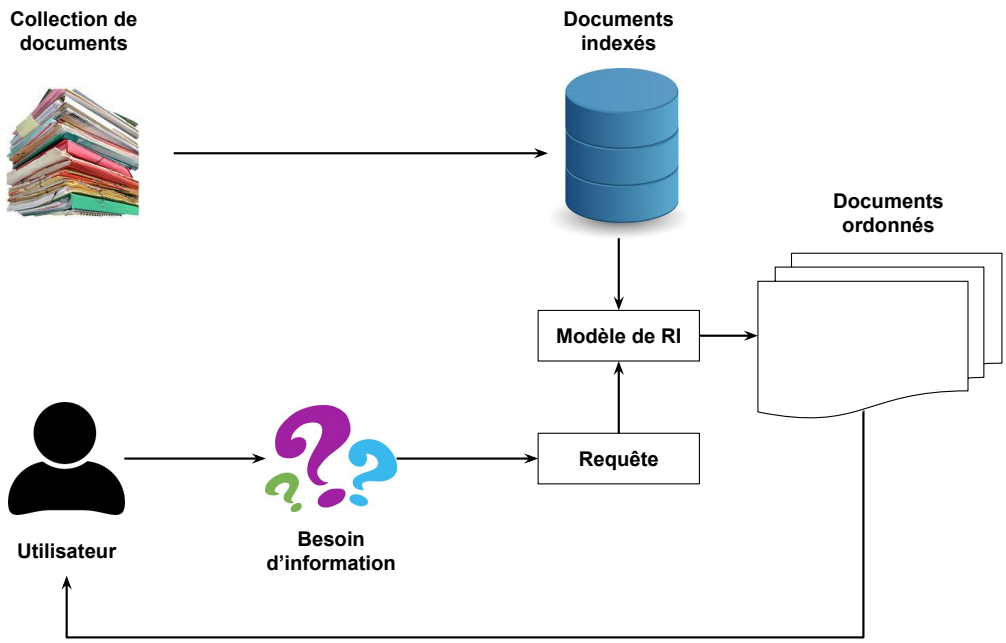

FIGURE 1.1 – Représentation globale du fonctionnement d'un système de RI et de son interaction avec l'utilisateur.

1.2 Recherche d'information textuelle

Les systèmes de recherche d'information textuels traditionnels s'appuient principalement sur un pré-traitement du texte (racinisation, suppression des mots vides, ...), suivi d'une correspondance exacte entre termes pour classer des documents en fonction de leur pertinence vis-à-vis d'une requête. Les modèles basés sur la correspondance exacte entre termes sont directement compatibles avec des structures de données (index inversé, arbre de recherche) permettant une recherche efficiente. En plus de leur rapidité, ces systèmes, couplés avec des méthodes d'expansion de requêtes, se sont révélés particulièrement performants et, jusqu'à récemment, difficiles à surpasser même en utilisant des méthodes modernes d'apprentissage profond (Yang et al., 2019a).

1.3 Apprentissage profond pour la Recherche d'information textuelle

Les méthodes d'apprentissage automatique, qui consistent en l'utilisation d'approches statistiques pour donner aux ordinateurs la capacité d'apprendre à résoudre des problèmes à partir de données, et plus particulièrement d'apprentissage profond (branche de l'apprentissage automatique utilisant des réseaux de neurones profonds) ont récemment largement surpassé les performances des modèles de RI traditionnels basés sur la correspondance exacte entre termes (Mitra et al., 2017; Pang et al., 2017; Dai et al., 2018; Fan et al., 2018; Guo et al., 2019b; Yang et al., 2019a; Han et al., 2020).

Les modèles d'apprentissage profonds pour la RI utilisent des requêtes dites résolues dont les documents pertinents et non-pertinents [1] sont connus afin d'apprendre à classer correctement les documents par rapport à une requête. Dans le cadre plus particulier de la RI textuelle, les modèles d'apprentissage profonds utilisent également des connaissances *a priori* sur le texte sous forme de vecteurs de mots dont les coefficients sont appris à l'aide d'une analyse statistique de grandes quantités de texte.

Les modèles d'apprentissage profonds présentent d'autres avantages que leurs performances. Ils permettent par exemple de considérablement réduire les procédés de pré-traitement du texte et les modèles les plus récent sont adaptables à de nombreuses tâches (Devlin et al., 2019).

1.4 Motivations

Malgré leur efficacité, les méthodes d'apprentissage profond présentent plusieurs limites. Dans ce travail de thèse, nous proposons d'aborder 3 limitations des méthodes d'apprentissage profond pour la RI :

1. la nécessité d'avoir de grandes quantités de données étiquetées.
2. une représentation du texte basée uniquement sur des analyses statistiques.
3. le manque d'efficience.

Quantité de données. Les méthodes d'apprentissage profond nécessitent de grandes quantités de données étiquetées Goodfellow et al. (2016) afin d'obtenir des gains

1. Ici, la pertinence peut être directement celle de l'utilisateur ou bien une estimation de la pertinence utilisateur, calculée en utilisant par exemple les clics sur un moteur de recherche (Zheng et al., 2018).

significatifs de performances par rapport aux modèles de référence basés sur la correspondance exacte ne nécessitant pas (ou très peu [1]) de données étiquetées. Cette contrainte est particulièrement problématique en RI textuelle, domaine pour lequel jusqu'à très récemment, il n'existait aucun jeu de données accessibles librement et ayant de grandes quantités de requêtes résolues. Les jeux de données accessibles librement comportaient seulement quelques centaines de requêtes étiquetées tandis que les jeux de données privés utilisées pour entraîner des modèles d'apprentissage profonds pour la RI comportent des centaines de milliers de requêtes résolues (Mitra et al., 2017).

Représentation du texte. Les méthodes d'apprentissage profond modélisent le texte à l'aide de vecteurs de mots dont le calcul prend en compte le contexte d'utilisation des mots sur de grandes quantités de texte. Ces méthodes permettent de modéliser des relations complexes entre les mots et donc de prendre en compte d'autres signaux que la correspondance exacte dans le calcul du score de pertinence. Les vecteurs de mots souffrent néanmoins du fait que les termes rares ne sont pas bien modélisés et que certaines relations entre termes sont difficilement voire impossible à extrapoler à l'aide d'analyse statistique du contexte d'utilisation des termes et ce même sur de grandes quantités de texte.

Efficience. Une des caractéristiques cruciales qu'un système de recherche d'information doit avoir est d'être capable de répondre à une requête de façon efficiente. Par exemple, le comportement d'un utilisateur de moteur de recherche est impacté si le recherche dure plus d'une seconde (Arapakis et al., 2014) et de façon plus générale, le temps d'attente tolérable pour un utilisateur d'un site Web est en moyenne de 2 secondes (Nah, 2003). Or, en plus de nécessiter des milliards d'opérations afin de prendre une unique décision [2], les méthodes d'apprentissage profond produisent des représentations de requêtes et de documents qui ne sont pas compatibles les structures de données permettant une recherche rapide. Ces méthodes ne sont donc pas efficientes.

1.5 Solutions proposées

Dans notre travail de doctorat, nous proposons d'utiliser différentes formes de connaissances *a priori* afin de palier aux 3 limitations présentées précédemment. Par connaissances *a priori*, nous désignons des connaissances sur la langue ou sur

1. Quelques données étiquetées peuvent être utiles pour choisir les valeurs de certains hyperparamètres.

2. Dans le cas de la RI, il s'agit de calculer un score de pertinence entre une requête et un document.

la RI textuelle ne provenant pas de l'ensemble des documents considérés par le système.

Texte structuré. Nous utilisons *Wikipédia* comme source de connaissances *a priori* sous forme de texte structuré afin de réduire la quantité de requêtes résolues nécessaire pour entraîner des modèles d'apprentissage profonds performants. Nous proposons *WIKIR* : un outil libre d'accès permettant de créer automatiquement depuis *Wikipédia* des collections de RI plusieurs dizaines de milliers de requêtes étiquetées. Nous utilisons ces collections de RI fabriquées automatiquement pour entraîner des modèles d'apprentissage profonds. Ces derniers sont ensuite affinés sur des collections standards de RI contenant quelques centaines de requêtes résolues. Nos expériences montrent que ces modèles neuronaux atteignent de meilleures performances que les modèles de référence malgré la faible quantité de requêtes résolues.

Ressources sémantiques. Nous supposons que la prise en compte de connaissances *a priori* sous forme de ressources sémantiques, qui proposent une formalisation explicite des significations des mots et de leurs relations éventuelles, consiste en une approche complémentaire aux vecteurs de mots basés sur une analyse purement statistique de la langue. Nous proposons des réseaux de neurones sémantiques utilisant des vecteurs de mots et des vecteurs de concepts entraînés sur des ressources sémantiques. Nos expériences montrent que les modèles neuronaux sémantiques atteignent de meilleures performances que les modèles neuronaux de référence pour la RI dans le domaine médical.

Texte non structuré. Finalement, nous proposons de nouvelles architectures de réseaux de neurones qui, au lieu d'augmenter la qualité des résultats au détriment de l'efficience et nécessitant de grande quantités de requête résolues, améliore significativement l'efficience des modèles basés sur la correspondance exacte sans dégrader la qualité des résultats et ne nécessitant que peu de requêtes résolues. Nos modèles neuronaux efficients utilisent des connaissances *a priori* sous forme de vecteurs de mots appris sur de grandes quantités de texte non structuré afin d'associer aux termes une valeur de discrimination (TDV, de l'anglais *Term Discrimination Value*) dont le but est de refléter l'utilité d'un terme afin de discriminer les documents. Les termes n'étant pas utiles pour discriminer les documents pertinents des documents non-pertinents ne sont pas pris en compte par le système de RI. Nos expériences montrent que nos modèles permettent de diminuer significativement le temps de recherche des modèles basés sur la correspondance exacte, en améliorant la qualité des résultats et en ne nécessitant que quelques centaines de requêtes résolues pour l'apprentissage.

1.6 Plan du manuscrit

Cette thèse est organisée en deux parties principales : l'état de l'art (chapitres 2, 3 et 4) et nos contributions (chapitres 5, 6 et 7). Nous détaillons le contenu de chacun des chapitres ci-dessous :

Chapitre 2. Ce chapitre présente le fonctionnement global d'un système de RI textuel et des différents modèles de référence basés sur la correspondance exacte utilisés dans cette thèse. Nous présentons brièvement certaines de leurs limites ainsi que quelques solutions communément utilisées.

Chapitre 3. Ce chapitre décrit le fonctionnement global des réseaux de neurones pour le traitement automatique du langage naturel (TALN). Nous détaillons également les notions liées aux réseaux de neurones qui sont importantes pour la compréhension de ce travail de thèse ainsi que les architectures utilisées dans les modèles états de l'art et dans les modèles que nous proposons.

Chapitre 4. Ce chapitre présente une formulation unifiée des modèles neuronaux pour la RI textuelle, les types d'architectures communément utilisés, les différents types de fonctions objectifs ainsi que les modèles neuronaux de référence utilisés dans nos expériences.

Chapitre 5. Ce chapitre décrit notre première contribution : *WIKIR*, un *toolkit* en libre accès pour construire des collections de RI à grande échelle basés sur *Wikipédia*. Nous y développons un cadre de travail général permettant de construire automatiquement des collections de RI à partir de *Wikipédia* et plus généralement à partir de n'importe quelle ressource satisfaisant quelques propriétés. Nous détaillons les différentes fonctionnalités proposées par *WIKIR*, ainsi que les collections que nous avons construites. Dans nos expériences, nous entraînons des modèles neuronaux sur les collections issues de *WIKIR* et les affinons sur des collections standards de RI contenant quelques centaines de requêtes résolues. Ces travaux ont donné lieu à deux articles de conférence : Frej et al. (2020d) et Frej et al. (2020c).

Chapitre 6. Ce chapitre présente les réseaux de neurones sémantiques qui utilisent : **(1)** des vecteurs de mots associés à la requête et au document comme le ferait un réseaux de neurones pour la RI textuelle ; **(2)** des vecteurs de concepts/entités issue de la ressource sémantique qui sont présents dans la requête et le document. Nous proposons plusieurs types d'architectures de réseaux de neurones sémantiques

et expérimentons sur deux types de collections de RI : une collection dans le domaine médical avec des concepts médicaux et des collections dans le domaine général avec des concepts non-spécifiques à un domaine particulier. Ces travaux ont donné lieu à un article de conférence Frej et al. (2020b) et un article d'atelier FREJ et al. (2020).

Chapitre 7. Ce chapitre présente les réseaux de neurones efficients que nous avons développés et qui permettent d'apprendre la TDV des termes à l'aide de leurs vecteurs de mots et de ne plus considérer les termes ayant une TDV nulle lors du calcul du score de pertinence. Dans nos expériences, nous évaluons les modèles neuronaux efficients sur les collections issues de *WIKIR* et sur des collections standards de RI contenant quelques centaines de requêtes résolues, et ce, afin d'évaluer la capacité des modèles efficients d'accélérer la recherche sans nécessiter beaucoup de requête résolues. Ces travaux ont donné lieu à un article de conférence Frej et al. (2020a).

Chapitre 8. Ce chapitre comprend les conclusions générales et les principales perspectives de cette thèse.

Première partie

État de l’art

Chapitre 2

Recherche d'Information

2.1 Introduction

Bien que le terme de Recherche d'Information ("*retrieval of information*") fut proposé pour la première fois par Mooers en 1950 (Mooers, 1950), l'idée d'accéder automatiquement et rapidement à de grandes quantités de données stockées de façon mécanisée a été popularisée en 1945 par l'article *As We May Think* de Vannevar Bush dans lequel il propose un "memex" : "*A memex is a device in which an individual stores all his books, records, and communications, and which is mechanized so that it may be consulted with exceeding speed and flexibility. It is an enlarged intimate supplement to his memory.*" (Bush, 1945). La première mise en œuvre pratique du "memex" fut le *rapid selector* (Shaw, 1949) qui a été conçu pour une utilisation en bibliothèque. Le *rapid selector* a été rapidement abandonné à cause de nombreuses restrictions (Bagg et Stevens, 1961) qui le rendaient difficilement exploitable et peu utile en pratique.

Depuis le "memex", de nombreux modèles de RI textuelle ont été proposés et expérimentés tels que les modèles booléens (Taube et al., 1952), les modèles vectoriels (Switzer, 1964; Salton, 1968), les modèles de langue pour la RI (Ponte et Croft, 1998), les modèles probabilistes (Robertson et Walker, 1994), les modèles d'apprentissage automatique (Wu et al., 2010) ou les modèles d'apprentissage profond (Mitra et al., 2017). Dans la suite de cet état de l'art dédié aux systèmes de RI textuelle, nous présenterons les différents traitements appliqués aux documents, les modèles basés sur la correspondance exacte, les mesures d'évaluation communément utilisées en RI et les limitations de la correspondance exacte.

2.2 Système de RI textuelle

Dans cette section, nous présentons le fonctionnement d'un système de RI textuelle tel que celui illustré sur la Figure 2.1. L'objectif d'un système de RI est de classer des documents par rapport à leur pertinence vis-à-vis du besoin de l'utilisateur exprimé sous forme de requête.

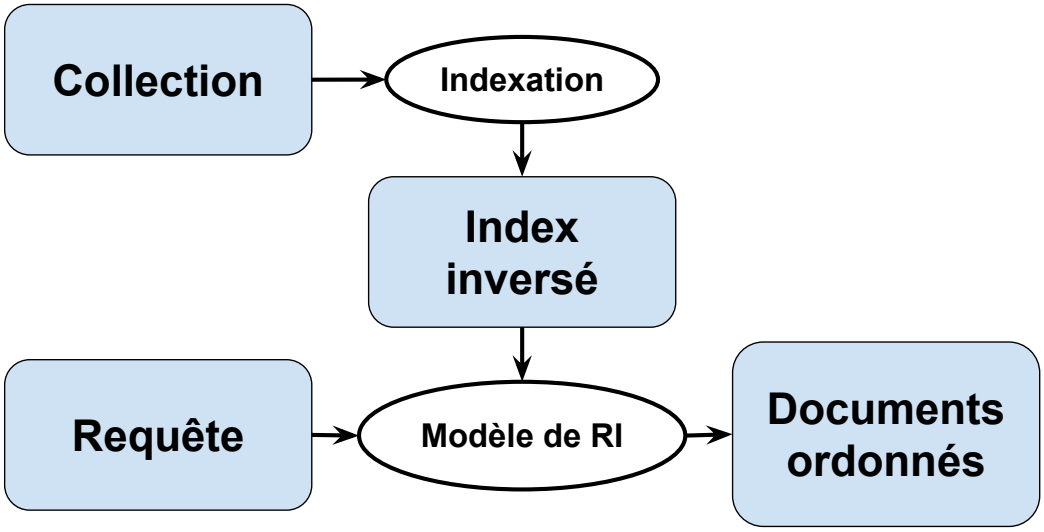

FIGURE 2.1 – Représentation simplifiée du fonctionnement d'un système de RI textuelle. Dans le domaine de la recherche d'information, le mot "indexation" désigne le procédé de construction de l'index inversé mais également l'ensemble des prétraitements linguistiques appliqués aux documents de la collection.

2.2.1 Pré-traitements linguistiques

Dans ce qui suit nous présentons les différents pré-traitements linguistiques communément appliqués aux documents : la segmentation, la normalisation textuelle, la racinisation et le filtrage par un antidictionnaire.

Segmentation

L'étape de segmentation (*tokenization*) consiste à séparer la séquence de caractères du document en termes d'indexation qui formeront le vocabulaire. Les deux

méthodes de segmentation les plus populaires pour les langues indo-européennes sont la segmentation par espaces et par n-gammes de caractères [1] et, plus récemment, des méthodes de segmentations s'adaptant à la langue et à la tâche considérée (Wang et al., 2020).

Normalisation textuelle

"*La normalisation textuelle rend les mots d'une même famille sous leur forme canonique en effectuant quelques transformations superficielles sur les séquences de caractères de ces mots*" (Amini et Gaussier, 2013). En pratique, la normalisation textuelle est un ensemble de règles spécifiques à chaque langue. Pour le français par exemple, ces règles sont habituellement : la suppression de la ponctuation, la mise en minuscule de tous les caractères, la suppression des accents et la suppression des caractères non-alphanumériques.

Racinisation

La racinisation (*stemming*) consiste à associer à un mot sa racine (*stem*). La racinisation peut entraîner des erreurs : par exemple si le mot *soleil* est réduit à la racine *sol*, alors le mot soleil et le mot sol seront associés au même terme et le système de RI pourrait renvoyer des documents parlant du sol à un utilisateur utilisant le mot soleil dans sa requête. Malgré ces potentielles erreurs, la racinisation est communément employée en RI puisqu'il a été empiriquement constaté qu'elle permet d'avoir de meilleurs classements (Walker et al., 1987), qu'elle réduit la taille du vocabulaire [2].

Filtrage par un antidictionnaire

Le filtrage par un antidictionnaire (*stop-words removal*) consiste en la suppression de mots vides qui sont extrêmement fréquents dans la langue considérée et qui n'apportent peu ou pas d'information sur le contenu des documents. En supprimant les mots peu informatifs, le filtrage par un antidictionnaire permet de réduire la taille en mémoire des documents indexés et d'accélérer la recherche tout en ayant un impact négligeable sur le classement. Voici quelques exemples de mots vides de la langue française tirés du *Natural Language Toolkit* (Bird, 2006) : "alors", "après", "car", "ce", "la", "laquelle", "le", "un", "une", "unes"...

1. Accompagnées d'autres règles spécifiques aux différentes langues.
2. Ce qui permet d'avoir un système plus rapide et utilisant moins de mémoire.

2.2.2 Index inversé

L'index inversé est une structure de données utilisée communément en RI textuelle pour permettre aux modèles d'accéder de façon efficiente aux documents. L'index inversé fait correspondre chaque terme du vocabulaire à l'ensemble des documents contenant ce terme et, si implémenté correctement, permet d'accéder rapidement à l'ensemble des documents contenant au moins un terme de la requête. Il est possible d'implémenter un index inversé de plusieurs manières. La Figure 2.2 présente un index inversé implémenté avec une table de hachage (*hash map*) dont les clés sont les termes du vocabulaire t et les valeurs sont des listes (*posting lists*) de paires $(d, \mathrm{tf}_{t,d})$.

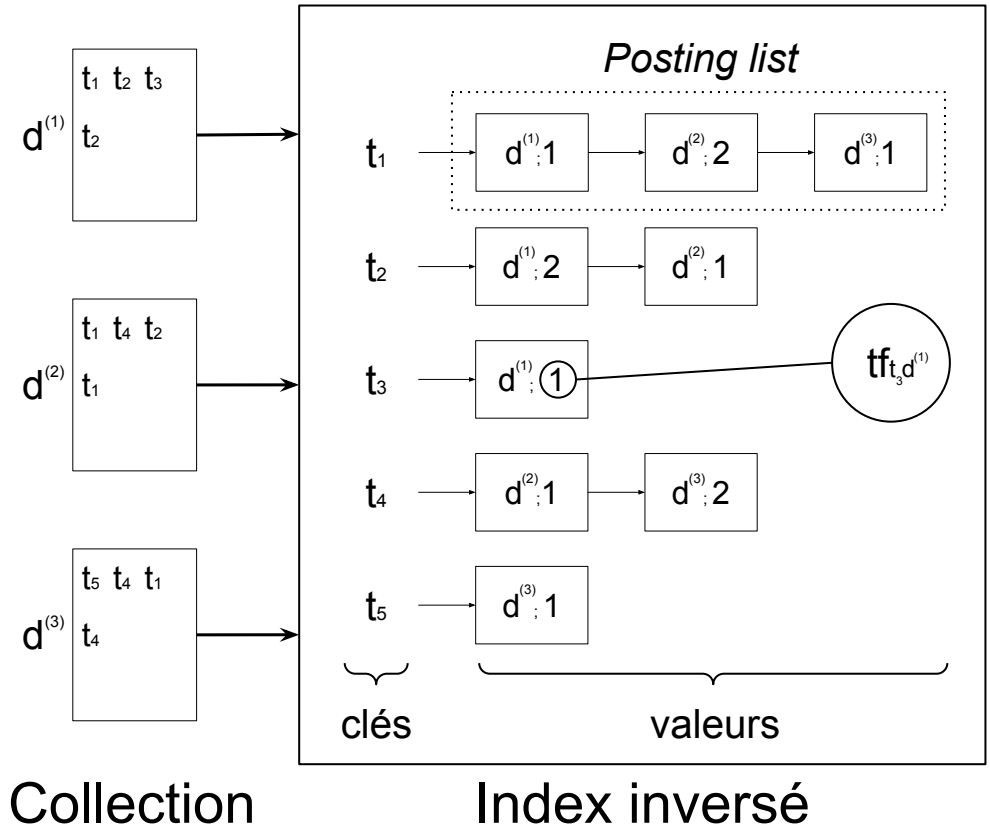

FIGURE 2.2 – Représentation simplifiée d'un index inversé associée à une collection de 3 documents.

L'utilisation d'un index inversé permet d'avoir un système de RI efficient à condition que le modèle de RI utilisé soit compatible avec l'index inversé. En effet,

puisqu'un index inversé permet d'avoir rapidement accès à l'ensemble des documents contenant au moins un terme de la requête, les modèles utilisés avec un index inversé supposent que seuls les documents contenant au moins un terme de la requête peuvent être pertinents. Par conséquent, seuls les documents contenant au moins un terme de la requête sont considérés et retournés par le système. Dans la suite de cette thèse, nous ferons référence à ce type de modèles comme les modèles basés sur la correspondance exacte (*exact matching*).

Finalement, notons que l'utilisation d'un index inversé ne permet une recherche efficiente qu'à la condition que les *posting lists* ne soient pas trop longues. En effet, si un terme du vocabulaire apparaît dans l'ensemble des documents, sa *posting list* sera de la taille de la collection et tous les documents seront pris en compte dans le classement et le système ne sera pas efficient. L'étape de filtrage par un antidictionnaire est donc essentielle afin d'éviter cette situation puisque les mots vides sont très fréquents et que leur suppression permet d'éviter d'avoir des *posting lists* trop longues. De plus, certains systèmes de RI proposent également de supprimer du vocabulaire les termes les plus fréquents de la collection afin d'accélérer la recherche.

2.2.3 Modèles basés sur la correspondance exacte

Dans ce qui suit nous présentons les modèles basés sur la correspondance exacte que nous avons utilisés dans le cadre de cette thèse.

TF-IDF

Le modèle TF-IDF à été popularisé par le modèle vectoriel (Salton et al., 1975) qui associe à la requête et au document un vecteur dans un espace de termes et calcule leur pertinence à l'aide d'une mesure de similarité vectorielle. TF-IDF fait appel à deux notions pour calculer le score de pertinence d'un document d par rapport à une requête r : le tf (term frequency) et l'idf (*inverted document frequency*) :

$$\text{TF-IDF}(r,d) = \sum_{t \in r} \text{tf}_{t,d}\ \text{idf}_t, \tag{2.1}$$

avec $\text{tf}_{t,d}$, le nombre d'occurrences du terme t dans le document d et idf_t la fréquence inverse de document du terme t calculée comme suit :

$$\text{idf}_t = \log \frac{|C| + 1}{\text{df}_t}, \tag{2.2}$$

avec $|C|$ le nombre total de documents dans la collection considérée et df_t le nombre de documents dans lesquels le terme t occurre. L'utilisation du tf permet de donner

plus d'importance aux documents contenant de nombreuses occurrences des termes de la requête (Maron et Kuhns, 1960). L'idf quant à elle reflète la spécificité d'un terme : plus l'idf d'un terme est élevée, moins ce terme occurre dans la collection et donc plus ce terme est spécifique. L'utilisation de l'idf permet donc de donner plus d'importances aux documents contenant des termes rares et donc spécifiques (Jones, 1972; Salton et Yang, 1973).

Okapi BM25

Une des limitation du modèle TF-IDF est qu'il ne prend pas en compte la longueur des documents lors du calcul de leur pertinence. Or, plus un document est long, plus il contient de termes et donc plus la probabilité qu'il contienne un terme de la requête sans être pertinent est élevé. Okapi BM25 (Robertson et Walker, 1994) est une variante du TF-IDF prenant en compte la longueur des documents et qui est encore utilisé de nos jours comme un modèle de référence (*baseline*) en RI textuelle. Okapi BM25 est un modèle dit probabiliste estimant $\mathbb{P}(pertinence|r,d)$: la probabilité de pertinence sachant la requête r et le document d et classant les documents en utilisant cette probabilité. Afin de faire le lien entre $\mathbb{P}(pertinence|r,d)$ et les formules présentées dans ce qui suit, le lecteur peut se référer à (Robertson et Zaragoza, 2009). La formule du calcul de pertinence de BM25 est la suivante :

$$\mathrm{BM25}(r,d) = \sum_{t \in r} \mathrm{idf}_t \frac{\mathrm{tf}_{t,d}(k_1+1)}{\mathrm{tf}_{t,d} + k_1\left(1 - b + b\frac{|d|}{avgdl}\right)}, \tag{2.3}$$

avec $k_1 \in \mathbb{R}^+$ et $b \in [0,1]$ des paramètres de BM25, $|d|$ la longueur du document d et $avgdl$ la longueur moyenne des documents. Plus un document est long, moins son score de pertinence selon BM25 sera élevé. Le paramètre b permet de donner plus ou moins d'importance à la longueur des documents. Dans le cas extrême où b est égal à 0, la longueur des documents ne sera pas prise en compte et plus la valeur de b est élevée, plus la longueur des documents aura une influence sur le score de pertinence. Pour comprendre le rôle du paramètre k_1, considérons le cas simplifié où l'on a une requête r à un terme t tel que $\mathrm{idf}_t = 1$ et où l'on pose $b = 0$, nous avons donc la formule de classement suivante :

$$\mathrm{BM25}(r,d) = \frac{\mathrm{tf}_{t,d}(k_1+1)}{\mathrm{tf}_{t,d} + k_1}. \tag{2.4}$$

Constatons que le score ci-dessus tend vers (k_1+1) lorsque $\mathrm{tf}_{t,d} \to +\infty$.

Le paramètre k_1 impose donc une borne supérieure au score de BM25 en fonction du tf d'un terme (voir Figure 2.3). Plus la valeur de k_1 est élevée, plus un terme

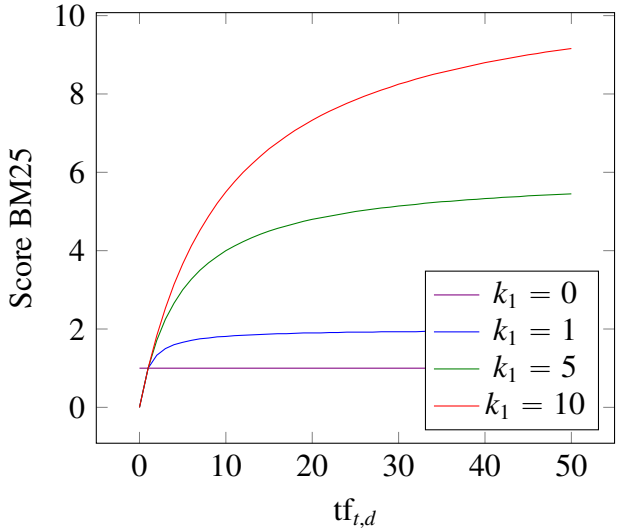

FIGURE 2.3 – Courbe de l'évolution du score BM25 en fonction du tf pour différentes valeurs de k_1 avec $b = 0$ et à idf constant

ayant une fréquence élevée contribuera de façon importante au score de BM25. k_1 a été introduit pour gérer les documents "saturés" par certains termes et ainsi éviter que les termes extrêmement présent dans certains documents aient un impact trop important dans le calcul du score de pertinence. Le rôle de k_1 est différent de celui de l'idf : l'idf permet de gérer les termes présents dans beaucoup de documents de la collection (et donc peu spécifiques) et k_1 permet de gérer les termes très fréquents au sein d'un document.

Modèles de langue pour la RI

Les modèles de langues pour la RI (Ponte et Croft, 1998) proposent quant à eux de calculer le score de pertinence en estimant $\mathbb{P}(r|\theta_d)$: la probabilité que la requête r soit générée par le modèle de langue associé au document d. Dans le cadre de cette thèse nous avons étudié deux modèles de langues communément utilisés en RI : le modèle de Jelinek-Mercer et le modèle de Dirichlet. Ces deux modèles sont dits uni-gramme : la probabilité de générer un terme de la requête est indépendante des termes précédents de la requête :

$$\mathbb{P}(r|\theta_d) = \prod_{t \in r} \mathbb{P}(t|\theta_d), \tag{2.5}$$

En pratique, pour des raisons liées à la précision des calculs effectués par les

machines et pour simplifier certaines expressions, le log de la probabilité est utilisé :

$$\log \mathbb{P}(r|\theta_d) = \sum_{t \in r} \log \mathbb{P}(t|\theta_d). \tag{2.6}$$

Afin de calculer la probabilité de générer le terme t sachant le modèle du document d : $\mathbb{P}(t|\theta_d)$, il est supposé que la distribution des termes du documents suit une loi multinomiale. Les paramètres de cette dernière sont estimés par maximum de vraisemblance, ce qui donne :

$$\mathbb{P}(t|\theta_d) = \frac{\mathrm{tf}_{t,d}}{|d|}. \tag{2.7}$$

Ce qui différencie les modèles de langue de Jelinek-Mercer et de Dirichlet est la méthode de lissage (*smoothing*) utilisée. Les méthodes de lissage permettent d'associer aux termes de la requête n'occurrant pas dans le document une probabilité non-nulle. En effet, au vu des équations 2.5 et 2.7, si un document d ne contient pas un terme t de la requête alors il aura un score de pertinence nulle : $\mathrm{tf}_{t,d} = 0 \Rightarrow \mathbb{P}(t|\theta_d) = 0 \Rightarrow \mathbb{P}(r|d) = 0$. En d'autre termes, sans lissage, seuls les documents contenant tous les termes de la requête sont considérés.

De façon plus concrète, le lissage consiste à utiliser $\mathbb{P}(t|\theta_C)$: la probabilité que le terme t soit généré par le modèle de langue associé à la collection C. De la même façon que pour $\mathbb{P}(t|\theta_D)$, et ce, pour les deux modèles de langue, $\mathbb{P}(t|\theta_C)$ est estimé par maximum de vraisemblance sur une loi multinomiale :

$$\mathbb{P}(t|\theta_C) = \frac{\mathrm{tf}_{t,C}}{\sum_{t' \in V} \mathrm{tf}_{t',C}}, \tag{2.8}$$

avec $\mathrm{tf}_{t,C}$ le nombre d'occurrences du terme t dans la collection C. Le modèle de langue de Jelinek-Mercer calcule le score de pertinence en faisant une combinaison linéaire entre $\mathbb{P}(t|\theta_d)$ et $\mathbb{P}(t|\theta_C)$:

$$\mathrm{JM}(r,d) = \sum_{t \in r} \log \left(\lambda \mathbb{P}(t|\theta_d) + (1-\lambda)\mathbb{P}(t|\theta_C) \right), \tag{2.9}$$

avec $\lambda \in [0,1]$ le paramètre permettant de donner plus ou moins d'importance au modèle de langue du document par rapport au modèle de langue de la collection.

Le modèle de langue de Dirichlet quant à lui pondère le modèle de langue de la collection par la longueur du document :

$$\mathrm{Dir}(r,d) = \sum_{t \in r} \log \left(\frac{\mathrm{tf}_{t,d} + \mu \mathbb{P}(t|\theta_C)}{|d| + \mu} \right), \tag{2.10}$$

avec $\mu \in \mathbb{R}^+$ le paramètre déterminant l'importance du modèle de langue de la collection dans le calcul du score de pertinence.

Il est possible de trouver dans la littérature scientifique d'autres formulations des modèles de langues que nous venons de présenter (Zhai et Lafferty, 2001). Ces formules produisent des classement de documents identiques que leurs homologues et sont plus proches des formules implémentées dans les systèmes de RI :

$$\mathrm{JM}(r,d) = \sum_{t \in r} \log \left(1 + \frac{\lambda \mathbb{P}(t|\theta_d)}{(1-\lambda)\mathbb{P}(t|\theta_C)} \right), \tag{2.11}$$

$$\mathrm{Dir}(r,d) = \sum_{t \in r} \log \left(1 + \frac{\mathrm{tf}_{t,d}}{\mu \mathbb{P}(t|\theta_C)} \right) + |r| \log \frac{\mu}{|d| + \mu}, \tag{2.12}$$

Boucle de rétropertinence en aveugle

La méthode de boucle de rétropertinence en aveugle (*pseudo relevance feedback*) est dérivée de la méthode de boucle de rétropertinence (*relevance feedback*) (Rocchio, 1971) qui consiste à modifier la requête en utilisant des termes de l'ensemble des documents jugés pertinents, noté $\mathcal{D}_p$ [1]. En pratique, la boucle de rétropertinence consiste à ajouter à la requête des termes pondérés issus de $\mathcal{D}_p$. La prise en compte des termes pondéré dans le calcul du score de pertinence se fait en multipliant le score des termes pondérés par leur poids associées. Par exemple le score du modèle de langue de Dirichlet avec une boucle de rétropertinence devient :

$$\mathrm{Dir}_{rf}(r,d) = \sum_{t \in r'} P(t) \log \left(\frac{\mathrm{tf}_{t,d} + \mu \mathbb{P}(t|\theta_C)}{|d| + \mu} \right), \tag{2.13}$$

avec $P(t)$ le poids associé au terme t et r' la requête r étendu avec des termes pondéré.

Dans le cadre de cette thèse, nous utilisons la méthode de pondération RM3 (Jaleel et al., 2004) (*relevance model 3*), originellement développée pour les modèles de langue mais également utilisée avec BM25. Étant donné une requête r et un ensemble de documents pertinents $\mathcal{D}_p$, la pondération RM3 d'un terme t dans une requête r est calculée comme suit :

1. Dans ses travaux originaux, Rocchio proposait d'utiliser également les documents non-pertinent pour modifier la requête mais cela est rarement fait en pratique (Amini et Gaussier, 2013)

$$
\begin{aligned}
\mathrm{RM3}_r(t) &= \alpha \mathbb{P}(t|\theta_q) + (1-\alpha)\mathbb{P}(t|\theta_{\mathcal{D}_p}) \\
&= \alpha \mathbb{P}(t|\theta_q) + (1-\alpha) \sum_{d \in \mathcal{D}_p} \left(\mathbb{P}(t|\theta_d) \prod_{t' \in r} \mathbb{P}(t'|\theta_d) \right),
\end{aligned} \tag{2.14}
$$

avec $\alpha \in [0, 1]$. RM3 pondère un terme t en fonction de 3 critères :

1. $\mathbb{P}(t|\theta_q)$: si t est déjà présent dans la requête r, il aura une pondération plus importante
2. $\sum_{d \in \mathcal{D}_p} \mathbb{P}(t|\theta_d)$: plus t occurre fréquemment dans des documents pertinents, plus sa pondération sera importante
3. $\prod_{t' \in r} \mathbb{P}(t'|\theta_d)$: si t occurre dans un document pertinent contenant des termes de la requête, sa pondération en sera augmentée

De la même façon que pour les modèles de langue présentés dans la section précédente, les probabilités sont estimées par maximum de vraisemblance en supposant que les distributions des termes suivent une loi multinomiale. En pratique, tous les termes ayant une pondération non-nulle ne sont pas ajoutés à la requête afin d'éviter que cette dernière ne soit trop longue : seuls les n termes ayant la pondération la plus élevée sont ajoutés.

Le procédé de boucle de rétropertinence, nécessitant une interaction avec l'utilisateur, a été automatisé en supposant que les k premiers documents retournés par un système sont pertinents. Cette méthode dite de boucle de rétropertinence en aveugle (*pseudo relevance feedback*) a été adaptée à tous les modèles présentés dans les sections précédentes et permet d'améliorer le classement des documents dans la majorité des cas, même s'il est possible que dans certains cas la requête générée automatiquement corresponde moins au besoin de l'utilisateur que la requête originale. Par exemple, si la requête parle de "mines de cuivre" et que les documents les plus pertinents selon le modèle parlent de "mines de cuivre au Chili", la requête pourrait être étendue avec le terme "Chili" qui n'a pas de lien avec le besoin de l'utilisateur (exemple tiré du livre de Manning et al. (2008)).

2.2.4 Évaluation

Afin d'évaluer un système de RI, il est nécessaire d'avoir :

1. Une collection de documents

2. Un ensemble [1] de requêtes
3. Un ensemble de jugements de pertinence indiquant la pertinence de chaque paire requête/document

Pour évaluer un système, on l'utilise pour produire un classement des documents de la collection pour chacune des requêtes, puis on utilise différentes mesures pour comparer le classement produit avec le classement selon les jugements de pertinence. Dans ce qui suit nous présentons les différentes mesures utilisées pour évaluer le classement produit par un modèle de RI.

Précision@k

La précision@k (notée $P@k$) indique la proportion de documents pertinents parmi les k documents ayant le score le plus élevé :

$$P@k = \frac{|\{\text{top-k documents retournés}\} \cap \{\text{documents pertinents}\}|}{k}. \tag{2.15}$$

Pour les petites valeurs de k (<10), la précision@k permet de savoir à quel point le modèle considéré est capable de retourner des documents pertinents dans le haut du classement. Il s'agit d'un mesure importante à maximiser pour les moteurs de recherche puisque les utilisateurs voudraient avoir tous les documents pertinents sur la première page (Manning et al., 2008).

Rappel

Le rappel indique la proportion de documents pertinents que le modèle est parvenu à retrouver :

$$\text{rappel} = \frac{|\{\text{documents retournés}\} \cap \{\text{documents pertinents}\}|}{|\{\text{documents pertinents}\}|}. \tag{2.16}$$

Un rappel égal à 1 indique que tous les documents pertinents ont été retournés. Le rappel est important à maximiser pour certaines applications dans lesquelles l'utilisateur accorde beaucoup d'importance à retrouver l'ensemble des résultats pertinents même au prix d'une faible précision.

1. Généralement composé au minimum de 50 requêtes

Mean Average Precision

La *Mean Average Precision* (MAP) est une mesure prenant en compte la précision@k pour toutes les valeurs de k :

$$\text{AP} = \frac{\sum_k P@k}{|\{\text{documents pertinents}\}|}$$

La MAP est une mesure prenant en compte à la fois le rappel et la précision puisqu'elle peut être interprétée comme l'aire sous la courbe précision/rappel. Puisqu'elle prend en compte à la fois le rappel et la précision, la MAP est une mesure évaluant la globalité du classement d'un système de RI.

nDCG@k

La nDCG@k (Normalized Discounted Cumulative Gain) (Järvelin et Kekäläinen, 2002) est une mesure évaluant le classement des k premiers documents retournés par le système. Contrairement à la précision@k, la nDCG@k prend en compte différents niveaux de pertinences et donne plus d'importance aux documents en haut du classement en pénalisant les documents en bas du classement :

$$\text{nDCG@k} = \frac{\text{DCG@k}}{\text{IDCG@k}} = \frac{\sum_{i=1}^{k} \frac{\text{rel}_i}{\log(i+1)}}{\text{IDCG@k}}, \tag{2.17}$$

avec rel_i le niveau de pertinence du document en $\text{i}^{\text{ème}}$ position du classement et IDCG@k le DCG@k du classement idéal. La présence de l'IDCG@k dans le dénominateur garanti que la nDCG@k soit comprise entre 0 et 1. Le logarithme au dénominateur de la DCG@k garanti le fait que la nDCG puisse distinguer de façon consistante des fonction de classement différentes (Wang et al., 2013).

2.3 Disparité des termes

Les modèles basés sur la correspondance exacte ne prennent en compte que les documents contenant des termes de la requête. Comme expliqué précédemment, cela leur permet d'être compatibles avec un index inversé rendant la recherche rapide, mais ces modèles souffrent d'un inconvénient majeur : la disparité des termes (*vocabulary mismatch*). En se basant principalement [1] sur la correspondance exacte

1. D'autres informations sont prises en compte telles que l'idf ou la longueur du document, mais le tf est le facteur le plus influent.

entre termes pour ordonner les documents, les modèles basés sur la correspondance exacte sont incapables de prendre en compte le fait que des termes soient polysémiques ou bien la présence de synonymes de termes de la requête dans le document ou encore le fait que l'utilisateur s'exprime dans un vocabulaire de néophyte et que les documents soient rédigés par des experts utilisant un vocabulaire spécifique. Dans les sections suivantes, nous décrivons les solutions à la disparité des termes utilisées dans le cadre ce cette thèse : l'expansion de requêtes, l'utilisation de ressources sémantiques et l'apprentissage profond.

2.3.1 Expansion de requêtes

Comme décrit dans la section 2.2.3, les méthodes d'expansion de requêtes consistent à ajouter des termes à la requête [1] dans le but d'améliorer la recherche. Il est possible d'étendre la requête en utilisant les termes des documents considérés comme pertinents (Rocchio, 1971), des bases de connaissances (Dalton et al., 2014) ou bien des vecteurs de mots appris à l'aide de modèles d'apprentissage automatique (Almasri et al., 2016).

Puisqu'elles ne font que modifier la requête, un avantage des méthodes d'expansion de requêtes vient du fait qu'elles sont indépendantes des modèles de RI utilisés après modification de la requête. Il est donc possible d'utiliser des modèles efficients basés sur la correspondance exacte sur la requête reformulée et, à condition que cette dernière ne soit pas trop longue, de bénéficier d'une recherche rapide. De plus, si les bons termes sont ajoutés à la requête, un modèle de correspondance exacte peut contourner le problème de disparité des termes. En pratique, il est néanmoins difficile de choisir quels termes ajouter à la requête, ce qui entraîne un risque de dégradation des résultats par rapport à la requête originale.

2.3.2 Ressources sémantiques

Plusieurs modèles ont déjà été proposés afin de s'adresser au problème de disparité des termes à l'aide de ressources sémantiques. En effet, les ressources sémantiques permettraient d'identifier les synonymes d'un terme, les différents sens qu'un terme pourrait avoir ou encore les différents termes correspondant au même concept/entité. Plusieurs stratégies ont été expérimentées afin de prendre en compte ces informations dans le cadre de la RI :

1. En théorie, il est également possible d'enlever des termes ou de reformuler la requête, mais en pratique les modèles ne font que rajouter des termes à la requête.

1. **Modèles logiques.** Les modèles logiques (van Rijsbergen, 1986; Sebastiani, 1994) cherchent à estimer la probabilité qu'un document implique une requête $\mathbb{P}(d \Rightarrow r)$ en s'appuyant sur des connaissances explicites tirées de ressources sémantiques.

2. **Extensions de requêtes.** Comme décrit précédemment, il est également possible d'utiliser une ressource sémantique pour étendre les requêtes. Par exemple, le modèle d'expansion de requêtes à l'aide d'entités (Dalton et al., 2014) utilise les connections entre éléments des ressources sémantique pour étendre les requêtes avec des entités et a été étudié en profondeur dans le domaine médical par Jimmy et al. (2019). Ils montrent que de telles méthodes nécessitent plusieurs choix clés et des décisions de conception pour être efficaces et sont donc difficiles à utiliser en pratique.

3. **Ajout de nouvelles caractéristiques.** Il est également possible d'utiliser des ressources sémantiques pour ajouter de nouvelles dimensions aux vecteurs de caractéristiques (*feature vectors*) utilisés dans les modèles d'apprentissage statistiques de type *learning-to-rank* (Soldaini et Goharian, 2017).

4. **Apprentissage non-supervisé.** Récemment, Tamine et al. (2019) ont proposé d'apprendre de manière jointe des représentations vectorielles de mots, de concepts et de documents à l'aide de réseaux de neurones non supervisés. Les représentations vectorielles de documents ainsi apprises prennent en compte les contenus textuel et conceptuel des documents et sont utilisées pour calculer des similarités requête/document.

2.3.3 Apprentissage profond

Les modèles d'apprentissage profond ont également été souvent proposés pour résoudre le problème de la disparité des termes (Mitra et al., 2017; Pang et al., 2017; Fan et al., 2018). En effet, la façon dont les réseaux de neurones calculent le score de pertinence entre une requête et un document (voir Chapitre 3) permet de prendre en compte des relations complexes entre les termes de la requête et du document. À la condition d'avoir assez de données étiquetées, les modèles d'apprentissage profond supervisés sont, à ce jour, les modèles de RI donnant les meilleurs résultats et c'est sur ce type de modèle que ce travail de thèse va se concentrer.

2.4 Conclusion

Dans ce chapitre, nous avons décrit le fonctionnement global d'un système de RI "traditionnel". Le but d'un système de RI est de classer des documents en fonction de leur pertinence par rapport au besoin de l'utilisateur de façon efficiente. Nous avons vu que pour être efficient, les modèles utilisés par un système de RI doivent être basés sur la corresponde exacte. Cela permet l'utilisation efficient d'un index inversé et donc d'avoir une recherche rapide. Les modèles basés sur la correspondance exacte présentent néanmoins des faiblesses, l'une d'entre elles étant la disparité des termes. Une solution à la disparité des termes est d'utiliser des réseaux de neurones pour la RI qui sont capables de modéliser des relations complexes entre termes. Dans le chapitre suivant, nous allons présenter les réseaux de neurones et leur utilisation pour le TALN.

Chapitre 3

Réseaux de neurones pour le TALN

3.1 Introduction

Les réseaux de neurones sont des modèles mathématiques paramétrables et différentiables utilisés en apprentissage automatique, branche de l'informatique utilisant des approches statistiques pour donner aux ordinateurs la capacité d'apprendre à partir de données. Les réseaux de neurones consistent en une succession de couches composées d'une opération linéaire (produit matriciel, convolution, ...) et d'une transformation non-linéaire (sigmoïde, ReLU, ...). L'architecture d'un réseau de neurones réfère à l'organisation des différentes couches le composant. Par exemple, un type d'architecture populaire en traitement d'image est le CNN (*Convolutional Neural Network*) qui est composé d'une succession de couches de convolutions suivie d'une succession de couches dites *fully connected*.

Dans la sous-section suivante, afin d'illustrer concrètement ce qu'est un réseau de neurones, nous présentons le Perceptron Multicouche (de l'anglais *Multiayer Perceptron* MLP) pour deux raisons :

— il s'agit d'un modèle conceptuellement simple et donc adapté à l'illustration

— il s'agit d'un modèle fréquemment utilisé dans de nombreuses architectures (voir section 4.5)

3.1.1 Perceptron Multicouche

Le Perceptron Multicouche (MLP de l'anglais *MultiLayer Perceptron*) (Ivakhnenko et Lapa, 1966) est un réseau de neurones composé d'au moins 3 couches : une couche d'entrée, une ou plusieurs couches cachées et une couche de sortie. La couche d'entrée est passive : elle n'applique pas d'opération mathématiques et ne

fait que transférer l'entrée à la première couche cachée. Les couches cachées et la couche de sortie prennent en entrée la sortie de la couche précédente, lui applique un produit matriciel, ajoute un biais et applique une opération non-linéaire qui, dans la majorité des cas, est une tangente hyperbolique, une sigmoïde ou une unité linéaire rectifiée (ReLU de l'anglais *Rectified Linear Unit* (Malik et Perona, 1990)). De façon plus formelle, la sortie de la k^{ème} couche (noté $X^{(k)}$) d'un MLP se calcule de la façon suivante :

$$X^{(k)} = \sigma\left(b^{(k)} + W^{(k)}X^{(k-1)}\right), \tag{3.1}$$

avec $b^{(k)}$ et $W^{(k)}$ le biais et la matrice associés à la k^{ème} couche cachée dont les coefficients font partie des paramètres du MLP. σ dénote une fonction non linéaire. Un MLP répète ce type d'opération autant de fois qu'il y a de couches cachées jusqu'à la couche de sortie. Par exemple, étant donné le vecteur $X \in \mathbb{R}^n$ en entrée, un MLP à deux couches cachées retournera le vecteur $Y \in \mathbb{R}^m$ calculé comme suit :

$$Y = \sigma\left(b^{(3)} + W^{(3)}\underbrace{\sigma\left(b^{(2)} + W^{(2)}\underbrace{\sigma\left(b^{(1)} + W^{(1)}X\right)}_{X^{(1)}}\right)}_{X^{(2)}}\right), \tag{3.2}$$

avec $X^{(1)} \in \mathbb{R}^{h_1}$ la sortie de la première couche cachée et $X^{(2)} \in \mathbb{R}^{h_2}$ la sortie de la seconde couche cachée. h_1 et h_2 sont les hyperparamètres correspondants à la taille de la première couche intermédiaire et de la seconde couche intermédiaire respectivement. L'architecture d'un tel MLP est illustrée sur la Figure 3.1 avec $n = 2$, $h_1 = 3$, $h_2 = 4$ et $m = 1$

La sortie de chaque couche d'un MLP (et plus généralement d'un réseau de neurones à plusieurs couches) peut être vue comme un vecteur de caractéristiques représentant les données en entrée. Plus la couche est profonde dans l'architecture, plus le vecteur de caractéristiques qu'il renvoie est difficilement interprétables par un humain. Ce sont ces différents niveaux de représentations qui permettent aux réseaux de neurones de modéliser les données en entrée de façon utile pour la tâche considérée (LeCun et al., 2015). La principale différence entre l'apprentissage profond à base de réseaux de neurones et les méthodes traditionnelles d'apprentissage automatique est que ces dernières modélisent les données à l'aide de vecteurs de caractéristiques dont les règles de construction sont définies par des experts du domaine de la tâche à résoudre tandis que les réseaux de neurones apprennent automatiquement des vecteurs de caractéristiques adaptés à la tâche considérée.

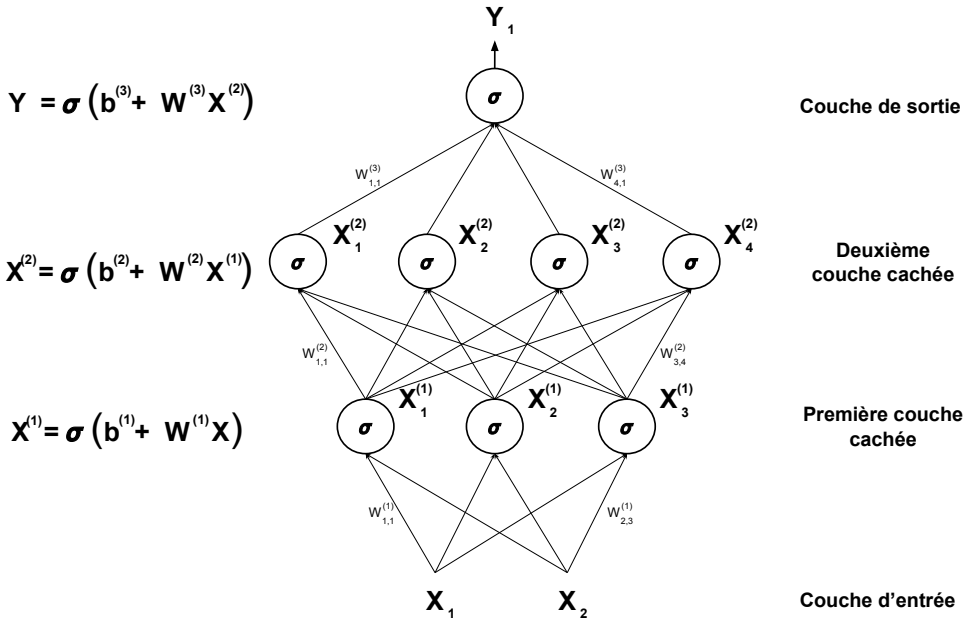

FIGURE 3.1 – Architecture d'un MLP de $\mathbb{R}^2$ dans $\mathbb{R}$ à deux couches cachées de tailles $h_1 = 3$ et $h_2 = 4$.

Pour avoir une compréhension visuelle et intuitive du fonctionnement d'un MLP, et pour constater la différence entre les vecteurs de caractéristiques appris pour une tâche et ceux fabriqués par des experts pour la même tâche, le lecteur peut utiliser le programme disponible à l'adresse https://playground.tensorflow.org et comparer les vecteurs (X_1, X_2) et (X_1^2, X_2^2) sur le jeu de données *circle*.

3.1.2 Apprentissage

De façon générale, afin d'apprendre à résoudre une tâche, un système utilisant un réseau de neurones f à besoin de :

1. Un ensemble de données X
2. Un ensemble d'étiquettes Y associées aux données que f doit prédire
3. Une fonction objectif $\mathcal{L}$ mesurant la différence entre la sortie du réseau de neurones $f(X^{(i)})$ et l'étiquette associée $Y^{(i)}$

Le but de la phase d'apprentissage est de trouver les paramètres de f qui minimisent $\mathcal{L}$ sur l'ensemble des paires $X^{(i)}, Y^{(i)}$:

$$f^* = \arg\min_f \sum_i \mathcal{L}\left(f(X^{(i)}), Y^{(i)}\right). \tag{3.3}$$

Les méthodes d'optimisation utilisées pour trouver les paramètres optimaux d'un réseau de neurones sont des méthodes basées sur la descente de gradient (LeCun et al., 2015). De façon générale, étant donné une fonction f la descente de gradient consiste à chercher un point x^* pour lequel f est minimal. Pour ce faire, on commence par choisir un point x_0 de façon aléatoire, puis on calcule le gradient de f en x, noté $\nabla f(x)$ et on met à jour x dans la direction opposée au gradient de f :

$$x_1 = x_0 - \gamma \nabla f(x_0), \tag{3.4}$$

avec $\gamma \in \mathbb{R}^+$ la vitesse d'apprentissage (*learning rate*). Ce procédé est répété jusqu'à ce que la norme du gradient calculé passe en dessous d'un seuil fixé en avance. Dans le cas des réseaux de neurones, la fonction que l'on cherche à minimiser est la fonction objectif et le point x mentionné précédemment correspond à l'ensemble des paramètres du réseau de neurones. En pratique, des méthodes plus complexes que celle décrite à l'équation (3.4) sont utilisées [1], mais elles sont toutes basées sur le calcul du gradient de la fonction objectif et sur son utilisation pour mettre à jour les paramètres du réseau de neurones.

3.1.3 Différentiabilité

Afin de pouvoir optimiser un réseau de neurones en utilisant des méthodes basées sur la descente de gradient, il faut pouvoir calculer le gradient de la fonction objectif en fonction des paramètres du réseau de neurones. Cela signifie que l'entrée des réseaux de neurones doit être composée de nombres réels et que les opérations utilisées doivent être différentiables.

La particularité des réseaux de neurones par rapport aux méthodes d'apprentissage automatique traditionnelles vient du fait que les réseaux de neurones utilisent les données brutes et apprennent à modéliser ces dernières afin de résoudre une tâche tandis que les méthodes traditionnelles nécessitent des experts définissant des règles pour modéliser les données brutes en représentations compatibles avec la méthode considérée. Par exemple, un réseau de neurones pour la RI prend en entrée les textes bruts de la requête et du document tandis qu'un modèle non neuronal

1. par exemple *Adam* (Kingma et Ba, 2015) utilise également l'historique des gradients calculées pour mettre à jour x

(*RankSVM* Joachims (2002)) prend en entrée un vecteur de caractéristiques [1] dont la construction suit des règles définies par des spécialistes de la RI.

Comme nous l'avons mentionné précédemment, l'entrée d'un réseau de neurones doit être composé de nombres réels à partir desquels il est possible de calculer un gradient. Ceci n'est pas problématique lorsque les données elles-mêmes sont composées de valeurs réelles (image, vidéo, signal sonore, ...) mais une étape de "plongement" dans un espace de réels est nécessaire lorsque des données symboliques sont considérées (texte, graphe, arbre, ...).

Dans les deux sections suivantes, nous présentons les méthodes de plongement de graphe relationnel et de plongement de texte utilisées dans cette thèse.

3.2 Représentations continues de graphe relationnel

3.2.1 Introduction

Dans ce travail de thèse, nous avons utilisé la méthode de plongement de graphe relationnel *TransE* Bordes et al. (2013). *TransE* a été développé afin de modéliser des données multi-relationnelles associées à des graphes dont les nœuds sont des entités et les arêtes sont des types de relations. Ces données sont représentées sous la forme d'un ensemble de triplets (e_s, r, e_c) avec e_s l'entité source, r la relation et e_c l'entité cible.

Contrairement à des méthodes de plongements de graphe utilisant des marches aléatoires telles que *node2vec* (Grover et Leskovec, 2016) et *DeepWalk* (Perozzi et al., 2014) ou factorisant la matrice d'adjacence (Ahmed et al., 2013) qui associent des vecteurs seulement aux nœuds des graphes, *TransE* propose également de d'associer des vecteurs [2] aux relations. *TransE* associe chaque type de relation à une translation dans un espace vectoriel dans lequel les entités sont plongées de sorte que si le triplet (e_s, r, e_c) est présent dans le graphe relationnel, alors le vecteur de l'entité cible $\overrightarrow{e_c}$ soit proche du vecteur de l'entité source $\overrightarrow{e_s}$ translaté à l'aide du vecteur de relation $\overrightarrow{r}$:

$$\overrightarrow{e_c} \approx \overrightarrow{e_s} + \overrightarrow{r}. \tag{3.5}$$

1. les caractéristiques sont censées contenir des informations utiles pour caractériser la pertinence du document par rapport à la requête considérée
2. dans le même espace vectoriel que les vecteurs de nœuds.

3.2.2 Entraînement

Étant donné un graphe relationnel $\mathcal{G}$ associé à un ensemble $\mathcal{T}$ de triplets composé de deux entités $e_s, e_c \in \mathcal{E}$ (l'ensemble des entités) et d'une relation $r \in \mathcal{R}$ (l'ensemble des types de relations), *TransE* associe à chaque entité et à chaque relation un vecteur dans $\mathbb{R}^k$. Les vecteurs sont appris en minimisant la fonction objectif $\mathcal{L}$ suivante :

$$\mathcal{L} = \sum_{(e_s,r,e_c)\in\mathcal{T}} \sum_{(e'_s,r,e'_c)\in\mathcal{T}'} \max\left(0, m + ||\overrightarrow{e_s} + \overrightarrow{r} - \overrightarrow{e_s}||_2 - ||\overrightarrow{e'_s} + \overrightarrow{r} - \overrightarrow{e'_s}||_2\right), \quad (3.6)$$

avec $m \in \mathbb{R}^+$ la marge [1], $||\cdot||_2$ la norme L_2 et $\mathcal{T}'_{(e_s,r,e_c)}$ un ensemble de triplets négatifs dont les éléments sont construits en remplaçant une entité du triplet (e_s, r, e_c) par une entité aléatoire :

$$\mathcal{T}'_{((e_s,r,e_c))} = \{(e'_s, r, e_c) | e'_s \in E\} \cup \{(e_s, r, e'_c) | e'_c \in E\}. \quad (3.7)$$

Par conséquent, *TransE* est entraîné en minimisant la distance entre $\overrightarrow{e_s} + \overrightarrow{r}$ et $\overrightarrow{e_s}$ pour les triplets $(e_s, r, e_c) \in \mathcal{T}$ et en maximisant la distance entre $\overrightarrow{e'_s} + \overrightarrow{r}$ et $\overrightarrow{e'_s}$ pour les triplets négatifs (e'_s, r, e'_c) ne faisant pas parti de $\mathcal{T}$. Les triplets négatifs sont utilisés dans la fonction objectifs afin d'éviter que le modèle converge vers une solution triviale où tous les vecteurs sont nuls. La Figure 3.2 montre les vecteurs appris par *TransE* en utilisant le triplet (**Jarnac**, **birthPlace**, **François Mitterand**) et le triplet négatif (**Jarnac**, **birthPlace**, **Achilles**).

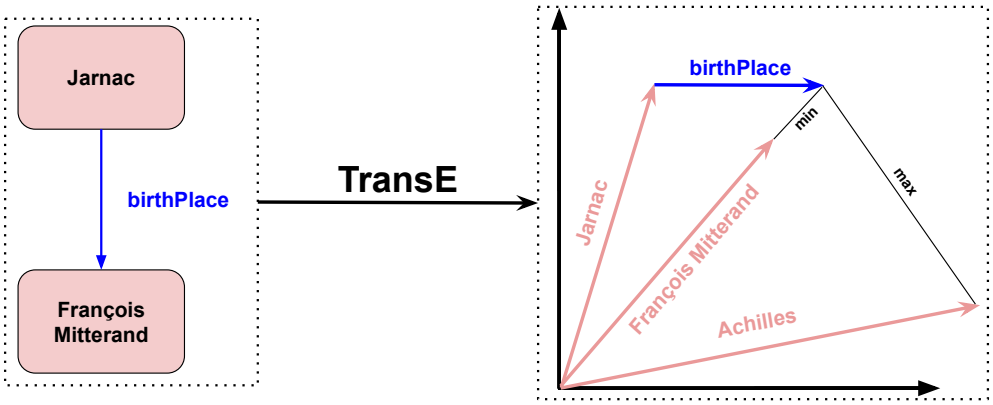

FIGURE 3.2 – Illustration des vecteurs TransE.

1. la valeur de m est un hyperparamètre fixé avant l'entraînement.

Dans ce travail de thèse, nous avons utilisé *TransE* afin de plonger des graphes relationnels, car *TransE* a été conçu pour plonger les graphes multi-relationnels [1] et parce qu'il s'agit d'une méthode qui passe à l'échelle : la complexité de la phase d'apprentissage est linéaire par rapport aux nombre de triplets. Il est possible de plonger un graphe à plusieurs millions de nœuds dans un espace vectoriel en moins de 48h à l'aide d'un CPU Intel(R) i5-6500 à 4 cœur et 16G de RAM.

3.3 Représentations continues du texte

Les deux modèles présentés dans ce qui suit sont des modèles de langue neuronaux. Un modèle de langue neuronal est un réseau de neurones modélisant la distribution de séquence de mots dans un langage naturel (Bengio, 2008). De façon plus concrète, étant donné une séquence de mots $m_1, m_2, m_3, ...m_T$, les modèles de langue neuronaux sont entraînés en maximisant la probabilité d'observer les mots de la séquence sachant leur contexte :

$$\prod_{t=1}^{T} \mathbb{P}\left(m_t | c_t\right), \tag{3.8}$$

avec c_t le contexte du mot m_t. La définition exacte du contexte d'un mot varie en fonction du modèle de langue considéré.

L'idée d'utiliser des modèles de langue pour apprendre des vecteurs de mots est tirée de l'hypothèse distributionnelle : " *les mots qui occurrent dans les mêmes contextes tendent à avoir des sens similaires* " (Harris, 1954). En effet, l'estimation de $\mathbb{P}\left(m_t | c_t\right)$ se fait en deux étapes : **(1)** m_t est associé à un vecteur x_{m_t} ; **(2)** un réseau de neurones est utilisé pour calculer $\mathbb{P}\left(m_t | c_t\right)$ en utilisant x_{m_t}. Ce dernier, ainsi que les paramètres du réseau de neurones sont ensuite optimisés pour mieux approximer $\mathbb{P}\left(m_t | c_t\right)$ en fonction des séquences des mots d'apprentissage. Par conséquent si deux mots m_i et m_j apparaissent dans des contextes similaires dans les séquences d'apprentissage, leurs vecteurs x_{m_i} et x_{m_j} seront mis à jour de façon similaire lors de la phase d'apprentissage. Par conséquent, m_i et m_j qui ont des sens similaires selon l'hypothèse distributionnelle auront également des vecteurs similaires si ils sont entraînés à l'aide de modèles de langue.

1. d'autres méthodes telles que nod2vec ou les GCN (Kipf et Welling, 2017) ont été conçues pour les graphes et ne prennent pas en compte les différents types de relations présents dans un graphe multi-relationnel

3.3.1 Word2vec

Word2vec (Mikolov et al., 2013a) est un groupe de deux modèles de langues neuronaux : *skip-gram* et *CBOW* (*continuous bag-of-words*). Dans ce travail de thèse nous n'avons utilisé que le modèle *skip-gram*, c'est donc celui que nous allons décrire dans cette section [1].

Étant donné une séquence d'apprentissage $m_1, m_2, m_3, ...m_T$ et un vocabulaire V associé, *skip-gram* est entraîné en minimisant la log-vraisemblance négative d'observer le contexte d'un mot de la séquence d'apprentissage sachant le mot lui-même [2] :

$$-\sum_{t=1}^{T} \log \mathbb{P}\left(c_t | m_t\right). \tag{3.9}$$

Dans le cas de *skip-gram*, le contexte d'un mot est défini comme un des mots dans son voisinage. Par exemple dans le cas d'un voisinage de taille 2, on a :

$$c_t \in \left\{m_{t-2}, m_{t-1}, m_{t+1}, m_{t+2}\right\}. \tag{3.10}$$

Les paramètres de *skip-gram* sont composés de deux matrices : une matrice de mots $W^{(m)} \in \mathbb{R}^{|V| \times d}$ et une matrice de contexte $W^{(c)} \in \mathbb{R}^{|V| \times d}$ dont chacune des lignes correspond respectivement à un vecteur de mot et à un vecteur de contexte, chacun de dimension d. La probabilité d'observer le contexte c_t sachant le mot m_t est calculée en effectuant un Softmax sur le produit scalaire entre le vecteur de mot de m_t et le vecteur de contexte de c_t :

$$\mathbb{P}\left(c_t | m_t\right) = \text{Softmax}\left(x_{m_t} \cdot x_{c_t}\right) = \frac{\exp\left(x_{m_t} \cdot x_{c_t}\right)}{\sum\limits_{m' \in V} \exp\left(x_{m'} \cdot x_{c_t}\right)}, \tag{3.11}$$

avec $x_{m_t} = W^{(m)}_{m_t}$ le vecteur de mot de m_t correspondant à la ligne de $W^{(m)}$ associée au mot m_t et $x_{c_t} = W^{(c)}_{c_t}$ le vecteur de contexte de c_t correspondant à la ligne de $W^{(c)}$ associée au mot c_t. La probabilité d'observer le contexte le contexte c_t sachant le mot m_t en fonction des matrices $W^{(m)}$ et $W^{(c)}$ est donc calculée comme suit :

1. Les deux modèles sont néanmoins très similaires. Pour en savoir plus, le lecteur peut se référer à l'article original de Mikolov et al. (2013a)

2. Il est possible d'utiliser la formule de Bayes pour calculer $\mathbb{P}\left(m_t | c_t\right)$ à partir de $\mathbb{P}\left(c_t | m_t\right)$ afin de pouvoir correspondre à la définition d'un modèle de langue

$$\mathbb{P}\left(m_t|c_t\right) = \frac{\exp\left(W^{(m)}_{m_t} \cdot W^{(c)}_{c_t}\right)}{\sum\limits_{m' \in V} \exp\left(W^{(m)}_{m'} \cdot W^{(c)}_{c_t}\right)}, \tag{3.12}$$

L'apprentissage des paramètres de *skip-gram* se fait :

1. en faisant passer une fenêtre glissante sur chacun des mots des séquences d'entraînement
2. en estimant la probabilité d'observer un mot de la fenêtre sachant le mot au centre de la fenêtre (voir Figure 3.3)
3. en mettant à jour les coefficients de $W^{(m)}$ et $W^{(c)}$ pour que la probabilité estimée corresponde à la probabilité observée.

Lorsque l'apprentissage est terminé, la matrice $W^{(m)}$ est utilisée pour extraire des vecteurs de mots qui seront ensuite donnés en entrée de réseaux de neurones pour le TALN ou la RI textuelle. L'utilisation de la matrice de contexte $W^{(c)}$ a été étudié (notamment pour la RI) mais reste marginale (Mitra et Craswell, 2018).

Bien que les premiers modèles de langues neuronaux (Bengio et al., 2000) soient antérieurs à Word2Vec, ce dernier est devenu populaire dans le monde de la recherche [1] en raison de sa simplicité, du fait que les vecteurs aient été mis en libre accès et qu'il est possible d'entraîner les vecteurs de façon efficiente [2] sans GPU à l'aide de l'implémentation en C fournie par les auteurs [3].

1. les deux articles (Mikolov et al., 2013a) et (Mikolov et al., 2013b) introduisant Word2Vec ont été cités 18517 et 23301 fois respectivement au 24/10/2020
2. moins de 24h pour entraîner Word2vec sur Wikipedia en anglais à l'aide d'un CPU Intel(R) i5-6500 à 4 core et 16G de RAM.
3. https://github.com/tmikolov/word2vec/blob/master/word2vec.c

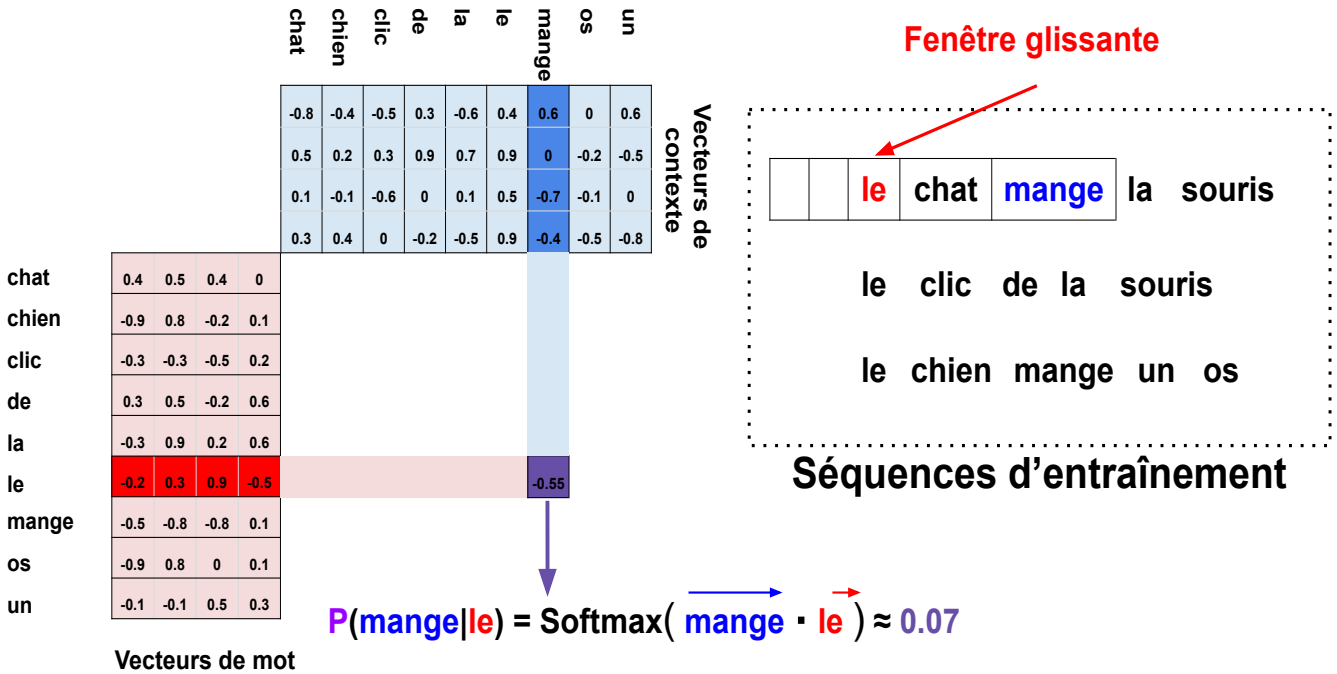

FIGURE 3.3 – Illustration de la phase d'apprentissage du modèle *skip-gram* dans le cas d'un voisinage de taille 2. Afin de simplifier l'illustration, le détail du calcul du Softmax n'a pas été précisé.

3.3.2 FastText

*fast*Text est un modèle de langue neuronale proposant de résoudre deux limitations de *Word2vec* (Bojanowski et al., 2017) :

— *Word2vec* n'est pas capable d'associer un vecteur à un mot inconnu n'occurrant pas dans les séquences d'entraînements.

— *Word2vec* ne prend pas en compte la structure interne des mots. Par exemple les vecteurs associés aux mots "*mange*", "*manger*" et "*manges*" dépendent uniquement de leur contexte. Le fait qu'ils aient une racine commune n'a aucune influence sur leur représentation.

Afin de résoudre ces limitations, *fast*Text apprend des vecteurs de n-grammes de caractères plutôt que des vecteur de mots. Plus concrètement, chaque mot est modélisé comme un *sac-de-mots* de n-grammes de caractères et le vecteur qui lui est associé est calculé en additionnant les vecteurs de n-grammes de caractères qui le composent. Par exemple, dans le cas $n = 3$, le mot "*mange*" est associé à l'ensemble de 3-grammes de caractères suivant :

$$\{\text{“}<ma\text{”}, \text{“}man\text{”}, \text{“}ang\text{”}, \text{“}nge\text{”}, \text{“}ge>\text{”}\},$$

avec "<" et ">" les symboles indiquant respectivement le début et la fin d'un mot. Le vecteur du mot "*mange*" est calculé comme suit :

$$\overrightarrow{mange} = \overrightarrow{<ma} + \overrightarrow{man} + \overrightarrow{ang} + \overrightarrow{nge} + \overrightarrow{ge>} \tag{3.13}$$

Le reste du modèle est strictement identique à *skip-gram* : les paramètres de *fast*Text sont organisés en une matrice de n-gramme de mots et une matrice de n-gramme de contexte et la probabilité d'observer un contexte sachant un mot est calculé en effectuant un Softmax sur le produit scalaire entre le vecteur de mot et le vecteur de contexte.

En utilisant des vecteurs de n-grammes de caractères, *fast*Text est capable de :

— associer un vecteur à des mots ne faisant pas parti du vocabulaire des séquences d'entraînement

— s'assurer que des mots ayant une même racine aient des vecteurs similaires (par exemple "*manger*" et "*manges*")

— d'apprendre de meilleures représentations des mots rares que *Word2Vec* (voir Tableau 3.1)

	skip-gram	*fast*Text
Corrélation de Spearman	43	47

TABLE 3.1 – Comparaison entre les vecteurs de mots *skip-gram* et *fast*Text sur le jeux de données de similarité entre mots rares : *Rare Word Dataset* (Luong et al., 2013). La corrélation de Spearman (Spearman, 1904) est effectuée entre les jugements humain et la similarité cosinus entre vecteurs de mots.

3.4 LSTM pour le TALN

3.4.1 Réseaux de neurones récurrents pour le TALN

Les réseaux de neurones récurrents (RNN de l'anglais *recurrent neural network*) sont une classe de réseaux de neurones qui en plus de traiter un élément en entrée prennent également en compte l'historique des entrées précédentes sous forme d'état interne. Par conséquent, les RNNs sont utilisés pour traiter des séquences et le traitement de chaque élément de la séquence sera fait en prenant en compte les éléments précédents. Les RNNs sont utilisés dans une variété de tâches nécessitant l'analyse et le traitement de séquences telles que les modèles de langue (Peters et al., 2018), la reconnaissance automatique de la parole (Graves et al., 2013), la traduction automatique (Bahdanau et al., 2015), la recherche d'information (Wan et al., 2016), la génération de description d'images (You et al., 2016), la prédiction de séries temporelles (Connor et al., 1994) etc...

Étant donné une séquence de vecteurs de mots $X = (x_{m_1}, ..., x_{m_T})$ à traiter/analyser par un RNN, ce dernier est appliqué à chacun des éléments de la séquence les uns à la suite des autres et, contrairement aux réseaux de neurones non-récurrents, prend deux entrée et retourne deux sorties. Un RNN retourne un vecteur de sortie y_t mais également un vecteur d'état interne h_t modélisant la mémoire du RNN. En entrée, un RNN prend un élément x_{m_t} de la séquence X et l'état interne associé à la sortie précédente h_{t-1}. Généralement, le premier état interne h_0 est égal au vecteur nul, ou bien il fait partie des paramètres du modèle et est donc appris. C'est la présence du vecteur d'état interne en entrée du RNN qui permet à ce dernier de prendre en compte les éléments précédents de la séquence.

Un exemple de RNN est le réseau d'Elman (Elman, 1990) dont la conception peut s'apparenter à celle d'un MLP à une couche cachée dont la sortie de la couche cachée est définie comme le vecteur d'état interne :

$$
\begin{aligned}
h_t &= \sigma\left(W_h x_{m_t} + U_h h_{t-1} + b_h\right) \\
y_t &= \sigma\left(W_y h_t + b_y\right)
\end{aligned}
\tag{3.14}
$$

En pratique, les réseaux d'Elman ne sont pas utilisés à cause de la difficulté qu'ils ont à modéliser des dépendances de long terme (Bengio et al., 1994).

3.4.2 LSTM pour le TALN

Les RNNs de type LSTM (de l'anglais *Long Short Term Memory*) ont été conçus afin de pouvoir modéliser des dépendances de long terme (Hochreiter et Schmidhuber, 1997). L'idée du LSTM est d'utiliser, en plus de l'état interne, des cellules de mémoire dont le rôle est de modéliser la mémoire à long terme du LSTM dans un vecteur C_t. Lors de l'analyse d'un nouvel élément de la séquence, des portes sont utilisées afin de décider comment modifier et utiliser la mémoire du LSTM. Les portes sont un moyen de laisser passer de l'information de manière facultative. Elles sont composées d'un produit matriciel, d'une sigmoïde et d'un produit élément par élément. LSTM utilise trois portes : une porte d'oubli (*forget gate*), une porte d'entrée (*input gate*) et une porte de sortie (*output gate*). Dans ce qui suit, nous décrivons le rôle et le fonctionnement de chacune des portes.

Porte d'oubli

Le rôle de la porte d'oubli est de décider quels éléments de la mémoire C_{t-1} ne sont plus utiles et de les pondérer en conséquence :

$$
C_t^f = C_{t-1} \odot f_t, \tag{3.15}
$$

avec f_t un vecteur de poids compris entre 0 et 1 dont le rôle est de pondérer les éléments de C_t. Les poids proches de 0 sont associés à des éléments de C_t à oublier et les poids proches de 1 sont associés à des éléments de C_t à préserver. Les poids de f_t sont calculés en fonction des entrées x_{m_t} et h_{t-1} de la façon suivante :

$$
f_t = \sigma\left(W_f\left[h_{t-1}, x_{m_t}\right] + b_f\right), \tag{3.16}
$$

avec σ la fonction sigmoïde, $[h_{t-1}, x_{m_t}]$ la concaténation des vecteurs h_{t-1} et x_{m_t}. W_f et b_f sont des paramètres du LSTM.

Porte d'entrée

La porte d'entrée est utilisée afin de décider quels nouveaux éléments seront rajoutés à la mémoire. Dans un premier temps une mémoire temporaire $\widetilde{C}_t$ est crée en fonction des entrées x_{m_t} et h_{t-1} :

$$\widetilde{C}_t = \tanh\left(W_C\left[h_{t-1}, x_{m_t}\right] + b_C\right), \tag{3.17}$$

avec W_C et b_C des paramètres du LSTM. Puis, de façon similaire à la porte d'oubli, la porte d'entrée est utilisée afin de décider quels éléments de la mémoire temporaire ne seront pas utilisés :

$$i_t = \sigma\left(W_i\left[h_{t-1}, x_{m_t}\right] + b_i\right), \tag{3.18}$$

avec W_i et b_i des paramètres du LSTM. Le nouveau vecteur de mémoire C_t est calculé en sommant le vecteur de mémoire précédent c_{t-1} pondéré par la porte d'oubli avec le vecteur de mémoire temporaire pondéré par la porte d'entrée :

$$C_t = C_{t-1} \odot f_t + \widetilde{C}_t \odot i_t. \tag{3.19}$$

Porte de sortie

Le rôle de la porte de sortie est de décider quels éléments de la mémoire C_t sont utiles dans le calcul du nouvel état interne h_t :

$$\begin{aligned} o_t &= \sigma\left(W_o\left[h_{t-1}, x_{m_t}\right] + b_o\right) \\ h_t &= o_t \odot \tanh\left(C_t\right) \end{aligned} \tag{3.20}$$

La tangente hyperbolique est utilisée afin de garantir que les valeurs de l'état caché soient bornées et pour éviter que les valeurs de gradients tendent vers 0 (Hochreiter et Schmidhuber, 1997). L'architecture globale du LSTM est illustré sur la Figure 3.4.

Le vecteur d'état interne h_t peut ensuite être utilisé directement pour calculer la sortie du modèle de façon similaire aux réseaux d'Elman, mais il peut être également utilisé en entrée d'un autre réseau de neurones ou bien en entrée d'un autre LSTM, dans ce cas, on parle de LSTM multi-couches.

LSTM bidirectionnel

Une limitation du LSTM vient du fait que le calcul des vecteurs des états internes ne prend en compte que les éléments précédents de la séquence, il s'agit

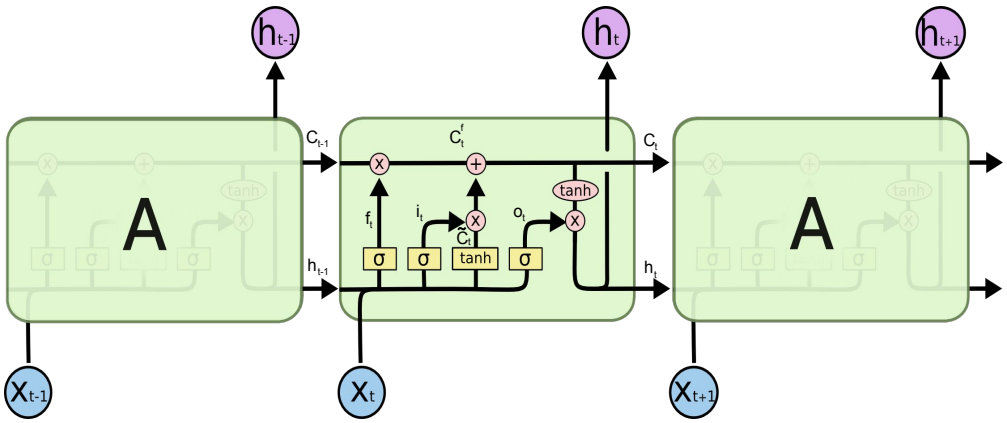

FIGURE 3.4 – Architecture d'un LSTM. Figure issue de https://colah.github.io/posts/2015-08-Understanding-LSTMs/

d'une propriété qui n'est pas nécessaire (et donc restrictive) pour certaines applications comme la modélisation de documents pour la RI. Afin de prendre en compte tous les autres éléments de la séquence dans le calcul du vecteur d'un état interne, en plus du LSTM utilisé comme décrit précédemment, un autre LSTM est utilisé sur la séquence inversée : au lieu d'analyser $(x_{m_1}, x_{m_2}, ..., x_{m_T})$, il analysera $(x_{m_T}, x_{m_{T-1}}, ..., x_{m_1})$. Ce nouvel LSTM prend en compte les éléments futurs de la séquence et en concaténant les états internes des deux LSTM, nous avons bien des représentations prenant en compte tous les autres éléments de la séquence. Dans ce cas, on parle de LSTM bidirectionnel.

3.5 Couches de convolutions pour le TALN

Bien qu'ayant été initialement développés pour la reconnaissance de motifs visuels (*visual pattern recognition*) (Fukushima, 1979), les réseaux de neurones utilisant des couches de convolution sont utilisés dans le cadre du TALN et de la RI textuelle. Deux types de couches de convolutions sont principalement utilisés : les couches de convolutions 1D et les couches de convolutions 2D.

3.5.1 Couches de convolutions 1D

Pour analyser une séquence de vecteur de mots $X = (x_{m_1}, x_{m_2}, ..., x_{m_T})$, les couches de convolutions 1D sont généralement préférées aux couches convolutions 2D. Les couches de convolutions 1D sont utilisées pour produire des représentations

de n-gramme de mots. Elles permettent de capturer les interactions locales entre les mots de la séquence en entrée.

De façon plus concrète, une couche de convolutions 1D est composée de k filtres, chacun étant caractérisé par une matrice F de taille $n \times d$ avec n la taille du filtre et d la dimension des vecteurs de mots. Un produit de Hadamard est appliqué entre F et les sous-séquences de vecteurs de mots de la séquence en entrée, puis les éléments de la matrice résultante sont tous additionnés (voir Figure 3.5). Par exemple, la sortie d'un filtre de convolution associé à la matrice $F \in \mathbb{R}^{3\times d}$ appliquée au premier tri-gramme de la séquence de vecteurs de mots X est calculé comme suit :

$$X^{(1)}_{1,1} = \sum_{i=1}^{3}\sum_{j=1}^{d} \left(X_{[1:3,:]} \odot F\right)_{i,j}. \tag{3.21}$$

Cette opération est répétée pour les k filtres de la couche de convolution, ce qui donne lieu à un vecteur $X^{(1)}_{1,:} \in \mathbb{R}^k$ associé au premier tri-gramme de X. En appliquant les k filtres à tous les tri-gramme de X, on a bien en sortie de la couche de convolutions une séquence de vecteurs $X^{(1)} \in \mathbb{R}^{T-2\times k}$ représentant les tri-grammes de la couche en entrée. Ce type de couche de convolution est dit 1D car la longueur des filtres de convolution est égale à la dimension des éléments de la séquence analysée et que les filtres "parcourent" cette dernière dans une seule direction.

En pratique, afin d'avoir une séquence de vecteurs en sortie de la couche de convolution de même longueur que la séquence en entrée, il est commun d'utiliser la technique de complétion de zéros (*zero-padding*) qui consiste à ajouter des vecteurs nuls au début et à la fin de la séquence en entrée.

Bien que les CNN ne peuvent prendre en compte que des interaction locales (taille du filtre), ils peuvent être préférés aux LSTM en TALN en raison de leur vitesse d'exécution sur des GPUs et aussi, car certaines applications du TALN ne nécessitent pas de modéliser des interactions distantes telles que certaines tâches de détection de sentiments et les systèmes de questions-réponses (Yin et al., 2017).

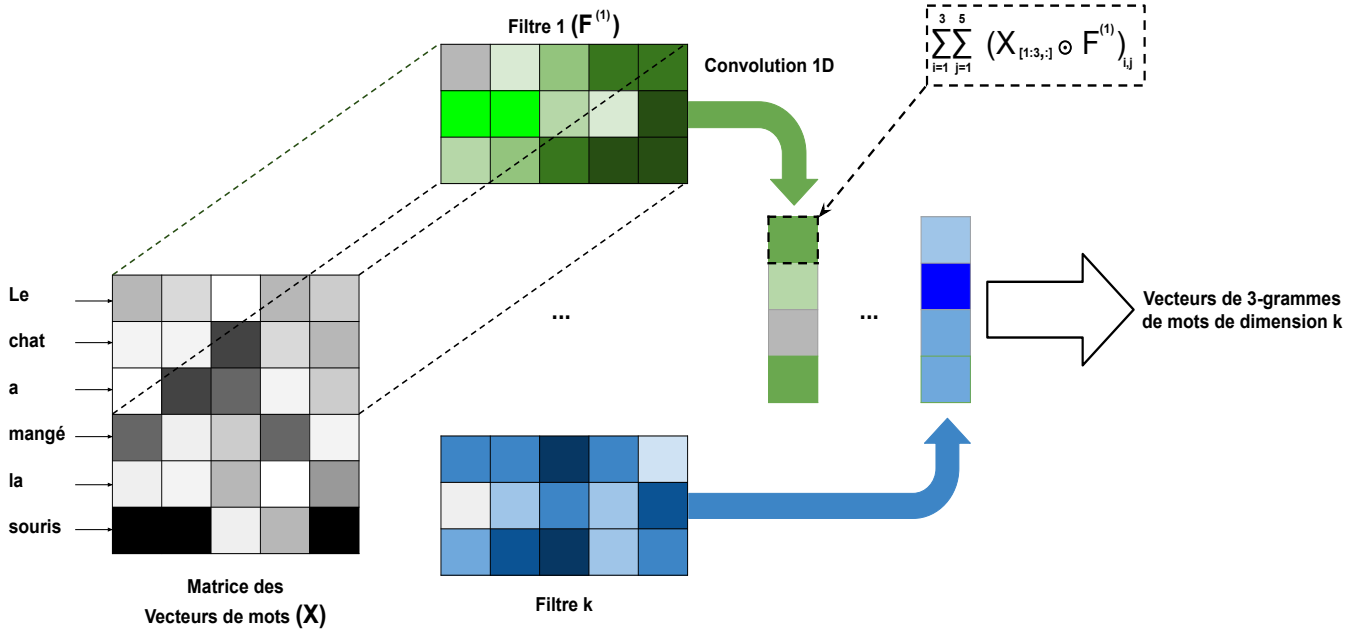

FIGURE 3.5 – Architecture simplifiée d'une couche de convolution 1D appliquée à une séquence de vecteur de mots. $X_{[1:3,:]}$ dénote les 3 premières lignes de la matrice X et $\odot$ dénote le produit élément par élément (ou produit de Hadamard)

3.5.2 Couches de convolutions 2D pour le TALN

Les couches de convolutions 2D sont également utilisées pour le TALN et sont notamment efficaces pour les tâches nécessitant de comparer des séquences telles que la RI, les systèmes de questions-réponses ou bien l'identification de paraphrase. Elles sont utilisées de la même façon qu'en traitement/analyse d'images : Étant donné une matrice [1], une couche de convolution 2D applique un ensemble de k filtres de convolution 2D et retourne k cartes de caractéristiques (*features map*) comme illustré sur la Figure 3.6.

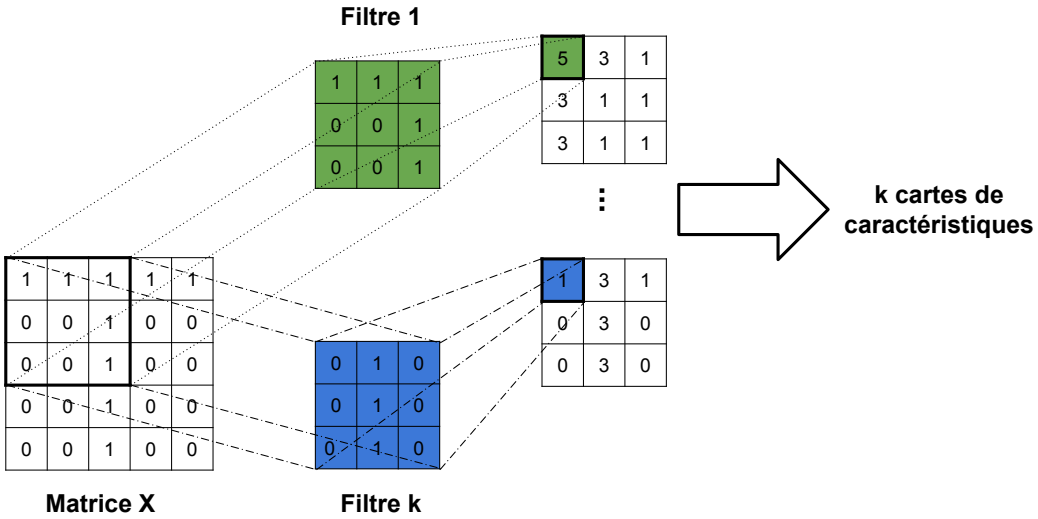

FIGURE 3.6 – Architecture simplifiée d'une couche de convolution 2D composée de k filtres de taille 3×3 appliquée à une Matrice.

Tout comme pour les couches de convolutions 1D, le calcul des filtres de convolution 2D se fait à l'aide d'un produit de Hadamard entre un sous-ensemble de l'entrée et le filtre, suivi de la somme des éléments de la matrice résultante. Par exemple, la Figure 3.6 illustre l'application d'un filtre de convolution 2D (noté F) de taille 3×3 sur les 3 premières lignes et les trois premières colonnes d'une matrice X et la valeur retournée $\left(F * X_{[1:3,1:3]}\right)$ se calcule de la façon suivante :

$$\left(F * X_{[1:3,1:3]}\right) = \sum_{i=1}^{3} \sum_{j=1}^{3} \left(X_{[1:3,1:3]} \odot F\right)_{i,j} . \tag{3.22}$$

1. En RI, il s'agit généralement d'une matrice d'interaction (voir section 4.5).

L'utilisation de couches de convolution 2D permet d'apprendre à identifier certains motifs locaux présents sur la matrice en entrée (LeCun et al., 2015). Dans le cas d'une matrice interaction entre les termes d'une requête et d'un documents, un filtre de convolution 2D de taille 3×3 permet d'analyser toutes les paires de tri-gramme de termes entre la requête et le document et d'identifier certains motifs d'interactions tels que la correspondance exacte.

3.5.3 Fonction d'activation et couches de pooling

En plus de l'utilisation de filtres, les couches de convolutions 1D et 2D font également appel à des fonctions d'activations et des couches dites de *pooling*. Les fonctions d'activations sont appliquées directement sur la sortie des filtres et sont les mêmes que celles utilisées par les autres réseaux de neurones : sigmoïde, tangente hyperbolique et ReLU.

Les couches de *pooling* quant à elles sont utilisées après les fonctions d'activation et ont pour objectif de fusionner les caractéristiques similaires produites par les convolutions pour en réduire la taille (LeCun et al., 2015). Le type de couche de *pooling* le plus commun est le *max-pooling* qui retourne la valeur maximale de son entrée. De la même façon que pour les couches de convolution 1D et 2D de tailles variables, il existe des couches de *pooling* 1D et 2D de tailles variables. Une illustration d'une couche de *max-pooling* 1D de taille 2 et d'une couche de *max-pooling* 2D de taille 2 est présentée sur la Figure 3.7.

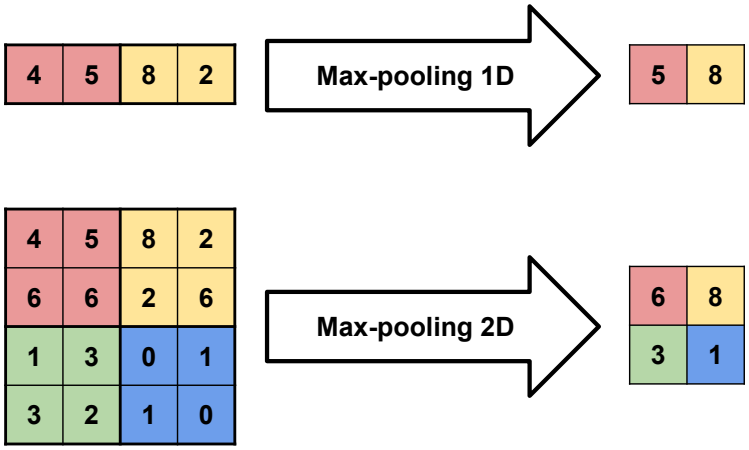

FIGURE 3.7 – Illustration de l'application d'une couche de max-pooling 1D de taille 2 et d'une couche de max-pooling 2D de taille 2.

3.6 Transformers

Transformer est un réseau de neurones ayant été initialement développé pour la traduction automatique (Vaswani et al., 2017) et qui a été adapté aux modèles de langue (Devlin et al., 2019) ce qui a conduit à des avancées des performances des systèmes états de l'art dans de nombreux domaines du TALN tel que la génération automatique de textes (Radford et al., 2019), la désambiguïsation lexicale (Vial et al., 2019), les systèmes de questions-réponses (Lan et al., 2020), la reconnaissance d'entités nommées (Devlin et al., 2019) ou l'analyse de sentiment (Sun et al., 2019). Bien que l'architecture originale de *Transformer* pour la traduction automatique était de type encodeur-décodeur (Sutskever et al., 2014; Vaswani et al., 2017), les tâches de TALN ne nécessitant pas la génération de séquences (RI incluse) n'ont fait l'usage que de l'encodeur *Transformer*. Par conséquent, dans cette thèse, nous ne nous intéresserons qu'à l'encodeur *Transformer*.

3.6.1 Vue d'ensemble d'un encodeur *Transformers*

Étant donné une séquence de T vecteurs de mots $X = (x_{m_1}, ..., x_{m_T})$, l'encodeur *Transformer* génère une nouvelle séquence de T vecteurs contextualisés $Z = (z_{m_1}, ..., x_{m_T})$. Ces vecteurs sont dits contextualisés car le vecteur d'un élément dépendra d'une combinaison linéaire des vecteurs des éléments de son contexte. Dans la section suivante, nous allons décrire le mécanisme d'auto-attention que *Transformer* utilise afin de produire les vecteurs contextualisées.

3.6.2 Auto-attention

Étant donné une séquence de vecteurs de mots $X = (x_{m_1}, ..., x_{m_T})$, le mécanisme d'auto-attention se décompose en trois étapes [1] :

1. Calcul d'une **matrice d'attention** A
2. Calcul de **Valeurs** V
3. Calcul des **vecteurs contextualisées** en utilisant A et V

Matrice d'attention

Étant donné un vecteur $x_{m_i} \in \mathbb{R}^d$ de la séquence X en entrée, un score d'attention $\alpha_{i,j}$ est calculé entre x_{m_i} et chacun des vecteurs x_{m_j} de X. Intuitivement, ce score

1. les étapes 1. et 2. sont effectuées en parallèle

représente l'importance que le mot m_j aura dans le calcul du vecteur contextualisé du mot m_i. La matrice d'attention A contient l'ensemble des scores d'attention de la séquence X : $A_{i,j} = \alpha_{i,j}$. Plusieurs méthodes pour calculer la matrice d'attention A à partir de X ont été expérimentés (Luong et al., 2015). Ici, nous ne présenterons que celle utilisée, entre autres, par *Transformer* et *BERT* (Devlin et al., 2019) :

$$A = \text{Softmax}\left(\frac{XW^Q(XW^K)^T}{\sqrt{d_k}}\right) = \text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right), \tag{3.23}$$

avec $W^Q \in \mathbb{R}^{d \times d_k}$ et $W^K \in \mathbb{R}^{d \times d_k}$ des matrices dont les coefficients sont des paramètres du modèle. Le facteur d'échelle $\sqrt{d_k}$ a été fixée empiriquement et permet d'avoir des gradient stables lors de l'entraînement (Vaswani et al., 2017). Cette formule a été retenue par la plupart des modèles utilisant l'auto-attention car, en pratique, elle est rapide à exécuter et est compatible avec des algorithmes efficient de multiplication de matrices en parallèle avec des GPUs (Vaswani et al., 2017). Une illustration du calcul de la matrice d'auto-attention est représentée sur la Figure 3.8.

Valeurs

La séquence des valeurs $V \in \mathbb{R}^{d_k \times n}$ est le résultat d'une transformation de la séquence d'entrée X. Tout comme pour la matrice d'attention, la méthode la plus populaire pour calculer V est celle qui en pratique est la plus rapide et compatible avec des algorithmes efficient de multiplication de matrices en parallèle (Vaswani et al., 2017) :

$$V = XW^V, \tag{3.24}$$

avec $W^v \in \mathbb{R}^{d \times d_k}$ une matrice dont les coefficients sont des paramètres du modèle.

Vecteurs contextualisés

Finalement, chaque vecteur contextualisé z_{m_i} de la séquence Z en sortie du mécanisme d'auto-attention est calculé en effectuant une combinaison linéaire des éléments v_i de la séquence des valeurs V. Les coefficients des combinaisons linéaires sont issus de la matrice d'auto-attention A :

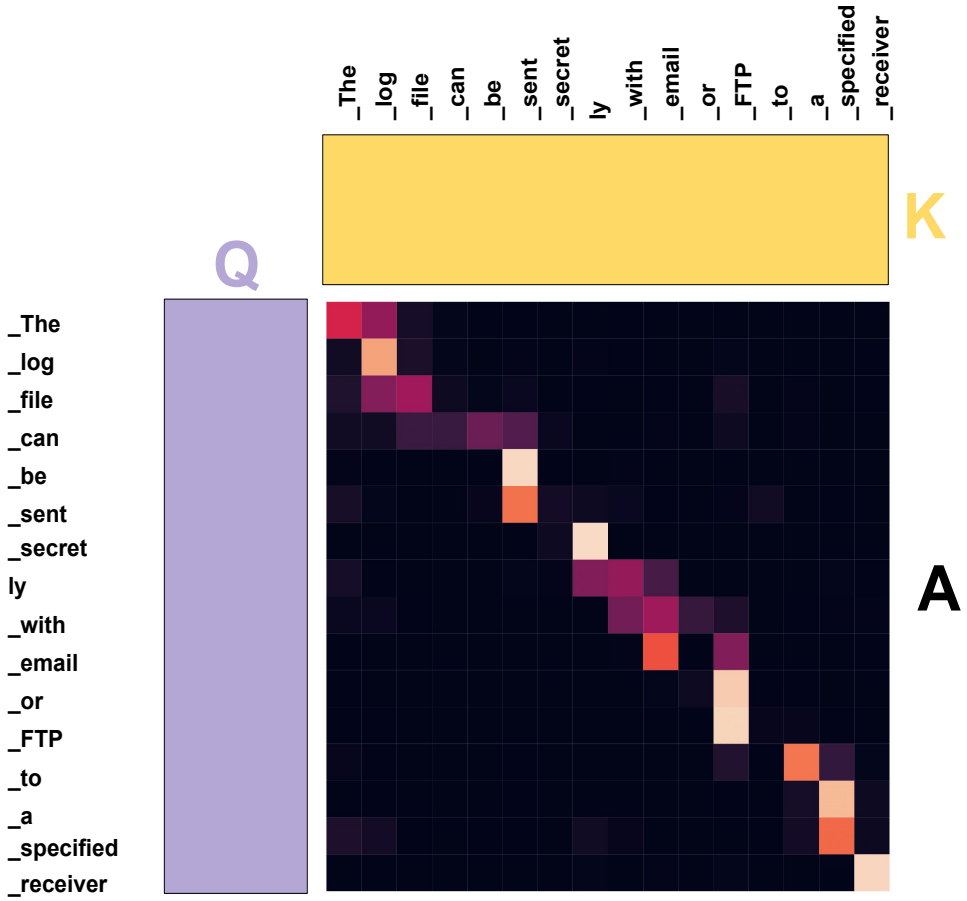

FIGURE 3.8 – Illustration du calcul de la matrice d'auto-attention. Pour plus de clarté, le calcul des matrices Q et K ainsi que la fonction softmax et le facteur d'échelle n'ont pas été représentés. Les valeurs représentés ont été calculés à l'aide du code disponible à cette adresse : https://nlp.seas.harvard.edu/2018/04/03/attention.html

$$
\begin{aligned}
Z =& \text{Attention}(W^Q, W^K, W^V, X) \\
=& \text{softmax}\left(\frac{XW^Q(XW^K)^T}{\sqrt{d_k}}\right) XW^V \\
=& \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V,
\end{aligned}
\tag{3.25}
$$

3.6.3 Auto-attention multi-têtes

En pratique, la majorité des modèles utilisent l'auto-attention multi-têtes au lieu d'une simple auto-attention. L'auto-attention multi-têtes est une généralisation de l'auto-attention consistant à appliquer h fois le mécanisme d'auto-attention à la séquence en utilisant h différentes matrices W_i^Q, W_i^K et W_i^V (avec $i \in [1..h]$). L'auto-attention multi-têtes apprend donc h matrices d'attentions (voir Figure 3.9) et retourne h séquences de vecteurs contextualisés [1] qui seront concaténés pour former une unique séquence :

$$
\begin{aligned}
Z = \text{MultiTêtes}(X) =& \text{Concat}\left(\text{tête}_1, ..., \text{tête}_h\right) W^O \\
\text{avec tête}_i =& \text{Attention}(W_i^Q, W_i^K, W_i^V, X),
\end{aligned}
\tag{3.26}
$$

avec $W_i^Q \in \mathbb{R}^{d \times d_k}$, $W_i^K \in \mathbb{R}^{d \times d_k}$, $W_i^V \in \mathbb{R}^{d \times d_k}$ et $W^O \in \mathbb{R}^{d_k h \times d}$ des matrices dont les coefficients sont des paramètres du modèle.

Selon les auteurs de l'article *Attention Is All You Need* (Vaswani et al., 2017) dans lequel est proposé le mécanisme d'auto-attention multi-têtes, ce dernier permet au modèle de porter son attention sur plusieurs éléments de la séquence en entrée, ce qu'un modèle utilisant une seule tête d'attention n'est pas capable de faire.

1. la simple auto-attention correspond au cas $h = 1$

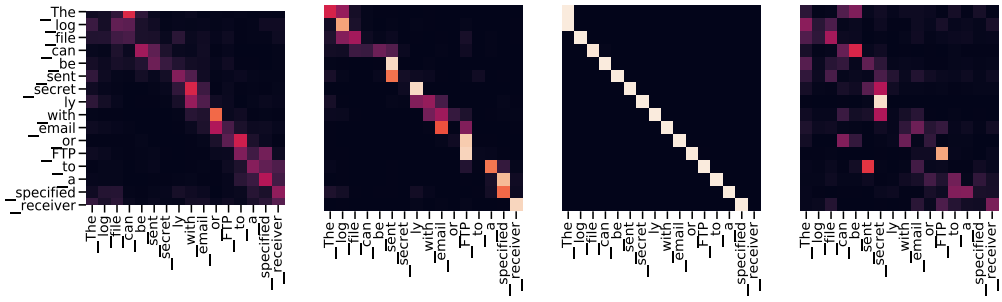

FIGURE 3.9 – Illustration des 4 matrices d'auto-attention produites par un encodeur Transformers à 4 têtes. Les valeurs représentés ont été calculés à l'aide du code disponible à cette adresse : https://nlp.seas.harvard.edu/2018/04/03/attention.html

3.6.4 Encodeur Transformers

L'architecture globale d'un encodeur *Transformers* est illustré sur la Figure 3.11.

Vecteurs de position

Les mécanismes d'auto-attention et d'auto-attention multi-têtes ne prennent pas en compte l'ordre des éléments de la séquence X en entrée. En effet, contrairement aux réseaux de neurones traditionnellement utilisés pour traiter des séquences (CNN, RNN) les opérations utilisées par l'encodeur *Transformers* ne prennent pas en compte l'ordre des élément de la séquence à encoder. Par exemple, permuter m_1 et m_2, aura pour seul effet la permutation de z_{m_1} et z_{m_2} dans la séquence en sortie et pas une modification des représentations z_{m_1} et z_{m_1}. Afin de prendre en compte la position des éléments de la séquence en entrée, des vecteurs de positions (notés PE) sont utilisés par l'encodeur *Transformers*. Ces vecteurs sont associés à la position absolue des éléments de la séquence en entrée et peuvent être soit appris (Gehring et al., 2017) soit fixés en utilisant les fonctions sinus et cosinus comme suit :

$$\begin{aligned} \mathrm{PE}_{(pos,2i)} &= \sin(pos/10000^{2i/d}) \\ \mathrm{PE}_{(pos,2i+1)} &= \cos(pos/10000^{2i/d}), \end{aligned} \tag{3.27}$$

avec pos la position et i la dimension. Cette formule a été proposée par Vaswani et al. (2017) et est utilisée car elle respecte 4 propriétés désirables [1] :

1. https://kazemnejad.com/blog/transformer_architecture_positional_encoding/

— Chaque vecteur est unique : deux positions différentes ne peuvent pas avoir des vecteurs identiques

— La distance entre deux vecteurs de positions successive doit être indépendante de la séquence considérée

— Le modèle doit être généralisable à des séquences de longueur arbitraire

— Le modèle doit être déterministe

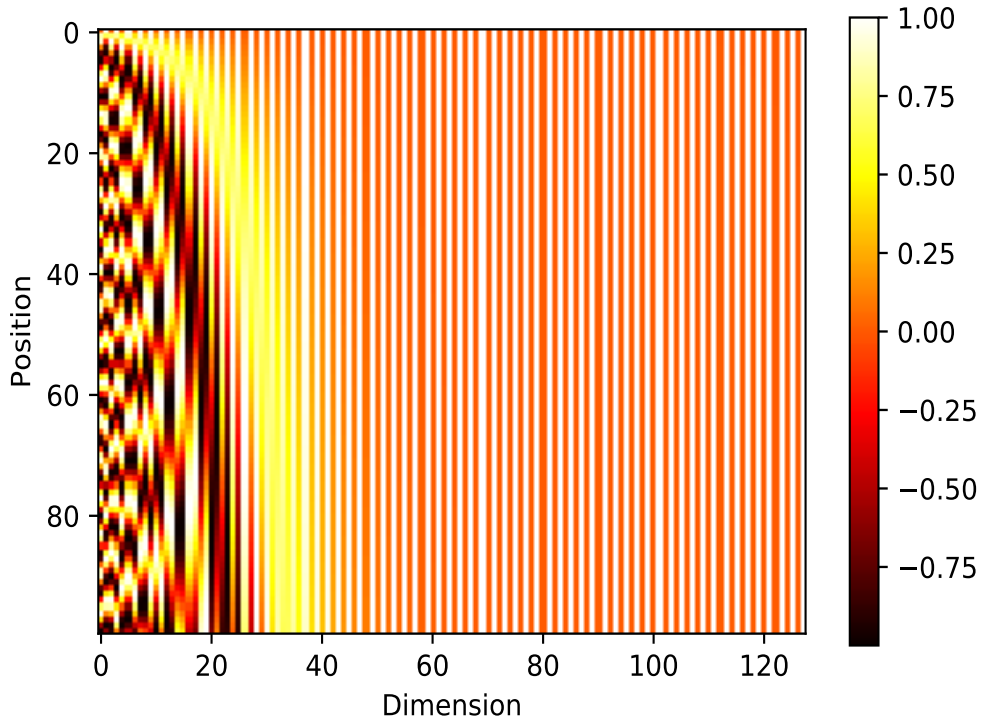

FIGURE 3.10 – *Heatmap* représentant des vecteurs de position de dimension 128 sur une séquence de longueur 100.

Les vecteurs de position dont les coefficients sont fixés sont généralement utilisés plutôt que des vecteurs appris, car ils généralisent mieux à des séquences plus longues que celles utilisées dans l'entraînement (Vaswani et al., 2017). Concrètement, les vecteurs de positions sont additionnés aux vecteurs de mots en entrée de l'encodeur comme illustré sur la Figure 3.11.

Connexions résiduelles, normalisation de couche et MLP

Le reste de l'architecture d'un encodeur *Transformers* (Voir Figure 3.11) est composée d'opérations que l'on retrouve dans la plupart des réseaux de neurones et que nous allons présenter brièvement :

— **Connexions résiduelles.** Il s'agit d'une méthode consistant à sommer les représentations d'une couche d'un réseau de neurones avec les représentations d'une couche précédente (He et al., 2016b). Cela permet de résoudre en partie les problèmes de disparition/explosion du gradient (*vanishing/exploding gradient problem*) et de manière plus générale, d'augmenter les performances du réseau.

— **Normalisation de couche.** Cette méthode normalise la sortie d'une couche d'un réseau de neurones en utilisant l'espérance et la variance des éléments en sortie de la couche (Ba et al., 2016). Cette méthode permet de diminuer le nombre d'étapes d'entraînement pour que le modèle converge et de stabiliser l'entraînement des réseaux de neurones récurrents.

— **MLP.** L'encodeur Transformers fait également appel à un MLP à une couche cachée avec une non-linéarité ReLU. Le MLP est appliqué à toutes les positions.

3.7 BERT

BERT (***B**idirectional **E**ncoder **R**epresentations from **T**ransformers* (Devlin et al., 2019)) est un modèle de langue neuronal utilisant des réseaux de neurones profonds et des encodeurs *Transformers*. Contrairement à *Word2Vec* et à *fast*Text qui associent un unique vecteur à un mot, BERT produit des vecteurs contextualisés dont les valeurs dépendent du contexte.

De la même façon que *fast*Text, BERT utilise des vecteurs de n-gramme de caractères plutôt que des vecteurs de mots afin de ne pas avoir le problème des mots hors-vocabulaire. L'architecture de BERT ainsi que le procédé de *tokenization* sont représentés sur la Figure 3.12. Les paramètres de BERT sont entraînés sur deux tâches : modèle de langue masqué et prédiction de phrase suivante. Le modèle de langue masqué consiste à masquer de façon aléatoire des mots des séquences d'entraînement et à entraîner le modèle à prédire les mots qui ont été masqués. La prédiction de phrase suivante consiste à présenter deux phrases au modèle qui doit décider si elles sont bien l'une à la suite de l'autre dans le corpus d'apprentissage.

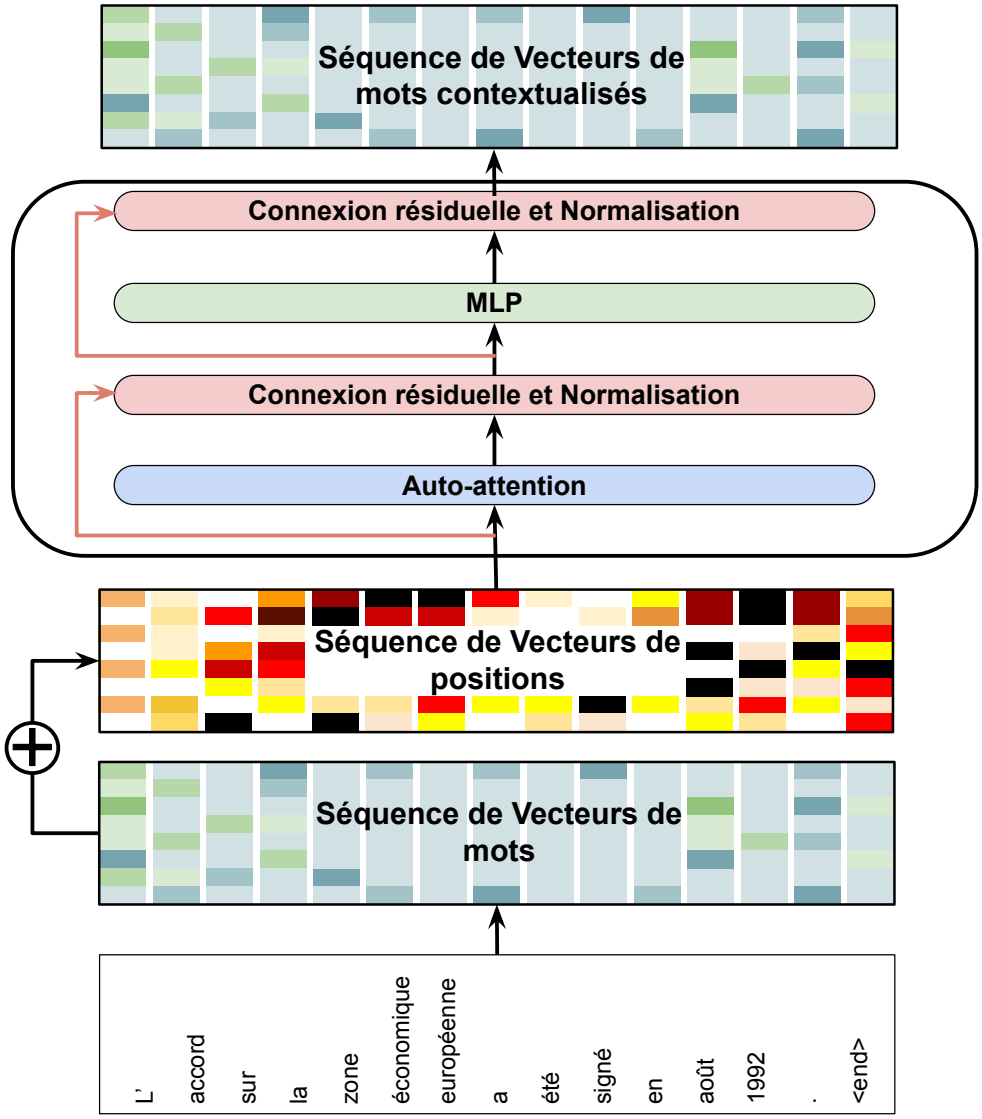

FIGURE 3.11 – Architecture d'un encodeur *Transformers* ayant une séquence de mots en entrée

Devlin et al. (2019) ont mis à disposition deux modèles pré-entraînés : $\text{BERT}_{\text{BASE}}$ et $\text{BERT}_{\text{LARGE}}$. $\text{BERT}_{\text{BASE}}$ est composé de 12 encodeurs Transformers de dimension 768 et à 12 têtes d'attentions et $\text{BERT}_{\text{LARGE}}$ est composé de 24 encodeurs Transformers de dimension 1024 et à 16 têtes d'attentions. Dans le cadre de cette thèse, nous n'avons utilisé que le modèle $\text{BERT}_{\text{BASE}}$ puisqu'il nécessite moins de puissance de calcul et moins de paramètres que $\text{BERT}_{\text{LARGE}}$.

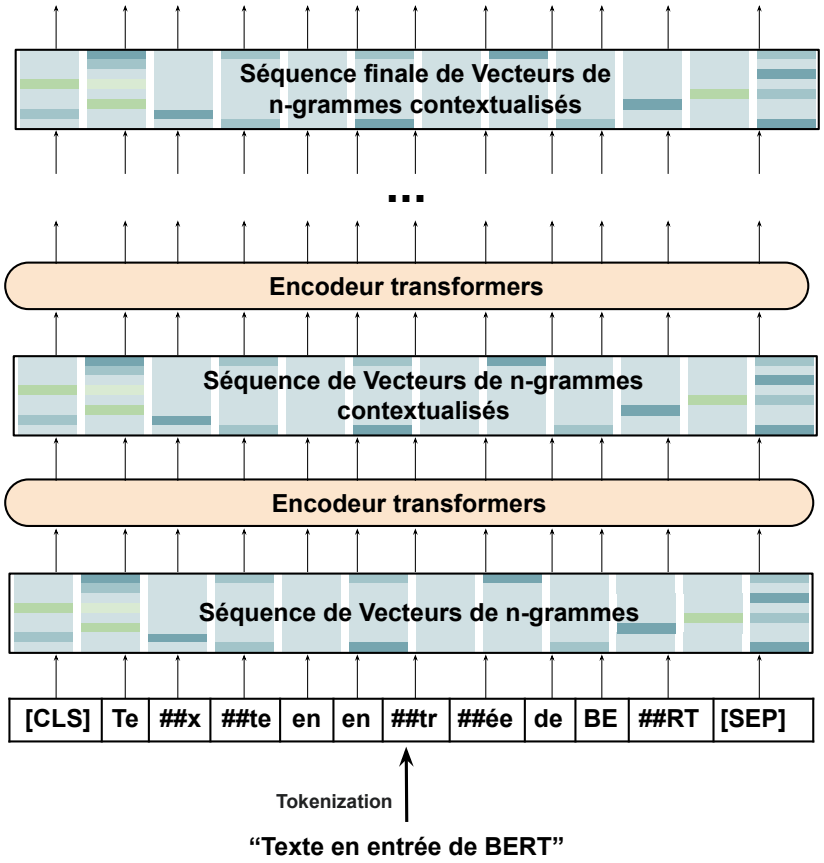

FIGURE 3.12 – Architecture du modèle BERT

En pratique, le principal avantage de BERT [1] par rapport à *Word2Vec* et *fast*Text vient du fait que les vecteurs de mots qu'il produit sont adaptables à plusieurs tâches du TALN [2] et que peu de données sont nécessaires pour entraîner des modèles états-

1. et plus généralement des modèles de langue neuronaux utilisant des encodeurs Transformers
2. classification de textes, paraphrase, inférence en langage naturel, analyse syntaxique, désambiguïsation automatique, ...

de-l'art (Brown et al., 2020).

BERT possède néanmoins un inconvénient majeur comparé à *fast*Text : à chaque nouvelle phrase/séquence, il faut utiliser l'ensemble du réseau de neurones pour calculer les vecteurs de n-grammes, ce qui correspond approximativement à 450 milliards d'opérations en virgule flottante pour une séquence de longueur 64. Tandis qu'associer des mots à des vecteurs *Word2Vec* ne nécessite que l'utilisation d'une table de correspondance.

3.8 Conclusion

Dans ce chapitre, nous avons décrit l'utilisation de réseaux de neurones pour le TALN. L'apprentissage des réseaux de neurones se fait à l'aide de méthodes basées sur la descente de gradient, ce qui implique que les opérations mathématiques utilisés doivent être différentiables et que leurs entrées doivent être composées de nombre réels. Dans le cas du TALN, des vecteurs de mots entraînés sur de grandes quantités de texte sont utilisés en entrée puis sont traités à l'aide de différents réseaux de neurones (CNN, LSTM, *Transformers*) dont les paramètres sont optimisés pour la tâche considérée. Dans le chapitre suivant, nous allons présenter l'utilisation des réseaux de neurones pour la tâche spécifique de recherche d'information textuelle.

Chapitre 4

Recherche d'information neuronale supervisée

4.1 Introduction

Dans ce qui suit, nous décrivons la formulation unifiée des modèles d'apprentissage d'ordonnancement neuronaux (NLTR, de l'anglais *neural learning-to-rank*) pour la RI proposée par Guo et al. (2019b).

Lors de la phase d'apprentissage, l'objectif d'un modèle NLTR pour la RI est de trouver les paramètres d'un réseau de neurones f qui minimisent une fonction objectif $\mathcal{L}$ sur les paires requête/document dont la pertinence est connue :

$$f^* = \arg\min_f \sum_i \sum_j \mathcal{L}\left(f, r^{(i)}, d^{(j)}, y_{i,j}\right), \tag{4.1}$$

avec $y_{i,j}$ le niveau de pertinence du document $d^{(j)}$ par rapport à la requête $r^{(i)}$. Les réseaux de neurones pour la RI peuvent être décrits de la façon suivante :

$$f(r,d) = g(\psi(r), \phi(d), \eta(r,d)), \tag{4.2}$$

ψ et ϕ sont des fonctions qui extraient des *features* à partir de r et d (nombre de termes, vecteurs de mots, ...). η extrait des *features* à partir de la paire (r,d) (score TF-IDF, score BM25, matrice de similarité entre les termes, ...). g est la fonction calculant le score de pertinence à partir des *features* (cosinus, rankSVM, LAMBDAMART, ...).

Dans la majorité des approches neuronales, ψ et ϕ sont des fonctions associant aux termes de r et d des vecteurs de mots pré-entraînés et g est la partie du réseau

de neurones calculant un score de similarité à partir des séquences de vecteurs de mots.

4.2 Architectures symétrique et asymétrique

Il est possible de séparer les architectures des modèles NLTR pour la RI en deux catégories : symétrique et asymétrique. Notons que les hypothèses sous-jacente de ces architectures ne sont pas spécifiques aux modèles neuronaux.

4.2.1 Architecture symétrique

L'hypothèse sous-jacente de ce type d'architecture est que les entrées r et d sont homogènes et sont par conséquent traitées de la même façon par le modèle. Cette hypothèse d'homogénéité peut sembler irréaliste pour la RI où les documents sont souvent longs et écrits en langage naturel alors que les requêtes sont courtes et composées de mots-clés. Elle peut néanmoins se révéler réaliste dans le cas où seul le titre du document est utilisé (Huang et al., 2013).

Le fait d'intervertir r avec d ne change pas la sortie d'un réseau de neurones ayant une architecture symétrique :

$$f_{\text{sym}}(r, d) = f_{\text{sym}}(d, r). \tag{4.3}$$

Dans le cas d'une architecture symétrique, les fonctions d'extractions de *features* ψ et ϕ sont identiques (les même features sont extraites de r et d) et η n'utilise que des *features* symétriques dont la valeur ne change pas si l'on intervertit r et d.

4.2.2 Architecture asymétrique

L'hypothèse sous-jacente de ce type d'architecture suppose que les entrées r et d sont hétérogènes et sont par conséquent traitées différemment par le modèle. Le fait d'intervertir r avec d changera la sortie d'un réseau de neurones ayant une architecture asymétrique :

$$f_{\text{asym}}(r, d) \neq f_{\text{asym}}(d, r). \tag{4.4}$$

Les architectures asymétriques sont préférées en RI en raison des différences de longueurs et de structures entre requêtes et documents.

4.3 Représentation et interaction

Il est également possible de séparer les architectures des modèles NLTR pour la RI en deux autres catégories : centré sur la représentation et centré sur l'interaction.

4.3.1 Architecture centrée sur la représentation

L'hypothèse sous-jacente de ce type d'architecture est que la pertinence dépend uniquement de la composition des textes en entrée. Par conséquent, les modèles de cette catégorie utilisent uniquement les fonctions d'extractions de *features* ψ et ϕ. Les architectures centrées sur la représentation produisent une représentation abstraite de r via ψ et une représentation abstraite de d via ϕ et utilisent une fonction de similarité (cosinus, produit scalaire, MLP, ...) pour calculer le score de pertinence. Ce type d'architecture est généralement plus adaptée pour les textes courts, car il est plus difficile de modéliser un long document à l'aide d'une représentation abstraite de taille fixe qu'un document court, ce qui peut poser problème en RI lorsque de longs documents sont à considérer.

Néanmoins, les modèles possédant ce type d'architecture permettent d'avoir des systèmes de RI efficients. En effet, les représentations abstraites des documents peuvent être calculées hors-ligne (*offline*). Par conséquent, lors de la phase de recherche, le modèle n'a besoin de calculer que la représentation de la requête et d'utiliser la fonction de similarité pour classer les documents.

4.3.2 Architecture centrée sur l'interaction

L'hypothèse sous-jacente de ce type d'architecture est que la pertinence dépend essentiellement des relations entre les termes de la requête et du document. Les modèles de cette catégorie utilisent uniquement la fonction d'extraction de *features* η et utilisent une fonction de similarité complexe (réseau de neurones) pour produire le score de pertinence. De plus, en utilisant des signaux d'interaction plutôt que des représentations abstraites, ce type de modèle est plus adapté à la RI puisqu'il peut prendre en compte les signaux de correspondance exacte entre termes (Guo et al., 2016). Les modèles basés sur l'interaction ont généralement moins de difficultés à modéliser de longs documents puisqu'ils ne sont pas restreints par des représentations de taille fixe.

Néanmoins, et contrairement aux modèles utilisant une architecture centrée sur la représentation, les modèles basés sur l'interaction ne permettent pas d'avoir un système de RI efficient. En effet, il n'est pas possible d'effectuer une partie significative des calculs hors-ligne puisque ce type de modèle a besoin de la paire requête/-

document avant de pouvoir calculer les *features* d'interaction. À cause de ce manque d'efficience, ces modèles sont souvent utilisés pour faire du re-classement : un premier classement est effectué à l'aide d'un modèle efficient (par exemple BM25) puis un second classement est effectué par le modèle non-efficient sur les documents les mieux classés par le modèle efficient (par exemple les 1000 documents les mieux classés).

4.4 Apprentissage

L'apprentissage des modèles NLTR se fait de la même façon que les autres approches neuronales : en définissant une fonction objectif à minimiser à l'aide d'une méthode basée sur la descente de gradient. Les fonctions objectifs des modèles NLTR peuvent être séparées en trois catégories (Li et al., 2008) : *pointwise*, *pairwise* et *listwise*.

4.4.1 Fonction objectif *pointwise*

L'approche *pointwise* consiste à considérer le problème de classement des documents comme un problème de classification ou de régression. Étant donné une paire requête/document $(r^{(i)}, d^{(j)})$ et son niveau de pertinence associé $y_{i,j}$ une fonction objectif *pointwise* mesurera la différence entre le niveau de pertinence $y_{i,j}$ et la sortie du réseau de neurones $f(r^{(i)}, d^{(j)})$. L'objectif global du modèle peut être formulé de la façon suivante :

$$\mathcal{L}_{point} = \sum_i \sum_j \mathcal{L}\left(y_{i,j}, f(r^{(i)}, d^{(j)})\right). \tag{4.5}$$

Lorsque le problème de classement est considéré comme un problème de classification, la fonction objectif *pointwise* la plus populaire est l'entropie croisée :

$$\mathcal{L}_{point\text{-}cr} = -\sum_i \sum_j y_{i,j} \log\left(f(r^{(i)}, d^{(j)})\right) + (1 - y_{i,j}) \log\left(1 - f(r^{(i)}, d^{(j)})\right). \tag{4.6}$$

Dans l'exemple ci-dessus, la sortie de la fonction de classement f doit être comprise entre 0 et 1. Il est également supposé qu'il n'y a que deux classes : pertinent ($y_{i,j} = 1$) et non-pertinent ($y_{i,j} = 0$), mais il est possible d'utiliser l'entropie croisée pour des problèmes à plus de deux classes ($C > 2$) :

$$\mathcal{L}_{point\text{-}cr} = -\sum_i \sum_j \sum_{c=1}^{C} \mathbb{P}\left(y_{i,j} = c\right) \log \mathbb{P}\left(f(r^{(i)}, d^{(j)}) = c\right). \tag{4.7}$$

Lorsque le problème de classement est considéré comme un problème de régression, il est possible d'utiliser l'erreur quadratique moyenne en tant que fonction objectif *pointwise* :

$$\mathcal{L}_{reg} = \sum_i \sum_j \left(y_{i,j} - f(r^{(i)}, d^{(j)})\right)^2. \tag{4.8}$$

Dans l'exemple ci-dessus, il est également supposé que les niveaux de pertinences $y_{i,j}$ prennent des valeurs numériques.

Les fonctions objectifs *pointwise* ont deux avantages : **(1)** le nombre total d'évaluation à effectuer évolue de façon linéaire avec le nombre total de jugements de pertinence puisqu'il suffit d'évaluer les paires de requêtes et de documents indépendamment les unes des autres. **(2)** les valeurs retournées par le modèle sont comparables aux jugements de pertinence et sont donc interprétables.

Néanmoins, en pratique les fonctions objectifs *pointwise* sont considérées comme peu performantes pour apprendre à classer puisqu'elles ne prennent pas en compte les informations relatives à l'ordre, elles ne garantissent pas un classement optimal, même lorsque le modèle parvient à un minimum global (Guo et al., 2019b).

4.4.2 Fonction objectif *pairwise*

L'approche *pairwise* consiste à mesurer le classement relatifs de deux documents par rapport à une requête. Étant donné une requête $r^{(i)}$ et deux documents $d^{(j)}$ et $d^{(k)}$ tel que $d^{(j)}$ soit plus pertinent que $d^{(k)}$ par rapport à $r^{(i)}$: $y_{i,j} \succ y_{i,k}$, une fonction objectif *pairwise* pénalisera les fonction de classement ne donnant pas à $d^{(j)}$ un score plus élevé qu'à $d^{(k)}$. L'objectif global de l'approche *pairwise* peut être formulé de la façon suivante :

$$\mathcal{L}_{pair} = \sum_i \sum_{\substack{(j,k) \\ \text{tels que} \\ y_{i,j} \succ y_{i,k}}} \mathcal{L}\left(f(r^{(i)}, d^{(j)}) - f(r^{(i)}, d^{(k)})\right). \tag{4.9}$$

Une fonction objectif *pairwise* populaire est la fonction de Hinge pour le classement :

$$\mathcal{L}_{\text{Hinge}} = \sum_i \sum_{\substack{(j,k) \\ \text{tels que} \\ y_{i,j} \succ y_{i,k}}} \max\left(0, m - f(r^{(i)}, d^{(j)}) + f(r^{(i)}, d^{(k)})\right), \tag{4.10}$$

avec $m > 0$ la marge. Cette fonction de perte est minimale (égale à 0) si pour toutes les paires de documents, le score du document le plus pertinent est supérieur au score du document le moins pertinent avec un différence au moins égale à m. Il est également possible de modifier l'entropie croisée en une fonction objectif de type *pairwise* (Burges et al., 2005) :

$$\begin{aligned}
\mathcal{L}_{pair\text{-}cr} &= -\sum_i \sum_{\substack{(j,k) \\ \text{tels que} \\ y_{i,j} \succ y_{i,k}}} \underbrace{\mathbb{P}\left(y_{i,j} \succ y_{i,k}\right)}_{=1} \log \mathbb{P}\left(d_j \succ d_k\right) \\
&= -\sum_i \sum_{\substack{(j,k) \\ \text{tels que} \\ y_{i,j} \succ y_{i,k}}} \log \mathbb{P}\left(d_j \succ d_k\right) \\
&= -\sum_i \sum_{\substack{(j,k) \\ \text{tels que} \\ y_{i,j} \succ y_{i,k}}} \log \sigma\left(f(r^{(i)}, d^{(j)}) - f(r^{(i)}, d^{(k)})\right)
\end{aligned} \tag{4.11}$$

avec $\sigma(x) = 1/(1 + \exp(-x))$ la fonction sigmoïde. Cette dernière est utilisée pour estimer la probabilité que $d^{(j)}$ soit plus pertinent que $d^{(k)}$: $\mathbb{P}\left(d^{(j)} \succ d^{(k)}\right) = \sigma\left(f(r^{(i)}, d^{(j)}) - f(r^{(i)}, d^{(k)})\right)$. Précisons également que $\mathbb{P}\left(y_{i,j} \succ y_{i,k}\right) = 1$ dans l'équation (4.11) puisque seules sont considérées les paires (j,k) tels que $y_{i,j} \succ y_{i,k}$.

Contrairement aux fonctions objectifs *pointwise*, les fonctions objectifs *pairwise* prennent en compte certaines informations relatives à l'ordre des documents. De plus, lorsque l'objectif global d'une fonction *pairwise* est minimisé et que toutes les relations entre documents sont prises en compte, le classement final sera bien optimal. Néanmoins, en pratique, il est difficile de prendre en compte toutes les relation entre documents pour chacune des requêtes puisque leur nombre évolue de façon quadratique avec le nombre total de jugements de pertinence pour chacune des requêtes (Chen et al., 2009). Une autre limite des approches *pairwise* vient du fait que le problème de classement n'est pas modélisé de manière directe, mais par l'intermédiaire de paires de documents considérés de façon identiques (Xia et al., 2008). Cette façon de modéliser est problématique puisque la majorité des métriques de RI accordent plus d'importance aux documents les mieux classés. Par conséquent deux modèles peuvent être considérés comme ayant des classements de documents

similaires selon une fonction objectif *pairwise* tout en ayant des classements de documents significativement différents (Guo et al., 2019b).

4.4.3 Fonction objectif *listwise*

L'approche *listwise* consiste à évaluer le classement d'un ensemble de documents par rapport à une requête. L'objectif global de l'approche *listwise* peut être formulé de la façon suivante :

$$\mathcal{L}_{list} = \sum_i \mathcal{L}\left\{y_{i,j}, f(r^{(i)}, d^{(j)}) | d^{(j)} \in D_i\right\}, \tag{4.12}$$

avec D_i l'ensemble des documents dont la pertinence par rapport à la requête $r^{(i)}$ est connue. Étant donné une requête r_i, on note π_i la liste des documents de D_i classés par ordre décroissant de pertinence. Xia et al. (2008) ont proposés *ListMLE* comme fonction objectif *listwise* :

$$\mathcal{L}_{listMLE} = -\sum_i \sum_{j=1}^{|\pi_i|} \log \mathbb{P}\left(y_{i,j} | D_i^{(j)}, f\right), \tag{4.13}$$

avec $\mathbb{P}\left(y_{i,j} | D_i^{(j)}, f\right)$ la probabilité que le $j^{\text{ème}}$ document de la liste optimale soit classé correctement par f :

$$\mathbb{P}\left(y_{i,j} | D_i^{(j)}, f\right) = \frac{\exp\left(f(r^{(i)}, d^{(j)})\right)}{\sum_{k=j}^{|\pi_i|} \exp\left(f(r^{(i)}, d^{(k)})\right)}. \tag{4.14}$$

Maximiser $\mathbb{P}\left(y_{i,j} | D_i^{(j)}, f\right)$ à l'aide de la fonction Softmax permet de faire en sorte que le score de $d^{(j)}$ soit supérieur aux scores de tous les documents moins bien classés dans la liste optimale π_i (il faut remarquer que la somme dans le dénominateur commence à j). Intuitivement, l'utilisation de la fonction objectif *ListMLE* est équivalent à minimiser la fonction log-vraisemblance négative (*negative log likelihood*) du classement optimal sachant la fonction de classement f.

Les approches *listwise* permettent généralement de produire de meilleurs résultats que les approches *pairwise* et *pointwise* puisqu'elles optimisent directement le classement des documents. La principale limitation des approches *listwise* réside dans leur complexité algorithmique.

En pratique, les approches *pairwise* et *listwise* sont les plus populaires pour optimiser les modèles NLTR pour la RI puisque, à la condition qu'elles soient utilisées

correctement, ce sont les approches produisant les meilleurs résultats (Qin et al., 2010).

4.5 Modèles

Tous les modèles NLTR présentés dans cette section calculent un score de pertinence entre une requête et un document. De plus, afin de faciliter la comparaison entre modèles, le nombre de paramètres indiqué ne prend pas en compte les vecteurs de mots. Nous reportons le nombre de paramètres de l'implémentation des modèles par *MatchZoo* (Guo et al., 2019a) avec des requêtes de longueur 10 et des documents de longueur 200. *MatchZoo* étant une bibliothèque *Python* implémentant des modèles d'apprentissage profond pour des tâches de comparaison de texte, puisque que nous utilisons cette implémentation des modèles lors de nos expériences.

4.5.1 ARC-I

ARC-I (*Architecture-I*) est un modèle symétrique centré sur la représentation dont l'architecture est représentée sur la Figure 4.1 (Hu et al., 2014). Le modèle prend en entrée les séquences de vecteurs de mots associées aux termes de la requête et du document. Puisque ARC-I est symétrique, les fonctions d'extraction de features ψ et ϕ sont identiques. Leur conception est inspirée de modèles de traitement d'images (LeCun et Bengio, 1998) : elles consistent en une succession de couches de convolution 1D avec une fonction d'activation ReLU et un *max-pooling*. Les deux représentations ainsi obtenues sont ensuite concaténées et traitées par un MLP qui retournera le score de pertinence. Les tailles variables des entrées sont gérées en utilisant du *zero-padding*. ARC-I est entraîné à l'aide de la fonction objectif *pairwise* de Hinge décrite à l'équation (4.10) avec une marge m égale à 1. Le nombre total de paramètres du modèle ARC-I est de l'ordre du demi-million (576 459).

4.5.2 ARC-II

ARC-II (*Architecture-II*) est un modèle symétrique centré sur l'interaction dont l'architecture est représentée sur la Figure 4.2 (Hu et al., 2014). Tout comme ARC-I, le modèle ARC-II prend en entrée les séquences de vecteur de mots associées aux termes des deux textes en entrée. Un ensemble de F matrices d'interactions I^f ($f \in [1, F]$) entre les deux séquences en entrée est ensuite créé en utilisant des

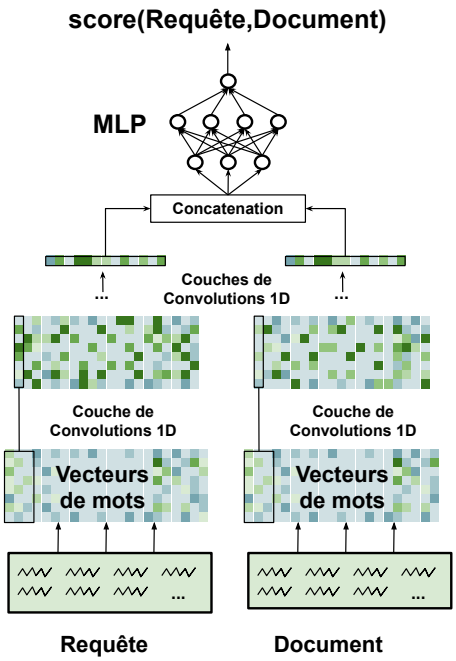

FIGURE 4.1 – Architecture du modèle ARC-I.

convolutions 1D calculant des scores d'interaction entre les n-grammes de termes de la requête et les n-grammes de termes du document :

$$I_{i,j}^{f} = \text{ReLU}\left(W^{f} Z_{i,j} + b^{f}\right), \tag{4.15}$$

avec b^f et W^f le biais et la matrice associés au f^{ème} filtre de convolution et $Z_{i,j}$ est la concaténation des vecteurs associés aux i^{ème} n-gramme de la requête et au j^{ème} n-gramme du document (voir Figure 4.2) . Par exemple, si l'on considère des tri-grammes :

$$Z_{i,j} = \left[x_{r_i}, x_{r_{i+1}}, x_{r_{i+2}}, x_{d_j}, x_{d_{j+1}}, x_{d_{j+2}}\right], \tag{4.16}$$

avec x_{r_i} le vecteur de mot du i^{ème} terme de la requête et x_{d_j} le vecteur de mot du j^{ème} terme du document.

Les matrices ainsi obtenues sont ensuite traitées de la même façon qu'une image le serait dans un CNN : par une succession de convolutions 2D, ReLU et *max-pooling* jusqu'à un MLP qui retournera le score de pertinence final. Les tailles variables des entrées sont gérées en utilisant du *zero-padding*. ARC-II est entraîné à l'aide de la fonction objectif *pairwise* de Hinge décrite à l'équation (4.10) avec

une marge m égale à 1. ARC-II contient approximativement 110 000 paramètres (113 793).

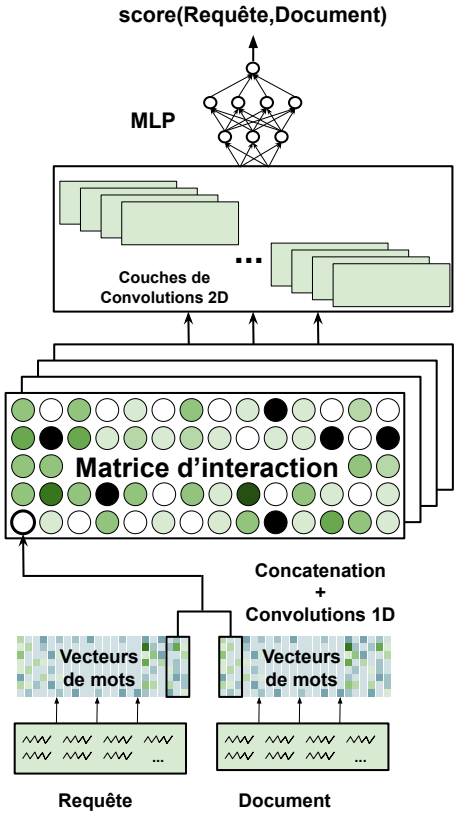

FIGURE 4.2 – Architecture du modèle ARC-II.

4.5.3 MatchPyramid

MatchPyramid (Pang et al., 2016b) est un modèle symétrique centré sur l'interaction dont l'architecture est représentée sur la Figure 4.3. MatchPyramid est similaire à ARC-II : une matrice d'interaction est calculée puis est traitée par un CNN comme le serait une image. La principale différence entre ces deux modèles réside dans le calcul de la matrice d'interaction I : MatchPyramid effectue un produit scalaire entre les vecteurs de mots associés aux termes de la requête avec les vecteurs de mots associés aux termes du document :

$$I_{i,j} = x_{r_i} \cdot x_{d_j},$$

L'avantage de cette approche vient du fait que si les vecteurs sont normalisés, le modèle est capable d'identifier et de prendre en compte les correspondances exactes puisque deux vecteurs égaux et normalisés ont un produit scalaire égale à 1. Les tailles variables des entrées sont gérées en utilisant du *zero-padding*. MatchPyramid est entraîné à l'aide d'une fonction objectif *pointwise* : l'entropie croisée décrite à l'équation (4.6). MatchPyramid contient 5 761 paramètres.

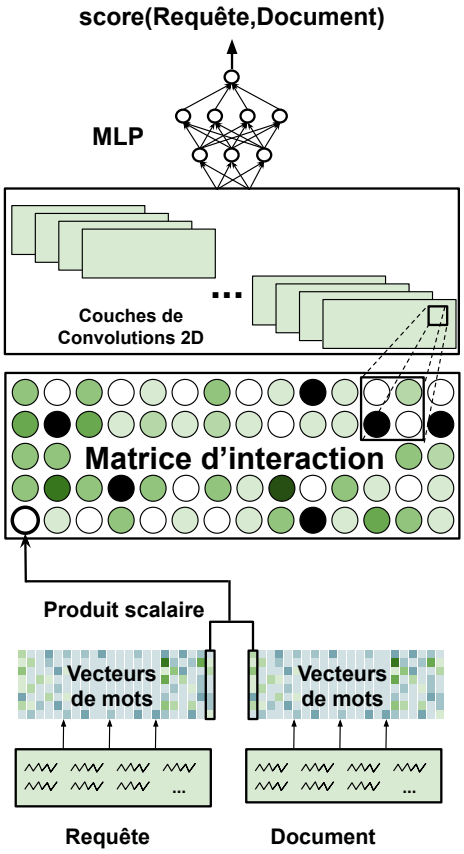

FIGURE 4.3 – Architecture du modèle MatchPyramid.

4.5.4 DRMM

DRMM (*Deep Relevance Matching Model*) est un modèle asymétrique centré sur l'interaction dont l'architecture est représentée sur la Figure 4.4 (Guo et al., 2016). Étant donné un terme de la requête, DRMM calcule la similarité cosinus

entre les vecteurs de mots de tous les termes du document et le vecteur de mots du terme de la requête considérée. Cet ensemble de similarités est ensuite résumé en un histogramme de correspondances. Afin de prendre en compte de façon explicite, les correspondances entre termes, une classe de l'histogramme est consacrée aux similarités cosinus exactement égales à 1. Les autres classes de l'histogramme correspondent à des intervalles de valeurs régulières entre -1 et 1.

Par exemple, si l'on considère un histogramme à 5 classes, elles seront associées aux intervalles de valeurs suivantes : $\{[-1,-0.5[;[-0.5,0[;[0,0.5[;[0.5,1[;[1,1]\}$. Étant donné la requête "*car*" et le document ("car", "rent", "truck", "bump", "injunction", "runway"), l'ensemble des similarités cosinus obtenu est le suivant : (1, 0.2, 0.7, 0.3, -0.1, 0.1). Cet ensemble est ensuite résumé en l'histogramme de correspondance suivant : [0,1,3,1,1] (exemple tiré de l'article original de Guo et al. (2016)).

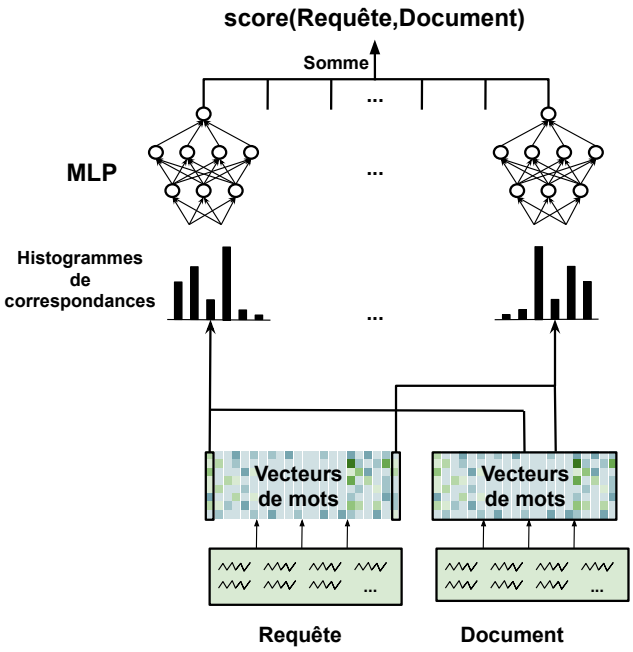

FIGURE 4.4 – Architecture du modèle DRMM.

Cet histogramme de correspondances est traité par un MLP qui retourne un score reflétant la similarité du terme de la requête avec l'ensemble des termes du document. Ce procédé est ainsi répété pour tous les termes de la requête, on obtient donc un score de pertinence associé à chacun des termes de la requête. Le score final de pertinence est calculé en additionnant les scores de pertinence de chacun des

termes de la requête. Puisque chacun des documents est associé à un histogramme de taille fixe indépendamment de leur nombre de termes, DRMM n'a pas besoin de *zero-padding* pour prendre en compte la taille variable des documents. DRMM est entraîné à l'aide de la fonction objectif *pairwise* de Hinge décrite à l'équation (4.10) avec une marge m égale à 1. DRMM contient 455 paramètres, par conséquent, il s'agit d'un des rares modèles NLTR n'ayant besoin que de peu de données d'entraînement (quelques dizaines de requêtes) pour produire un classement de même qualité que BM25.

4.5.5 DUET

DUET (Mitra et al., 2017) est un modèle asymétrique dont l'architecture est représentée sur la Figure 4.5. DUET est un modèle hybride utilisant à la fois une matrice d'interaction et des représentations abstraites de requêtes et de documents. DUET peut être séparé en deux sous-modèles : un modèle local (à gauche sur la Figure 4.5) et un modèle distribué (à droite sur la Figure 4.5).

Le modèle local construit une matrice d'interaction binaire I indiquant de façon explicite les termes en commun entre la requête et le document :

$$I_{i,j} = \begin{cases} 1, & \text{si } r_i = d_j \\ 0, & \text{sinon} \end{cases},$$

avec r_i le i^{ème} terme de la requête r et d_j le j^{ème} terme du document d. La matrice d'interaction est ensuite traitée par une couche de convolution suivie d'un MLP qui retourne un score de similarité local entre la requête et le document. Bien que le modèle local prend en compte de façon explicite la correspondance exacte entre termes de la requête et du document, il ne prend pas en compte les interactions possibles entres des termes différents.

Le modèle distribué prend en entrée les séquences de vecteurs de mots associés aux termes de la requête et du document. Des convolutions et un MLP sont appliqués de façon asymétrique à ces deux séquences afin de leur associer des représentations abstraites. Ces dernières sont ensuite combinées avec un produit terme à terme (produit de Hadamard) suivi de convolutions et d'un MLP qui retourne un score de similarité distribué entre la requête et le document. Les scores de similarité local et distribué sont enfin additionnés afin d'obtenir le score de pertinence final entre la requête et le document.

Les tailles variables des entrées sont gérées en utilisant du *zero-padding*. DUET est entraîné à l'aide de la fonction objectif *listwise ListMLE* (équation 4.13). Afin

de limiter le nombre d'opérations à effectuer lors de l'estimation de $\mathbb{P}\left(y_{i,j}|D_i^{(j)}, f\right)$, seuls 5 documents sont considérés plutôt que la liste entière. DUET contient approximativement 415 000 paramètres (415 780).

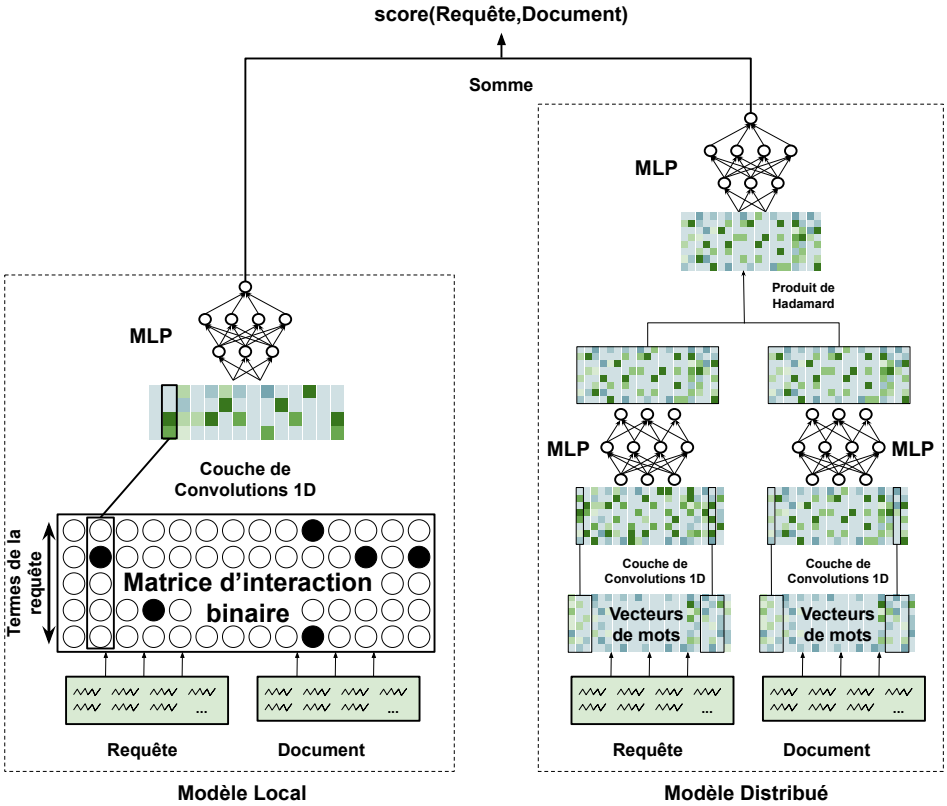

FIGURE 4.5 – Architecture du modèle DUET.

4.5.6 KNRM

KNRM (*Kernel-based Neural Ranking Model*) est un modèle asymétrique centré sur l'interaction dont l'architecture est représentée sur la Figure 4.6 (Xiong et al., 2017). KNRM construit une matrice d'interaction I en utilisant la similarité cosinus entre les vecteurs de mots associés aux termes de la requête avec les vecteurs de mots associés aux termes du document :

$$I_{i,j} = \cos\left(x_{r_i}, x_{d_j}\right) \tag{4.17}$$

KNRM utilise par la suite un ensemble de K noyaux RBF (*radial basis function*) afin d'associer à chacune des lignes de I (correspondant aux termes de la requête) un vecteur de caractéristiques de taille K :

$$K_k(I_i) = \sum_j \exp\left(\frac{(I_{i,j} - \mu_k)^2}{2\sigma_k^2}\right), \tag{4.18}$$

avec K_k la valeur de la k^{ème} dimension de $\overrightarrow{K}(I_i)$ le vecteur de caractéristiques associé au terme q_i et au document d. μ_k et σ_k sont les deux hyperparamètres caractérisant le k^{ème} kernel de KNRM. Le noyau RBF calcule comment les similarités cosinus entre un terme de la requête et l'ensemble des termes du document sont distribuées : plus il y a de similarité cosinus proche de μ_k, plus la valeur du noyau sera élevée. Par exemple, si $\mu_k = 1$, plus il y a de similarité cosinus égales à 1 (donc plus le document contient le terme de la requête considéré), plus la valeur de noyau sera élevée.

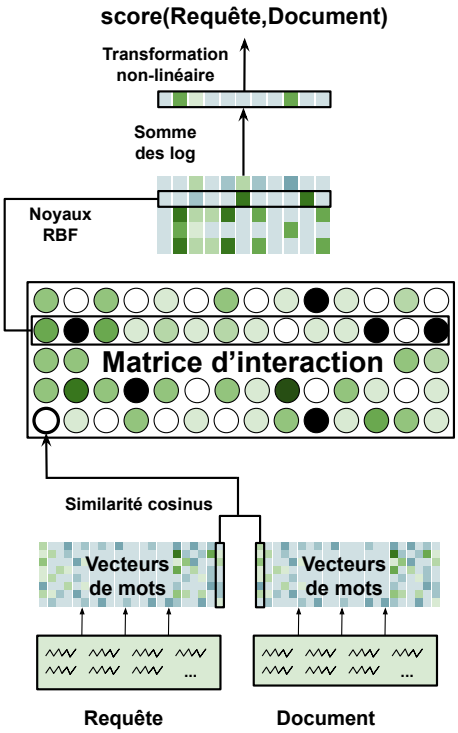

FIGURE 4.6 – Architecture du modèle KNRM.

Le vecteur de caractéristiques $\overrightarrow{K}(I)$ associé à la paire (q, d) est obtenu en addi-

tionnant le log des vecteurs de caractéristiques de chaque terme de la requête :

$$\overrightarrow{K}(I) = \sum_i \log \overrightarrow{K}(I_i). \tag{4.19}$$

Le vecteur de caractéristiques ainsi obtenu est traité par une couche finale composée d'une transformation linéaire et d'une non-linéarité :

$$f(q,d) = \tanh\left(w^T \cdot \overrightarrow{K}(I) + b\right), \tag{4.20}$$

avec w et b des paramètres de KNRM.

Puisque la taille du vecteur des caractéristiques $K(I)$ est indépendante de la longueur de la requête et du document en entrée, le modèle n'a pas besoin de *zero-padding*. KNRM est entraîné à l'aide de la fonction objectif *pairwise* de Hinge décrite à l'équation (4.10) avec une marge m égale à 1. KNRM contient très peu de paramètres : $w \in \mathbb{R}^K$ et $b \in \mathbb{R}$. Sachant que le modèle initialement proposé par Xiong et al. (2017) était composé de 11 kernels, KNRM contient uniquement 12 paramètres (w de dimension 11 et le biais b).

4.5.7 CKNRM

CKNRM (Convolutional Kernel-based Neural Ranking Model) est une extension du modèle KNRM (Dai et al., 2018). CKNRM est un modèle asymétrique centré sur l'interaction dont l'architecture est représentée sur la Figure 4.7. Étant donné deux séquences de vecteurs de mots associées respectivement à la requête et au document, CKNRM utilise plusieurs couches de convolutions pour associer aux n-grammes de termes un ensemble de vecteurs. Les vecteurs de n-grammes associés à la requête sont ensuite comparés aux vecteurs de n-grammes associés au document avec la similarité cosinus pour former un ensemble de matrices d'interactions.

Chacune des matrices d'interaction est ensuite traitée de façon analogue à KNRM : utilisation d'un ensemble de kernels pour associer à chaque matrice d'interactions un vecteur de caractéristiques. Les vecteurs de caractéristiques sont ensuite agrégés en une unique représentation qui est traitée par une couche finale composée d'une transformation linéaire et d'une non-linéarité et retournant le score de pertinence final.

Les tailles des vecteurs des caractéristiques $K(I)$ étant indépendantes de la longueur de la requête et du document en entrée, le modèle n'a pas besoin de *zero-padding*. CKNRM est entraîné à l'aide d'une fonction objectif *pairwise* : la fonction de Hinge pour le classement décrite à l'équation (4.10) avec une marge m égale à 1. CKNRM contient approximativement 230 000 paramètres (230 884).

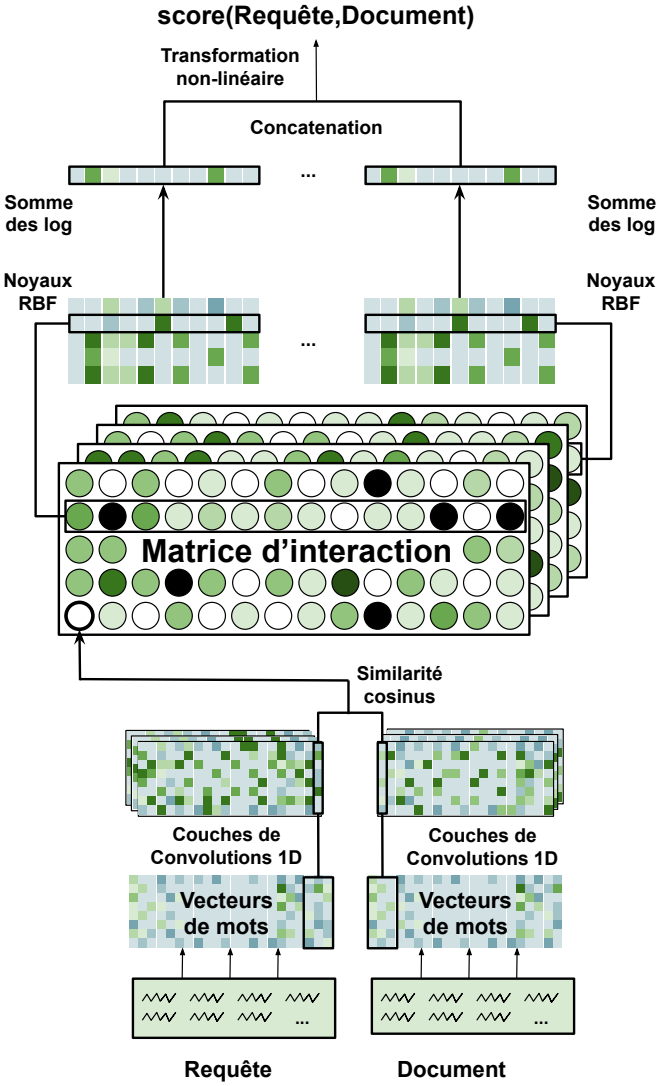

FIGURE 4.7 – Architecture du modèle CKNRM.

Modèle	Symétrique	Représentation	Interaction	Fonction Objectif	Nombre de Paramètres
KNRM	✗	✗	✓	*Pairwise* (Hinge)	12
DRMM	✗	✗	✓	*Pairwise* (Hinge)	455
MatchPyramid	✓	✗	✓	*Pointwise* (Entropie Croisée)	5.7k
ARC-II	✓	✗	✓	*Pairwise* (Hinge)	113k
CKNRM	✗	✗	✓	*Pairwise* (Hinge)	230k
DUET	✗	✓	✓	*Listwise* (*ListeMLE*)	415k
ARC-I	✓	✓	✗	*Pairwise* (Hinge)	576k

TABLE 4.1 – Tableau récapitulatif des modèles NLTR de référence. Les modèles sont classés par ordre croissant de leur nombre de paramètres.

4.6 BERT et modèles *Transformers* pré-entraînés pour la RI

Afin d'exploiter correctement les vecteurs de mots pré-entraînés à l'aide de l'architecture *Transformers* pour la RI, il ne suffit pas de donner ces vecteurs en entrée à un réseau de neurones pour la RI. Certaines pratiques sont à adapter afin de surpasser les modèles neuronaux "*pré-Transformers*" présentés dans la section précédente. Dans cette section, nous résumons les pratiques présentées par Lin et al. (2020). L'utilité de ces pratiques n'ayant fait consensus que récemment, nous n'avons pas pu les expérimenter dans ce travail de thèse.

4.6.1 Gérer la longueur des documents

Afin de pouvoir analyser de longs documents pouvant contenir des milliers de mots, BERT peut être utilisé pour calculer un score de pertinence entre la requête et chacune des phrases du document. Puisque la complexité des *Transformers* est quadratique par rapport à la longueur de la séquence en entrée, il est plus efficient d'analyser chaque phrase du document séparément plutôt que d'analyser directement le document. Les scores de pertinence de chacune des phrases sont ensuite agrégés pour obtenir un score de pertinence entre la requête et le document. Cette méthode s'est révélée empiriquement efficace (Yilmaz et al., 2019), et ce malgré le fait que le modèle n'a pas de vision "globale" du document.

4.6.2 Collection d'affinage

Qiao et al. (2019) ont montré que les vecteurs BERT, si ils sont utilisés directement pour la RI, produisent de moins bon résultats que les modèles basés sur la correspondance exacte. Il est nécessaire de les affiner sur la collection de RI considérée. Or, comme nous l'avons vu précédemment, la majorité des collections de RI *ad-hoc* pour le classement de document n'ont pas assez de données d'apprentissage pour suffisamment entraîner les vecteurs BERT. Comme l'ont montré il est possible de palier à ce problème, en affinant les vecteurs BERT non pas sur une collection de RI pour le classement de documents mais sur une tâche similaire telle que le classement de passage ou la tâche de question/réponses. Les résultats des expériences de Yilmaz et al. (2019) nous indique qu'il n'est même pas nécessaire d'affiner BERT sur la collection de test pour surpasser l'état de l'art. En effet, un modèle BERT affiné sur la collection MS MARCO (tâche de classement de passages) et sur la collection TREC Microblog Lin et al. (2014) surpasse les performances

du modèle BM25-RM sur la collection Robust04 sans avoir besoin d'être affiné sur cette collection.

4.7 Limites des réseaux de neurones pour la RI

Malgré leur efficacité, les méthodes *neural learning-to-rank* (NLTR) pour la RI présentent plusieurs limites.

4.7.1 Quantité de données étiquetées

Les méthodes NLTR pour la RI nécessitent de grandes quantités de requêtes résolues [1] afin d'obtenir des gains significatifs de performances par rapport aux modèles de référence basés sur la correspondance exacte Goodfellow et al. (2016).

4.7.2 Ressources sémantiques

En plus de nécessiter de grandes quantités de données annotées, les méthodes NLTR pour la RI ne modélisent la langue qu'à l'aide de méthodes analysant de grandes quantités de texte (Word2vec, *fast*Text, BERT ...). De façon plus spécifique, les méthodes NLTR ne modélisent pas la langue à l'aide de ressources sémantiques contenant des formulation explicites des significations des mots et de leurs relations dont les modèles NLTR pourraient tirer parti.

4.7.3 Incompatibilité avec l'index inversé

Finalement, une des caractéristiques cruciales qu'un système de recherche d'information doit avoir est d'être capable de répondre à une requête de façon efficiente (Arapakis et al., 2014). Or, en plus de nécessiter de faire des milliards d'opérations afin de prendre une unique décision [2], les méthodes d'apprentissage profond produisent des représentations de requêtes et de documents qui ne sont pas compatibles avec un index inversé et qui ne permettent donc pas une recherche efficiente.

1. Par requête résolue, nous désignons des requêtes dont au moins un document pertinent est connu.
2. Dans le cas de la RI, il s'agit de calculer un score entre une requête et un document.

4.8 Conclusion

Dans ce chapitre, nous avons décrit l'utilisation de réseaux de neurones pour la RI, les différents types d'architectures et de fonctions objectif utilisés et avons fait un état des modèles de référence NLTR pour la RI. Nous avons présenté 3 limitations des modèles NLTR pour la RI : **(1)** la nécessité d'avoir de grandes quantités de requêtes résolues ; **(2)** l'absence de prise en compte de ressources sémantiques pour modéliser le langage ; **(3)** l'incompatibilité avec l'index inversé qui entraîne un manque d'efficience des modèles NLTR. Nous allons aborder ces trois limitations dans nos contributions décrites dans les chapitres suivants.

Deuxième partie

Contributions

Chapitre 5

WikIR

5.1 Introduction

L'apprentissage profond s'est révélé efficace dans diverses tâches de traitement automatique du langage naturel (TALN) telles que les modèles de langue, la compréhension du langage naturel et les systèmes de questions-réponses (Devlin et al., 2019). Cependant, de grandes quantités de données publiques et étiquetées sont des facteurs clés pour le développement de modèles d'apprentissage profond efficaces et réplicables (Goodfellow et al., 2016).

Malgré les progrès en TALN permis par les réseaux de neurones profonds, la RI *ad-hoc* sur des documents textuels n'a pas bénéficié de l'apprentissage profond autant que les autres domaines du TALN (Dehghani et al., 2017). Cela est principalement dû à : **(1)** la complexité de la tâche en utilisant uniquement des données non étiquetées (Dehghani et al., 2017) ; **(2)** le manque de collections ayant de grandes quantités de requêtes résolues et libres de droit qui sont cruciales pour développer et entraîner des modèles NLTR efficaces pour la RI *ad-hoc* (voir le Tableau 5.1).

Précisons que la collection *Yahoo ! LETOR* (Chapelle et Chang, 2011) contenant approximativement 30k requêtes résolues est en libre accès, mais seuls les vecteurs de caractéristiques décrivant les paires requête-document sont fournis. Cette collection convient aux modèles *learning-to-rank* basés sur les caractéristiques, mais pas aux réseaux de neurones qui nécessitent le contenu textuel original des requêtes et des documents.

Par conséquent, les modèles NLTR pour la RI *ad-hoc* sont développés en utilisant l'une des approches suivantes :

1. En utilisant de grandes quantités de données collectées par des moteurs de recherche commerciaux qui ne sont pas accessibles au public (par exemple

Collection	#Requêtes	#Documents	Moy $\#d^+/r$	Langue	Année
GOV2	150	25M	181.51	Anglais	2004
Robust04	250	0.5M	63.28	Anglais	2004
MQ2008	784	14k	3.82	Anglais	2008
MQ2007	1,692	65k	10.63	Anglais	2009
ClueWeb09	200	1B	74.62	Anglais	2009
Sogou-QCL	537k	5.4M	14.40	Chinois	2018
MSMACRO	372k	3.2M	1	Anglais	2020

TABLE 5.1 – Statistiques de plusieurs collections de RI *ad-hoc* en accès libre. Moy $\#d^+/r$ indique le nombre moyen de documents pertinents par requête.

le modèle DUET Mitra et al. (2017)). Ce processus est coûteux, long et n'est pas reproductible.

2. En utilisant des collections accessibles librement qui comportent peu de requêtes résolues, comme *MQ2007* et *MQ2008* Qin et al. (2010) (par exemple les modèles DeepRank Pang et al. (2017) et HiNT Fan et al. (2018). Cette approche peut limiter la conception et le nombre de paramètres des modèle en raison de la faible quantité de requêtes résolues.
3. En utilisant une supervision faible (*weak supervision*) qui consiste à pré-entraîner un modèle NLTR à imiter le classement d'une approche non supervisée Dehghani et al. (2017) (par exemple BM25). Cependant, cette méthode peut biaiser les modèles NLTR de façon à ce qu'ils ne classent que de la même manière que le modèle non supervisé.

Récemment, Zheng et al. (2018) ont proposé *Sogou-QCL*, une collection en chinois, réalisé à partir d'un moteur de recherche commercial. En 2016, Microsoft AI & Research a publié MS MARCO : un ensemble de collections axées sur l'apprentissage profond pour la recherche (Nguyen et al., 2016). MS MARCO est régulièrement mise à jour avec de nouvelles tâches telles que les systèmes de questions-réponses, l'extraction de mots clés ou le classement de passages. Cependant, la collection associée à la tâche de classement des documents (qui a été utilisée pour la tâche d'apprentissage profond de TREC 2019 (Craswell et al., 2020)) n'a été rendu publique que récemment (11 août 2020) et n'a pas pu être utilisée dans ce travail de thèse.

Dans ce travail de thèse, nous tirons parti de la nature semi-structurée de *Wikipédia* afin de construire de façon automatique des collections de RI. Nous proposons *WIKIR* (Frej et al., 2020d) : une boîte à outils (*toolkit*) en libre accès pour construire des collections de RI basées sur *Wikipédia* et ayant de grandes quanti-

tés [1] de requêtes résolues. Les collections libres d'accès créées à l'aide de *WIKIR* permettront d'entraîner et d'évaluer des modèles NLTR pour la RI. Nous proposons également d'utiliser *Wikipédia* en tant que source de connaissances *a priori* en pré-entraînant des modèles NLTR sur les collections créées par *WIKIR*. Ces modèles seront par la suite affinés (*fine-tuning*) sur des collections de RI traditionnelles ayant peu de requêtes résolues afin d'étudier si les connaissances *a priori* tirées de *Wikipédia* permettent de compenser le manque de requêtes résolues des collections de RI traditionnelles.

5.2 Un cadre général pour la création automatique de collection de RI

Dans cette section, nous proposons un cadre général pour créer automatiquement une collection de RI à partir d'une ressource $\mathcal{S}$ composée d'un ensemble de documents. Une collection de RI est composée de :

- $\mathcal{D}$, un ensemble de documents ;
- $\mathcal{R}$, un ensemble de requêtes ;
- $\mathcal{R}el$, un ensemble de jugements de pertinence (*relevance level*) pour toutes les paires requête-document (Manning et al., 2008).

5.2.1 Propriétés

Nous définissons 3 propriétés que $\mathcal{S}$ doit satisfaire pour être utilisé pour construire une collection de RI :

1. **Existence.** Il existe au moins un sujet (*topic* [2]) lié à chaque document de $\mathcal{S}$.
2. **Identification.** Il existe une fonction $id()$ qui associe à tout document de $\mathcal{S}$ l'ensemble des sujets qu'il contient.
3. **Description.** Il existe une fonction $desc()$ qui associe à chaque sujet une description courte et précise.

1. De l'ordre de grandeur de la dizaine de milliers.
2. Ici, nous utilisons le mot sujet ("*topic*") avec le sens : "Matière, thème, ce qui est en question." à notre connaissance, il n'existe pas de définition précise de "*topic*" faisant consensus en dans le domaine de la RI (Hjørland, 2001).

5.2.2 Construction du jeu de données

Dans ce qui suit, nous décrivons comment utiliser une ressource $\mathcal{S}$ qui satisfait les trois propriétés énumérées ci-dessus pour construire automatiquement une collection de RI.

Construction des documents

Nous choisissons un sous-ensemble de la ressource $\mathcal{S}$ pour construire l'ensemble des documents : $\mathcal{D} \subseteq \mathcal{S}$. Par exemple, si $\mathcal{S}$ est l'ensemble des articles de *Wikipédia*, nous pouvons choisir $\mathcal{D}$ comme l'ensemble des articles de *Wikipédia* contenant plus de 1000 mots.

Construction des requêtes

Nous commençons par identifier tous les sujets dans l'ensemble des documents $\mathcal{D}$ en utilisant la fonction $id()$:

$$\mathcal{T}_{\mathcal{D}} = \bigcup_{d \in \mathcal{D}} id(d),$$

avec $\mathcal{T}_{\mathcal{D}}$ l'ensemble de tous les sujets de $\mathcal{D}$. Cela est possible parce que la fonction $id()$ existe (*Identification*) et parce que chaque document est lié à au moins un sujet (*Existence*). Ensuite, nous utilisons la fonction $desc()$ sur tous les sujets pour construire l'ensemble des requêtes $\mathcal{R}$:

$$\mathcal{R} = \left\{ desc(t) \middle| t \in \mathcal{T}_{\mathcal{D}} \right\}.$$

Construction des jugements de pertinence

$\mathcal{R}el$ est l'ensemble de tous les triplets (requête, document, niveau de pertinence) :

$$\mathcal{R}el = \left\{ \left(r, d, rel(r,d)\right) \middle| \left(r, d\right) \in \mathcal{R} \times \mathcal{D} \right\},$$

avec $rel()$ une fonction qui associe chaque paire requête-document à un niveau de pertinence. Nous proposons d'attribuer un niveau de pertinence positif (désigné par val^{+}) au document d par rapport à la requête r si d contient le sujet qui a été utilisé pour construire r. Dans le cas contraire, un label de pertinence négatif ou nul (désigné par val^{-}) est attribué :

$$rel(r,d) = \begin{cases} val^{+} \in \mathbb{R}_{*}^{+}, & \text{si } t_r \in id(d), \\ val^{-} \in \mathbb{R}^{-}, & \text{sinon,} \end{cases}$$

avec t_r le sujet qui a été utilisé pour construire la requête r : $desc(t_r) = r$.

5.2.3 Le cas de *Wikipédia*

Dans cette sous-section, nous montrons que l'ensemble des articles de *Wikipédia* $\mathcal{W}$ satisfait les propriétés d'*Existence*, d'*Identifiabilité* et de *Descriptibilité*. Une description simplifiée du processus de construction d'une collection de RI à l'aide de 2 articles de *Wikipédia* est présentée dans la figure 5.1.

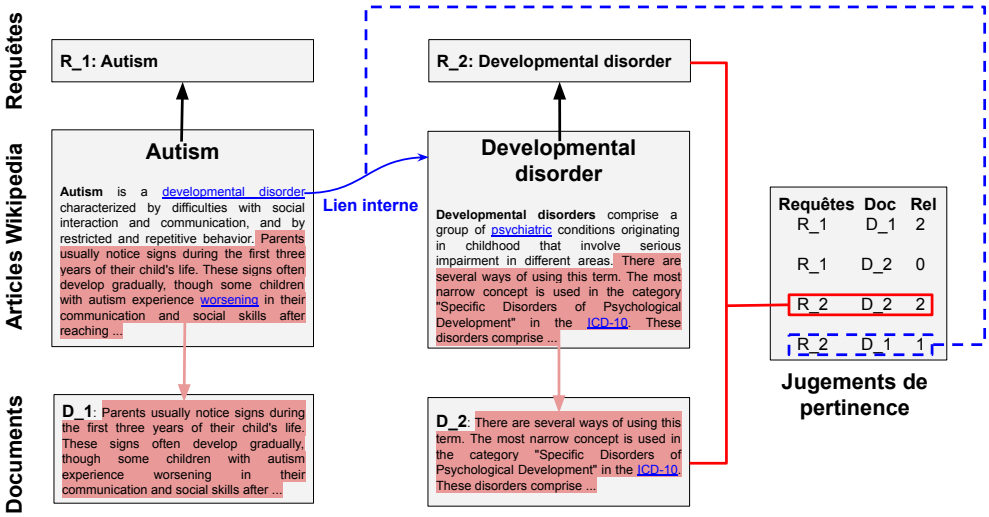

FIGURE 5.1 – Description du processus de construction d'une collection de RI par *WIKIR* en utilisant seulement deux articles. Dans cet exemple, les requêtes sont construites à partir du titre des articles. Les documents sont construits en utilisant le texte des articles sans le titre et sans la première phrase.

Existence

Chaque article *Wikipédia* est lié à au moins un sujet : son sujet principal [1].

1. En pratique, tous les articles *Wikipédia* ne sont pas forcément lié à un sujet, il existe par exemple des articles d'homonymie qui listent l'ensemble des articles partageant un même nom.

Identification

Nous supposons que si un article a contient un lien interne vers un autre article a_t dans sa première phrase (dénommée f_a), alors le sujet principal de a_t est un sujet de a. L'intuition derrière cette hypothèse est que la première phrase de la plupart des articles *Wikipédia* est une bonne description du contenu de l'article (Sasaki et al., 2018) et si un lien est présent, il pointe vers un sujet important de l'article considéré. Nous proposons donc de définir $id()$ comme suit :

$$id(a) = \{s_a\} \bigcup \left\{ s_{a_t} \in \mathcal{W} \middle| \exists\ f_{a_t} \underset{\text{lien}}{\longrightarrow} a \right\}, \tag{5.1}$$

avec s_a le sujet principal de l'article a et $f_{a_t} \underset{\text{lien}}{\longrightarrow} a$ dénote la présence d'un lien interne dans la première phrase de l'article a_t qui pointe vers l'article a. Ainsi, $id()$ considère l'ensemble des sujets liés à l'article a comme le sujet principal de a : s_a et le sujet principal de tous les articles qui pointent vers a dans leur première phrase. Par exemple, l'ensemble des sujets liés à l'article *Désordre du développement* est composé de son sujet principal et du sujet principal de l'article *Autisme* parce qu'il y a un lien dans la première phrase de l'article *Autisme* qui pointe vers l'article *Désordre du développement* (voir Figure 5.1).

Description

Nous proposons d'associer une description courte et précise aux sujets principaux des articles de *Wikipédia* en utilisant le titre des articles :

$$desc(s_a) = titre_a, \tag{5.2}$$

avec $titre_a$ le titre de l'article a. Afin d'étudier la robustesse des modèles NLTR, nous proposons également d'associer une description longue et bruitée aux sujets principaux des articles de *Wikipédia* en utilisant la première phrase des articles :

$$desc(s_a) = f_{a_t}. \tag{5.3}$$

5.3 Description de la boîte à outils WIKIR

Dans cette section, nous décrivons la boîte à outils *WIKIR* et rendons explicites les motivations qui sous-tendent certaines décisions de conception. Pour une liste

Nous les identifions à l'aide de la présence de la chaîne de caractères "*(disambiguation)*" dans le titre et ne les considérons pas lors de la construction de la collection.

exhaustive des options disponibles et pour avoir des exemples sur la façon d'utiliser *WIKIR*, le lecteur pourra consulter notre dépôt *github* : https://github.com/getalp/wikIR.

5.3.1 *WIKIR* pour créer des collections de RI

Pour créer une collection de RI à l'aide d'une base de données au format XML issue de *Wikimedia*, [1] *WIKIR* suit 3 étapes principales : construction, traitement et stockage.

Construction

Lecture de la base données XML. Nous utilisons *WikiExtractor* [2] pour extraire du texte à partir de la base de données au format XML. Nous obtenons ainsi un fichier *json* (décrit sur la Figure 5.2) qui contient l'url, le titre et le texte de tous les articles de *Wikipédia*. Lorsque nous utilisons *wikiextractor*, nous utilisons l'option permettant de préserver les liens dans le texte afin de construire les jugements de pertinence.

Extraction des documents. L'ensemble de documents $\mathcal{D}$ est extrait en utilisant le champ "text" associé à chaque article du fichier *json* produit lors de l'étape précédente. La première ligne du champ"text" (qui correspond au titre de l'article) est supprimée. Nous supprimons les titres afin d'éviter de faciliter considérablement la tâche de classement pour les modèles NLTR puisqu'ils prennent en compte l'ordre des mots. En effet, étant donné une requête, le document le plus pertinent commencera toujours par la requête elle-même.

Construction des requêtes. Comme décrit dans la section 5.2.2, pour construire des requêtes, nous avons besoin d'une fonction $id()$ et d'une fonction $desc()$. *WIKIR* utilise la fonction $id()$ définie par l'équation (5.1). La fonction $desc()$ est définie à l'aide de l'équation (5.2) ou de l'équation (5.3). De façon moins formelle, les sujets sont identifiés à l'aide de liens internes et sont décrits à l'aide des titres des articles ou des premières phrases des articles. Le processus de construction des requêtes est le même que dans la section 5.2.2.

Construction des jugements de pertinence. Comme expliqué dans la section 5.2.2, afin de construire $\mathcal{R}el$, nous devons définir la fonction $rel()$. Pour ce faire, nous supposons que le document le plus pertinent pour une requête est le document

1. https://dumps.wikimedia.org/backup-index.html
2. https://github.com/attardi/wikiextractor

```
1    {"id": "12",
2     "url": "https://en.Wikipedia.org/wiki?curid=12",
3     "title": "Anarchism",
4     "text": "Anarchism\n\nAnarchism is an <a href=\"
        anti-authoritarian\">anti-authoritarian</a> <a
        href=\"political%20philosophy\">political
        philosophy</a> that advocates ... "
5    }
6    {"id": "25",
7     "url": "https://en.Wikipedia.org/wiki?curid=25",
8     "title": "Autism",
9     "text": ""Autism\n\nAutism is a <a href=\"
        developmental%20disorder\">developmental
        disorder</a> characterized by difficulties with
         ... "
10   }
11   ...
```

FIGURE 5.2 – *json* file extracted from English *Wikipédia* dump using *WikiExtractor*

construit à partir du même article que la requête. Par conséquent, nous définissons $rel()$ comme :

$$rel(r,d) = \begin{cases} 2, & \text{si } a_r = a_d, \\ 1, & \text{si } a_d \in id(d) \backslash a_d, \\ 0, & \text{sinon,} \end{cases}$$

avec a_r (respectivement a_d) l'article *Wikipédia* utilisé pour construire la requête r (respectivement le document d). Ainsi, nous attribuons un niveau de pertinence égal à 2 pour les paires requête-document qui proviennent du même article. Nous attribuons un niveau de pertinence égal à 1 aux paires requête-document pour lesquelles il existe un lien dans la première phrase de l'article du document qui pointe vers l'article de la requête. Par exemple, si nous considérons la requête *'Developmental disorder"*, le document le plus pertinent (pertinence = 2) est *"Developmental disorders comprise a group of ... "* parce que la requête et le document sont construits à partir du même article. Le document *"Autism is a developmental disorder characterized by ... "* est pertinent (pertinence = 1) parce que l'article *Autisme* contient un lien vers l'article *Developmental disorder* (voir Figure 5.1).

Traitement

Sélection des requêtes. Afin d'avoir des collections équilibrées, nous ne sélectionnons que les requêtes qui comportent un nombre minimum de documents pertinents (5 par défaut). Nous avons également limité la longueur des requêtes à un maximum de 10 mots.

Pré-traitement. *WIKIR* supprime les cibles dans les références hypertext mais conserve le texte. Par exemple, *"worsening"* devient *"worsening"*. De plus, tout caractère non alphanumérique est supprimé. Par défaut, *WIKIR* met également en minuscules tous les caractères.

Séparation en ensembles d'entraînement, de validation et de test. Les requêtes et leurs jugements de pertinence associés (*qrels*) sont séparés de façon aléatoire en ensembles d'entraînement, de validation et de test. Par défaut, 80% des requêtes (et des *qrels* associés) sont placés dans l'ensemble d'entraînement, 10% dans l'ensemble de validation et 10% dans l'ensemble de test. Le même ensemble de documents est considéré dans les ensembles d'entraînement, de validation et de test puisque en RI *ad-hoc*, nous considérons avoir un ensemble fixe de documents à classer (Baeza-Yates et Ribeiro-Neto, 2011).

Stockage

WIKIR créée un dossier qui contient l'ensemble des documents dans le fichier *documents.format* ainsi que trois sous-dossiers : *training*, *validation* et *test*. Chacun d'entre eux contient deux fichiers :

1) *qrels* qui contient les jugements de pertinence des requêtes dans le format *trec_eval*.[1] Pour limiter la taille du fichier *qrels*, nous ne sauvegardons que les niveaux de pertinence positifs. Cela signifie que chaque paire requête-document qui n'est pas enregistrée dans le fichier *qrels* est non-pertinente.

2) *queries.format* qui contient les requêtes.

Par défaut, les requêtes et les documents sont enregistrés sous forme de *DataFrame* au format *csv*. Le format des fichiers est compatible avec la bibliothèque *MatchZoo* (Guo et al., 2019a) : une bibliothèque *Python* implémentant des modèles d'apprentissage profond pour des tâches de comparaison de texte telles que la RI et les systèmes question-réponses.

Les requêtes et les documents peuvent également être enregistrés dans un fichier au format XML compatible avec le système de recherche d'information *Terrier*.

1. https://trec.nist.gov/trec_eval/

5.3.2 WIKIR pour BM25

Motivation

Une fois le jeu de données créé, *WIKIR* peut être utilisé pour appliquer Okapi BM25 (Robertson et Walker, 1994). Nous proposons cette option, car la grande majorité des modèles NLTR développés pour la RI *ad-hoc* ne sont pas compatibles avec un index inversé (Zamani et al., 2018). Afin de pouvoir classer des documents en temps réel, les modèles NLTR utilisent la stratégie dite de reclassement :

— un modèle efficient tel que BM25 est utilisé pour effectuer un premier classement des documents

— les documents les mieux classés par le modèle efficient sont sélectionnés (par exemple les 1000 documents ayant le score le plus élevé)

— les documents sélectionnés à l'étape précédente sont reclassés par le modèle NLTR

Cette stratégie permet aux modèles NLTR de ne comparer une requête qu'à un sous-ensemble de taille restreinte de l'ensemble des documents. Il s'agit ainsi de permettre la réalisation d'un classement en temps réel. La principale limitation de la stratégie de reclassement réside dans le fait que tout document pertinent qui n'est pas sélectionné par le modèle efficient ne sera pas retourné par le système. En d'autres termes, le rappel d'un modèle NLTR utilisant la stratégie de reclassement est borné par le rappel du modèle efficient utilisé.

Résultats

Les résultats de BM25 sont enregistrés dans les sous-dossiers *training*, *validation* et *test*. Trois fichiers sont créés dans chacun de ces sous-dossiers :

1) *BM25.res* qui contient les résultats de BM25 enregistrés au format *trec_eval*.

2) *BM25.metrics.json* qui contient les valeurs de plusieurs mesures d'évaluation de RI.

3) *BM25.qrels.csv* qui contient les k documents (1000 par défaut) les plus pertinents pour chaque requête selon BM25 ainsi que leur niveau de pertinence associé.

5.3.3 Reclassement neuronal avec *WIKIR*

WIKIR peut être utilisé pour entraîner et évaluer des modèles NLTR sur les collections qu'il a créées. Comme expliqué dans la section 5.3.2, nous effectuons un reclassement neural en utilisant BM25 comme classeur initial. Nous avons utilisé la

bibliothèque *Python MatchZoo* implémentant des modèles d'apprentissage profond pour des tâches de comparaison de texte pour l'entraînement et l'évaluation des modèles NLTR. Nous avons utilisé *MatchZoo* parce qu'elle a été acceptée comme un outil fiable de correspondance neuronal de texte (Guo et al., 2019a). Tout modèle disponible dans *MatchZoo* peut être entraîné et évalué avec *WIKIR*. Une fois l'entraînement terminé et les classements des documents sauvegardés, *WIKIR* peut être utilisé pour calculer les mesures d'évaluation et la significativité statistique.

5.3.4 Pré-entraînement des modèles NLTR avec *WIKIR*

WIKIR permet également de sauvegarder les modèles NLTR entraînés sur les collections qu'il a créées. Ces modèles peuvent, par la suite, être affinés sur des collections de RI ayant peu de requêtes résolues. Cette technique de pré-entraînement et d'affinage permet d'incorporer des connaissances *a priori* venant de *Wikipédia* aux modèles NLTR et leur permet ainsi de ne nécessiter que peu de requêtes résolues afin de pouvoir surclasser des modèles basés sur la correspondance exacte.

5.4 Protocole expérimental

Afin d'évaluer l'utilité des collections créées par *WIKIR* pour réduire la quantité de requêtes résolues nécessaires pour que les modèles NLTR surpassent les modèles de référence BM25 et BM25-RM3, nous avons procédé de la façon suivante :

1. Pré-entraînement de modèles NLTR pour la RI sur les collections *Wikipédia*
2. Sélection des modèles NLTR à performance égale ou supérieur aux modèles de référence sur les collections *Wikipédia*
3. Utilisation de la méthode d'apprentissage fin [1] (*fine-tuning*) pour entraîner les réseaux sélectionnés sur les collections TREC (qui ont peu de requêtes résolues).

5.4.1 Collections de Recherche d'Information

Collections *Wikipédia*

Dans ce qui suit, nous décrivons wikIR78k et wikIRS78k : les deux collections créées par *WIKIR* que nous avons utilisés dans nos expériences pour pré-entraîner les modèles NLTR.

1. L'apprentissage fin consiste utiliser un réseau de neurones pré-entraîné sur un jeu de données et à continuer à l'entraîner sur un nouveau jeu de données.

wikIR78k. wikIR78k est une collection contient 78 631 requêtes résolues. Pour construire wikIR78k, nous avons utilisé l'ensemble complet des articles de *Wikipédia*. Pour construire les requêtes, nous avons utilisé les titres des articles. De plus, nous avons supprimé la première phrase de chaque article lors de la construction des documents. Nous avons fait ce choix, car toutes les informations que nous utilisons pour évaluer la pertinence sont contenues dans la première phrase des articles (voir Section 5.2.2) et nous ne voulons pas que les modèles qui prennent en compte l'ordre des mots utilisent ce biais à leur avantage.

wikIRS78k. Le procédé de construction de wikIRS78k est le même que celui de wikIR78k, à l'exception de la construction des requêtes : nous avons utilisé les premières phrases des articles au lieu de leurs titres. Nous proposons une collection avec des requêtes courtes et bien définies (wikIR78k) et une collection avec des requêtes longues et bruitées (wikIRS78k) pour étudier la robustesse des modèles de RI face à des requêtes bruitées. Les statistiques des collections sont affichées sur le tableau 5.2.

	wikIRS78k	wikIR78k
Nombre de documents	2.4M	2.4M
Longueur moyenne des documents	744.58	744.58
Nombre de requêtes	78k	78k
Longueur moyenne des requêtes	2.45	9.80
Nombre moyen de document pertinent par requête	39.02	39.02

TABLE 5.2 – Statistiques des collections wikIR78k et wikIRS78k.

Collections TREC

TREC (*Text REtrieval Conference*) a pour but de soutenir la communauté de chercheurs en recherche d'information en fournissant des collection et des outils pour l'évaluation de méthodologies de RI textuelle à grande échelle. Nous utilisons 3 collections TREC pour la RI *ad-hoc* pour affiner les modèles NLTR pré-entraînés sur les collections *Wikipédia* :

— AP88-89 composé de 150 requêtes résolues, de 164 597 documents et de 15 856 jugements de pertinence positifs.

— FT91-94 composé de 200 requêtes résolues, de 210 158 documents et de 6 486 jugements de pertinence positifs.

— LA composé de 150 requêtes résolues, de 131 896 documents et de 3 553 jugements de pertinence positifs.

Des exemples de requêtes et de document pertinents associés tirés des collections TREC sont illustrés sur le Tableau 5.3.

Collection	Requête	Document pertinent associé
AP88-89	Design of the "Star Wars" Anti-missile Defense System	A research satellite launched last week to test elements of the proposed Star Wars antimissile shield ...
FT91-94	Combating Alien Smuggling vs. drugs	INTERIOR ministers from 35 European countries yesterday agreed to crack down on the wave of illegal immigration from ...
LA	International Organized Crime	Nearly two years after he challenged the United States to bring him to justice on drug charges Manuel A Noriega stood before ...

TABLE 5.3 – Exemples de requêtes et d'un document pertinent associé provenant du NFCorpus.

Nous utilisons la validation croisée à 5 blocs (*5-fold cross validation*) pour l'entraînement et la sélection des hyperparamètres des modèles sur les collections TREC.

5.4.2 Modèles de référence basés sur la correspondance exacte

Nous utilisons deux modèles de référence basés sur la correspondance exacte : BM25 et BM25-RM3. L'indexation des collections ainsi que l'évaluation de BM25 et BM25-RM3 sont faites par le système de RI Terrier [1]. Les hyperparamètres k_1 et b de BM25 sont sélectionnés avec la méthode de recherche en grille (*grid search*) parmi les intervalles $[0, 8]$ avec un pas de 0.1 pour k_1 et $[0, 1]$ avec un pas de 0.05 pour b. Les hyperparamètres de BM25-RM3 sont également sélectionnés avec la méthode de recherche en grille : le nombre de documents pertinents est choisi parmi l'intervalle $[5, 30]$ avec un pas de 5, le nombre maximal de termes à ajouter à la requête est choisi parmi l'intervalle $[10, 50]$ avec un pas de 10 et α est choisi parmi

1. http://terrier.org/

l'intervalle $[0, 1]$ avec un pas de 0.05. Les hyperparamètres sont choisis pour maximiser la ndcg@5 sur les ensembles de validation.

5.4.3 Modèles NLTR

Nous entraînons et évaluons les modèles NLTR pour la RI avec la bibliothèque *MatchZoo* (Guo et al., 2019a). Nous utilisons les embeddings *Word2vec* (Mikolov et al., 2013a) de dimension 300 fourni par *MatchZoo*. Nous utilisons la méthode d'optimisation Adam Kingma et Ba (2015) avec un taux d'apprentissage égal à 0.001 sur les collections *Wikipédia* et égal à 0.0001 sur les collections TREC. Nous utilisons la fonction de perte d'entropie croisée *pairwise* fournie par *MatchZoo*. Les hyperparamètres des différents modèles NLTR (nombre de couches, choix de la fonction d'activation, taille des couches, ...) sont sélectionnés par recherche aléatoire (*random search*) sur 20 exécutions en utilisant le *Tuner* de *MatchZoo* qui fourni les intervalles de valeurs possibles pour l'ensemble des hyperparamètres de chacun des modèles. Les hyperparamètres sont choisis pour maximiser la ndcg@5 sur les ensembles de validation. Les modèles NLTR sont évalués en utilisant la méthode de reclassement sur les 1000 premiers documents de BM25-RM3.

5.4.4 Évaluation

Les mesures d'évaluation sont calculées sur les 1000 documents les mieux classés par les modèles. Les mesures d'évaluation utilisées sont calculées avec *pytrec_eval* (Gysel et de Rijke, 2018) : un *wrappeur python* de l'outil d'évaluation *trec_eval* [1], communément utilisé en RI. Nous utilisons un test t apparié (Urbano et al., 2013) pour mesurer la significativité des différences de résultats entre les réseaux de neurones et BM25-RM3.

5.5 Résultats

5.5.1 Modèles NLTR sur les collections *Wikipédia*

Les performances des modèles de références et des modèles NLTR sur WikIR78k et WikIRS78k sont illustrés respectivement sur les Tableaux 5.4 et 5.5.

1. https://trec.nist.gov/trec_eval/

WikIR78k

Rappelons que les requêtes de la collection WikIR78k sont construites en utilisant les titres des articles *Wikipédia* et sont par conséquent courtes et bien définies. Comme on peut le voir sur le Tableau 5.4, lorsque les requêtes sont courtes et bien définies (wikIR78k), BM25-RM3 est un modèle de référence solide. En effet, parmi les modèles NLTR, seul DRMM parvient à surpasser BM25-RM3 de façon statistiquement significative sur l'ensemble des mesures. En outre, même si les modèles DUET et Conv-KNRM obtiennent de meilleurs résultats que BM25, ils ne parviennent pas à obtenir de meilleurs résultats que BM25-RM3. Aucun des autres modèles NLTR testés ne parvient à obtenir des performances similaires à BM25 et BM25-RM3. Cela tend à montrer que lorsque les requêtes sont courtes et bien définies, les modèles de correspondance exacte fournissent des résultats difficilement surpassables par des modèles NLTR même si ces derniers disposent de plusieurs dizaines de milliers de requêtes résolues pour leur entraînement.

WikIRS78k

Rappelons que les requêtes de la collection WikIRS78k sont construites en utilisant les premières phrases des articles *Wikipédia* et sont par conséquent longues et bruitées. Comme on pouvait s'y attendre, BM25 et BM25-RM3 obtiennent de moins bonne performances sur des requêtes longues et bruitées (-21.34% et -16.97%, respectivement sur la P@5 comparé à wikIR78k). C'est également le cas pour les modèles NLTR DRMM (-15.47% sur la P@5 comparé à wikIR78k) et DUET (-7.73% sur la P@5 comparé à wikIR78k), mais il est intéressant de noter que ce n'est pas le cas pour le modèle CKNRM qui obtient des performances similaires (-0.46% sur la P@5 comparé à wikIR78k). De plus, les modèles neuronaux DUET, DRMM et CKNRM parviennent à surpasser BM25-RM3 de façon statistiquement significative sur l'ensemble des mesures. Ces résultats tendent à montrer que lorsque les requêtes sont longues et bruitées, les modèles NLTR fournissent des résultats de meilleurs qualités que les modèles de référence basés sur la correspondance exacte. Cela est probablement dû au fait que, étant donné un nombre suffisant de requêtes résolues, les modèles NLTR ont la capacité de se débarrasser du bruit présent dans les requêtes.

wikIR78k								
Modèle	P@5	P@10	P@20	nDCG@5	nDCG@10	nDCG@20	nDCG	MAP
BM25	0.2622	0.2039	0.1498	0.3269	0.3045	0.3098	0.3555	0.1498
BM25-RM3	0.2844	0.2235	0.1663	0.3442	0.3238	0.3305	0.3813	0.1710
ArcI	0.1604^{-}	0.1524^{-}	0.1325^{-}	0.1644^{-}	0.1666^{-}	0.1934^{-}	0.2701^{-}	0.1025^{-}
ArcII	0.1669^{-}	0.1556^{-}	0.1410^{-}	0.1682^{-}	0.1737^{-}	0.1957^{-}	0.2752^{-}	0.1102^{-}
MatchPyramid	0.2481^{-}	0.2136^{-}	0.1595	0.2862^{-}	0.2638^{-}	0.2957^{-}	0.3423^{-}	0.1422^{-}
DRMM	$\mathbf{0.2941}^{+}$	$\mathbf{0.2312}^{+}$	0.1745^{+}	$\mathbf{0.3659}^{+}$	$\mathbf{0.3394}^{+}$	$\mathbf{0.3420}^{+}$	$\mathbf{0.3861}^{+}$	$\mathbf{0.1773}^{+}$
DUET	0.2819	0.2243	0.1721^{+}	0.3520^{+}	0.3239	0.3297	0.3746^{-}	0.1656^{-}
KNRM	0.1482^{-}	0.1408^{-}	0.1278^{-}	0.1388^{-}	0.1487^{-}	0.1700^{-}	0.2614^{-}	0.0961^{-}
CKNRM	0.2792	0.2268	$\mathbf{0.1762}^{+}$	0.3285^{-}	0.3114^{-}	0.3199^{-}	0.3622^{-}	0.1599^{-}

TABLE 5.4 – Comparaison des performances des différents modèles sur wikIR78k. Les amélioration/dégradations statistiquement significatives par rapport à BM25-RM3 sont notées par (+/-) avec une p-value < 0.01.

wikIRS78k

Modèle	P@5	P@10	P@20	nDCG@5	nDCG@10	nDCG@20	nDCG	MAP
BM25	0.2177	0.1634	0.1186	0.2944	0.2673	0.2695	0.3085	0.1163
BM25-RM3	0.2237	0.1698	0.1243	0.3015	0.2733	0.2753	0.3158	0.1216
ArcI	0.1241^{-}	0.1174^{-}	0.1051^{-}	0.1181^{-}	0.1277^{-}	0.1520^{-}	0.2199^{-}	0.0745^{-}
ArcII	0.1513^{-}	0.1374^{-}	0.1189^{-}	0.1440^{-}	0.1524^{-}	0.1749^{-}	0.2349^{-}	0.0842^{-}
MatchPyramid	0.2140^{-}	0.1742	0.1363^{+}	0.2378^{-}	0.2329^{-}	0.2435^{-}	0.2779^{-}	0.1104^{-}
DRMM	0.2486^{+}	0.1867^{+}	0.1404^{+}	0.3306^{+}	0.2991^{+}	0.2976^{+}	0.3288^{+}	0.1361^{+}
DUET	0.2601^{+}	0.1998^{+}	0.1479^{+}	0.3318^{+}	0.3051^{+}	0.3042^{+}	0.3281^{+}	0.1367^{+}
KNRM	0.1490^{-}	0.1291^{-}	0.1066^{-}	0.1546^{-}	0.1590^{-}	0.1757^{-}	0.2350^{-}	0.0816^{-}
CKNRM	$\mathbf{0.2779}^{+}$	$\mathbf{0.2148}^{+}$	$\mathbf{0.1567}^{+}$	$\mathbf{0.3361}^{+}$	$\mathbf{0.3134}^{+}$	$\mathbf{0.3150}^{+}$	$\mathbf{0.3369}^{+}$	$\mathbf{0.1465}^{+}$

TABLE 5.5 – Comparaison des performances des différents modèles sur wikIRS78k. Les amélioration/dégradations statistiquement significatives par rapport à BM25-RM3 sont notées par (+/-) avec une p-value < 0.01.

Sélection des réseaux de neurones

Nous n'utilisons la méthode d'apprentissage fin sur les collections TREC qu'avec les modèles DRMM, DUET et CKNRM puisque ce sont les seuls modèles NLTR qui sont parvenus à surpasser BM25 et BM25-RM sur au moins une des deux collection *Wikipédia*.

5.5.2 Réseaux pré-entraînés sur TREC

Les résultats de nos modèles sur les collections TREC sont illustrés sur le Tableau 5.6. Les gains de performances obtenus à l'aide du pré-entraînement sur les collections *Wikipédia* sont illustrés sur le Tableau 5.7.

Modèles sans pré-entraînement

Dans un premier temps, nous constatons que sur les 3 collections TREC, aucun des modèles NLTR sans pré-entraînement ne parvient à surpasser BM25-RM3. Seul DRMM parvient à avoir des résultats comparables à BM25, et ce, malgré la faible quantité de données d'apprentissage disponible sur les collections TREC. Cela est probablement dû au fait que DRMM prend en compte de façon explicite le signal de correspondance exacte et que DRMM contient peu de paramètres à optimiser (455 paramètres) et qu'il ne nécessite donc pas beaucoup de données d'apprentissage pour obtenir des performances similaires à celles de BM25. Ces résultats sont cohérents avec d'autres expériences qui utilisent des modèles implémentés par *MatchZoo* sur d'autres collections avec quelques centaines de requêtes résolues (Yang et al., 2019a).

Modèles pré-entraînés sur WikIR78k

Le fait de pré-entraîner les modèles neuronaux sur WikIR78k (notés $\text{DRMM}_{\text{WikIR}}$, $\text{DUET}_{\text{WikIR}}$, et $\text{CKNRM}_{\text{WikIR}}$) permet d'augmenter leurs performances sur toutes les collections TREC. Cela est particulièrement le cas pour les modèles DUET et CKNRM sur les collections LA et FT91-94 dont la MAP est multipliée jusqu'à 5 (voir Tableau 5.7). Ces gains importants sont dus au fait que les modèles DUET et CKNRM ont des performances faibles sur LA et FT91-94 sans pré-entraînement, ce qui laisse un grande marge aux améliorations. Puisque le modèle DRMM sans pré-entraînement à déjà de bonnes performances (relativement aux modèles de référence), les gains relatifs qu'il obtient avec un pré-entraînement sur WikIR78k sont moins importants que les autres modèles neuronaux (de +2.8% à +13.7%).

$DRMM_{WikIR}$ et $CKNRM_{WikIR}$ ont globalement des performances similaires à celles de BM25-RM3 tandis que $DUET_{WikIR}$ est moins performant que BM25-RM3 de façon statistiquement significative sur les collections LA et FT91-94.

Ces résultats montrent que le fait de pré-entraîner des réseaux de neurones pour la RI sur WikIR78k permet d'avoir de bien meilleures performances que celles des mêmes modèles sans pré-entraînement sur des collections n'ayant qu'entre 150 et 200 requêtes résolues.

Modèles pré-entraînés sur WikIRS78k

Nous pouvons constater sur le Tableau 5.7 que les gains de performance sont plus importants lorsque les modèles neuronaux sont pré-entraînés sur WikIRS78k plutôt que sur WikIR78k. De plus, $DRMM_{WikIR}$, $DUET_{WikIR}$, et $CKNRM_{WikIR}$ obtiennent tout trois des améliorations de performances statistiquement significatives comparées à BM25-RM3 [1].

Ces résultats suggèrent que les modèles pré-entraînés sur une collection avec des requêtes longues et bruitées (WikIRS78k) permettent d'atteindre de meilleurs performances que les modèles pré-entraînés sur une collection avec des requêtes courtes et bien définies (WikIR78k). Nous suggérons que ces gains de performances sont dus au fait que les modèles entraînés sur WikIRS78k sont plus robustes que ceux entraînés sur WikIR78k dans le sens où ils sont capables de distinguer les termes des requêtes importants de ceux qui le sont moins.

5.6 Conclusions

Dans ce chapitre, nous proposons *WIKIR* un *toolkit* pour la construction de collections de RI à grande échelle à partir de *Wikipédia*. *WIKIR* peut également être utilisé pour entraîner et évaluer des modèles neuronaux pour la RI. Nous proposons un cadre général pour construire une collection de RI à partir de n'importe quelle ressource satisfaisant trois propriétés. En outre, nous avons construit et mis à en libre accès wikIR78k et wikIRS78k : deux collection de RI à grande échelle [2] en utilisant *WIKIR*.

Nous avons utilisé wikIR78k et wikIRS78k pour pré-entraîner des modèles neuronaux pour la RI et nous les avons affinés sur des collections TREC ayant peu

1. à l'exception de $DUET_{WikIR}$ sur la collection FT91-94 qui a bien de meilleurs performances que BM25-RM mais dont les améliorations ne sont pas statistiquement significatives
2. ayant 79k requêtes résolues

de requêtes résolues. Nous montrons que les modèles neuronaux pré-entraînés atteignent des performances supérieures à leurs homologues sans pré-entraînement et, si pré-entraînés sur wikIRS78k, parviennent également à surpasser sur toutes les mesures BM25-RM3 le modèle de référence basé sur la correspondance exacte malgré la faible quantité de requêtes résolues.

	AP88-89			**LA**			**FT91-94**		
Modèle	P@5	ndcg@5	MAP	P@5	ndcg@5	MAP	P@5	ndcg@5	MAP
BM25	0.4573	0.4470	0.2755	0.3093	0.3498	0.2221	0.3330	0.3531	0.2351
BM25-RM3	0.4720	0.4599	0.2908	0.3240	0.3626	0.2357	0.3473	0.3667	0.2513
DRMM	0.4521	0.4405	0.2665^{-}	0.3162	0.3622	0.2254	0.3344	0.3595	0.2465
DUET	0.4265^{-}	0.4386^{-}	0.2532^{-}	0.1753^{-}	0.1526^{-}	0.0522^{-}	0.1804^{-}	0.1688^{-}	0.0857^{-}
CKNRM	0.4352^{-}	0.4413	0.2563^{-}	0.2044^{-}	0.2265^{-}	0.1130^{-}	0.3263	0.3595	0.2281^{-}
$\text{DRMM}_{\text{WikIR}}$	0.4897	0.4758	0.3031	0.3317	0.3725	0.2426	0.3598	0.3698	0.2672
$\text{DUET}_{\text{WikIR}}$	0.4631	0.4761^{+}	0.2901	0.3013^{-}	0.2979^{-}	0.2188	0.2990^{-}	0.2876^{-}	0.2051^{-}
$\text{CKNRM}_{\text{WikIR}}$	0.4734	0.4782^{+}	0.2923	0.3147	0.3354	0.2237	0.3552	0.3880	0.2551
$\text{DRMM}_{\text{WikIRS}}$	0.4997^{+}	0.4858^{+}	0.3135^{+}	0.3427^{+}	0.3816	0.2545^{+}	0.3786^{+}	0.3808	0.2794^{+}
$\text{DUET}_{\text{WikIRS}}$	0.5332^{+}	0.5461^{+}	$\mathbf{0.3551}^{+}$	0.3863^{+}	0.3965^{+}	0.2743^{+}	0.3511	0.3719	0.2704
$\text{CKNRM}_{\text{WikIRS}}$	$\mathbf{0.5450}^{+}$	$\mathbf{0.5507}^{+}$	0.3483^{+}	$\mathbf{0.4021}^{+}$	$\mathbf{0.4105}^{+}$	$\mathbf{0.2824}^{+}$	$\mathbf{0.4243}^{+}$	$\mathbf{0.4431}^{+}$	$\mathbf{0.2898}^{+}$

TABLE 5.6 – Performances sur les collections TREC des modèles de références, des réseaux de neurone, des réseaux de neurone pré-entraînés sur wikIR78k et des réseaux de neurone pré-entraînés sur wikIRS78k. Les amélioration/dégradations statistiquement significatives par rapport à BM25-RM3 sont notées par (+/-) avec une p-value < 0.01.

	AP88-89			**LA**			**FT91-94**		
Modèle	P@5	ndcg@5	MAP	P@5	ndcg@5	MAP	P@5	ndcg@5	MAP
$\text{DRMM}_{\text{WikIR}}$	+8.3%	+8.0%	+13.7%	+4.9%	+2.8%	+7.6%	+7.5%	+2.8%	+8.3%
$\text{DUET}_{\text{WikIR}}$	+8.5%	+8.5%	+14.5%	+71.8%	+95.2%	+319%	+65.7%	+70.3%	+139%
$\text{CKNRM}_{\text{WikIR}}$	+8.7%	+8.3%	+14.0%	+53.9%	+48.0%	+97.9%	+8.8%	+7.9%	+11.8%
$\text{DRMM}_{\text{WikIRS}}$	+10.5%	+10.2%	+17.6%	+8.3%	+5.3%	+12.9%	+13.2%	+5.9%	+13.3%
$\text{DUET}_{\text{WikIRS}}$	+25.0%	+24.5%	+40.2%	+120%	+159%	+425%	+94.6%	+120%	+215%
$\text{CKNRM}_{\text{WikIRS}}$	+25.2%	+24.7%	+35.8%	+96.7%	+81.2%	+149%	+30.0%	+23.2%	+27.0%

TABLE 5.7 – Gains de performance des modèles neuronaux pré-entraînés comparés aux modèles sans pré-entraînement

Chapitre 6

Réseaux de Neurones Sémantiques pour la Recherche d'Information

6.1 Introduction

Dans des domaines spécialisés comme le domaine médical, les non-spécialistes expriment leurs requêtes en langage "courant", alors que les documents contiennent des termes spécifiques à un domaine. Par exemple, si un utilisateur demande *"Comment les régimes à base de plantes peuvent prolonger la vie ?"*, un système de recherche d'information (RI) utilisant un modèle basé sur la correspondance exacte ne pourra pas récupérer des documents pertinents tels que *"Un examen de la dépendance à la méthionine et du rôle de la restriction de méthionine dans la lutte contre le cancer et l'allongement de l'espérance de vie"*. Pour récupérer ce document, un système de RI doit pouvoir associer *"régimes à base de plantes"* avec *"restriction en méthionine"*.

D'une part, les modèles *Neural Learning-To-Rank* (NLTR) qui utilisent des vecteurs de mots appris sur de grandes quantités de texte brut forment une approche prometteuse à ce problème.

D'autre part, il est possible d'utiliser des ressources sémantiques pour étendre les requêtes et/ou les documents avec des concepts. Cela permet d'aborder le problème de disparité des termes puisque le même concept peut être associé à des mots appartenant à des vocabulaires non-spécialistes et experts. Cependant, il s'agit d'une tâche difficile car la ressource sémantique peut être incomplète, conduire à de l'ajout de bruit et nécessiter des adaptations au cas par cas (Jimmy et al., 2019). Dans ce travail, nous étudions la possibilité pour les modèles NLTR d'ignorer le bruit introduit par les ressources sémantiques et de se concentrer sur les connais-

sances pertinentes pour améliorer la recherche.

Nous proposons des modèles NLTR dits sémantiques (Frej et al., 2020b; FREJ et al., 2020) qui utilisent :

— des vecteurs de mots pré-entraînés sur une grande quantité de texte.

— des vecteurs de concepts pré-entraînés sur une ressource sémantique.

6.2 Description des modèles

Dans cette section, nous décrivons les 3 types de modèles sémantiques que nous proposons : les modèles basés sur la représentation, les modèles basés sur des interactions séparées et les modèles basés sur une interaction globale. Nous proposons ces types de modèles afin d'étudier quels types d'architectures sont les mieux adaptées aux réseaux de neurones sémantiques.

6.2.1 Entrée des modèles

Tous les modèles que nous proposons prennent 4 séquences de vecteurs en entrée :

1. Une séquence de vecteurs de mots (ou de n-grammes de caractères) associée à la requête
2. Une séquence de vecteurs de concepts associée à la requête
3. Une séquence de vecteurs de mots (ou de n-grammes de caractères) associée au document
4. Une séquence de vecteurs de concepts associée au document

Dans le reste de ce chapitre, le terme *"mots"* inclura également le terme *"n-gramme de caractères"*. Par exemple au lieu de dire *"vecteurs de mots ou vecteurs de n-gramme de caractères"*, nous dirons seulement *"vecteurs de mots"*.

Vecteurs de mots

Nous associons des séquences de vecteurs de mots aux requêtes et aux documents de façon standard : les requêtes et documents sont segmentés en séquence de termes, puis chacun des termes est associé à un vecteur de mot. Notons que la méthode de segmentation dépend des vecteurs de mots utilisés.

Vecteurs de concepts

Les requêtes et les documents sont annotés avec un ensemble de concepts candidats provenant de la RS. Nous adoptons la même stratégie que Shen et al. (2018) : les mots sont annotés avec leurs K-meilleurs concepts candidats afin de traiter l'ambiguïté possible de certains mots. Par exemple, la requête *"statin breast cancer survival nationwide"* est associée à la séquence de concepts suivante : [C0360714 , C0075191 , C0638232 , C0006142 , C0678222 , C0006142 , C1332438 , C0006142 , C0678222 , C0038952 , C0220921 , C0038952]. Les concepts sont ordonnés par ordre d'apparition dans le texte et par score d'annotateur pour les concepts annotés sur les mêmes mots.

6.2.2 Encodeurs de séquences

Tous les modèles que nous proposons traitent les différentes séquences de vecteurs à l'aide d'un encodeur qui renvoie une nouvelle séquence de vecteurs de même longueur. Dans ce travail de thèse, nous avons étudié l'utilisation de 4 encodeurs :

1. Encodeurs identité (noté ID) qui ne modifient pas la séquence de vecteurs
2. Encodeurs CNN qui traitent la séquence à l'aide de convolution 1D
3. Encodeurs LSTM qui traitent la séquence avec des LSTM bi-directionnel
4. Encodeurs *Transformers* (noté $Transf$)

6.2.3 Modèles basés sur la représentation

Nos modèles basés sur la représentation (notés Rep [1]), dont l'architecture est illustrée sur la Figure 6.1, effectuent la somme des éléments en sortie de chacun des encodeurs. Quatre représentations vectorielles sont donc obtenues :

1. r_t, la représentation vectorielle du texte de la requête
2. d_t, la représentation vectorielle du texte du document
3. r_c, la représentation vectorielle des concepts de la requête
4. d_c, la représentation vectorielle des concepts du document

Les modèles Rep calculent ensuite une similarité conceptuelle en utilisant le cosinus entre r_c et d_c ainsi qu'une similarité textuelle en utilisant le cosinus entre r_c et d_c. Le score de pertinence entre la requête R et le document D est calculée

1. Le modèle basé sur la représentation utilisant des encodeurs CNN sera noté Rep-CNN

à l'aide d'une combinaison linéaire entre la similarité conceptuelle et la similarité textuelle :

$$\mathrm{Rel}(R, D) = a \cos(r_t, d_t) + b \cos(r_c, d_c), \tag{6.1}$$

avec $a \in \mathbb{R}$ et $b \in \mathbb{R}$ deux paramètres appris pendant l'entraînement.

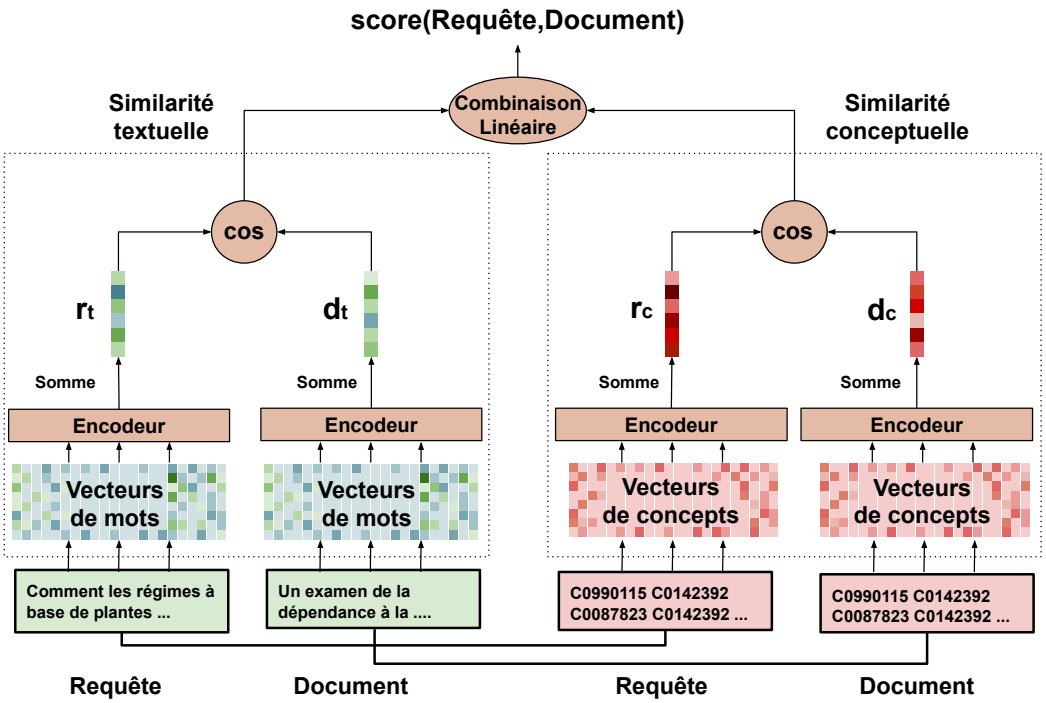

FIGURE 6.1 – Architecture des modèles NLTR sémantiques basés sur la représentation

6.2.4 Modèles basés sur l'interaction

Nous proposons deux familles de modèles basées sur l'interaction : une famille de modèles à interactions séparées (Figure 6.3) et une famille de modèles à interaction globale (Figure 6.4).

Les modèles d'interactions que nous proposons calculent tous une ou plusieurs matrices d'interactions entre les éléments en sortie des différents encodeurs. Les matrices d'interaction sont calculées en utilisant la similarité par Noyau Gaussien (*Gaussian Kernel*) comme suggéré par Pang et al. (2016a). La similarité par Noyau Gaussien entre deux vecteurs u et v est égale à :

$$e^{-||u-v||_2^2}. \tag{6.2}$$

Intuitivement, la similarité par Noyau Gaussien est mieux adaptée à la RI que d'autres similarités telles que le produit scalaire ou bien la similarité cosinus, car elle permet de mieux distinguer les signaux de correspondance exacte des autres signaux d'interaction (voir Figure 6.2).

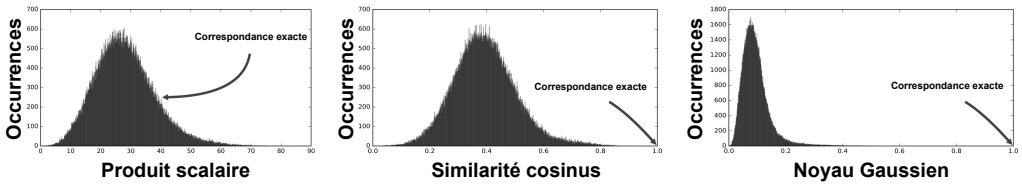

FIGURE 6.2 – Histogrammes des similarités : produit scalaire, cosinus et noyau Gaussien entre un mot fixé et l'ensemble des mots du vocabulaire de la collection TREC *Robust04*. La flèche indique le score de similarité entre deux mots identiques. Figure traduite depuis l'article de Pang et al. (2016a)

Interactions séparées

Les modèles d'interaction séparés (notés SInter) calculent deux matrices d'interaction :

1. une matrice d'interaction textuelle comparant les mots de la requête avec les mots du document.
2. une matrice d'interaction conceptuelle comparant les concepts de la requête avec les concepts du document.

Comme décrit précédemment, les interactions sont calculées avec la similarité par noyau Gaussien entre les éléments en sortie des différents encodeurs. La matrice d'interaction textuelle est calculée en comparant les sorties de l'encodeur associé aux mots de la requête avec les sorties de l'encodeur associée aux mots du document. La matrice d'interaction conceptuelle est calculée de façon similaire en comparant les sorties des encodeurs associés aux concepts (voir Figure 6.3).

Par exemple, dans le cas d'encodeurs identités, les éléments de la matrice d'interaction textuelle T sont calculés comme suit :

$$T_{i,j} = e^{-||x_{r_i}-x_{d_j}||_2^2}, \tag{6.3}$$

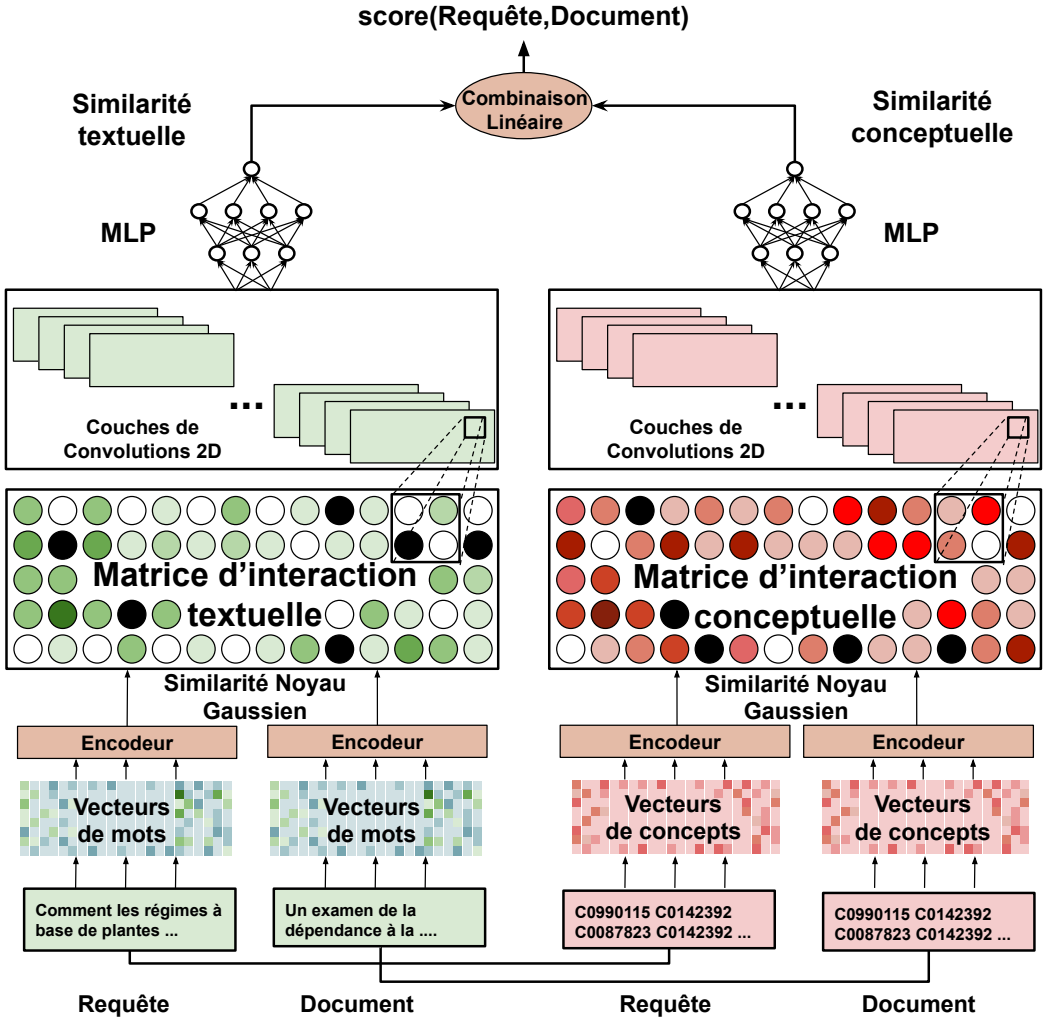

FIGURE 6.3 – Architecture des modèles NLTR sémantiques basés sur les interactions textuelle et conceptuelle

avec x_{r_i} le vecteur de mots associé au i^{ème} terme de la requête et x_{d_j} le vecteur de mot associé au j^{ème} terme du document.

Les deux matrices d'interaction sont ensuite traitées en utilisant des couches de convolution 2D et un MLP, de façon analogue de façon aux modèles ARC-II (Hu et al., 2014) et MatchPyramid (Pang et al., 2016b). Le MLP associé à la matrice d'interaction textuelle (resp. conceptuelle) retourne un score de similarité textuelle (resp. conceptuelle). De la même façon que pour les modèles basés sur la représentation que nous proposons, une combinaison linéaire entre le score de similarité textuelle et le score de similarité conceptuelle est utilisée pour calculer le score de pertinence entre la requête et le document.

Interaction globale

Les modèles d'interaction séparés décrits dans la section précédente ne considèrent que les interactions entre les mots de la requête et les mots du document ainsi que les interactions entre les concepts de la requête et les concepts du document. Les modèles d'interaction globale (notés GInter) que nous proposons prennent également en compte toutes les interactions entre mots en concepts. Les modèles GInter construisent une matrice d'interaction globale (voir Figure 6.4) composée de 4 sous-matrices contenant les interactions suivantes :

1. interactions mots requête - mots document
2. interactions mots requête - concepts documents
3. interactions mots documents - concepts requête
4. interactions concepts requête - concepts documents

Toutes les interactions sont calculées avec la similarité par noyau Gaussien. De la même façon que pour les modèles à interactions séparées, la matrice d'interaction globale est ensuite traitée par des couches de convolutions 2D suivies d'un MLP qui retourne le score final de pertinence entre la requête et le document.

6.3 Protocole expérimental

Nous évaluons les modèles NLTR sémantiques sur les collections suivantes :

— La collection médicale NFCorpus avec des concepts médicaux UMLS

— Les deux collections *Wikipédia* avec des concepts DBpedia

— Les collections TREC avec les modèles pré-entraînés sur les collections *Wikipédia* avec des concepts DBpedia

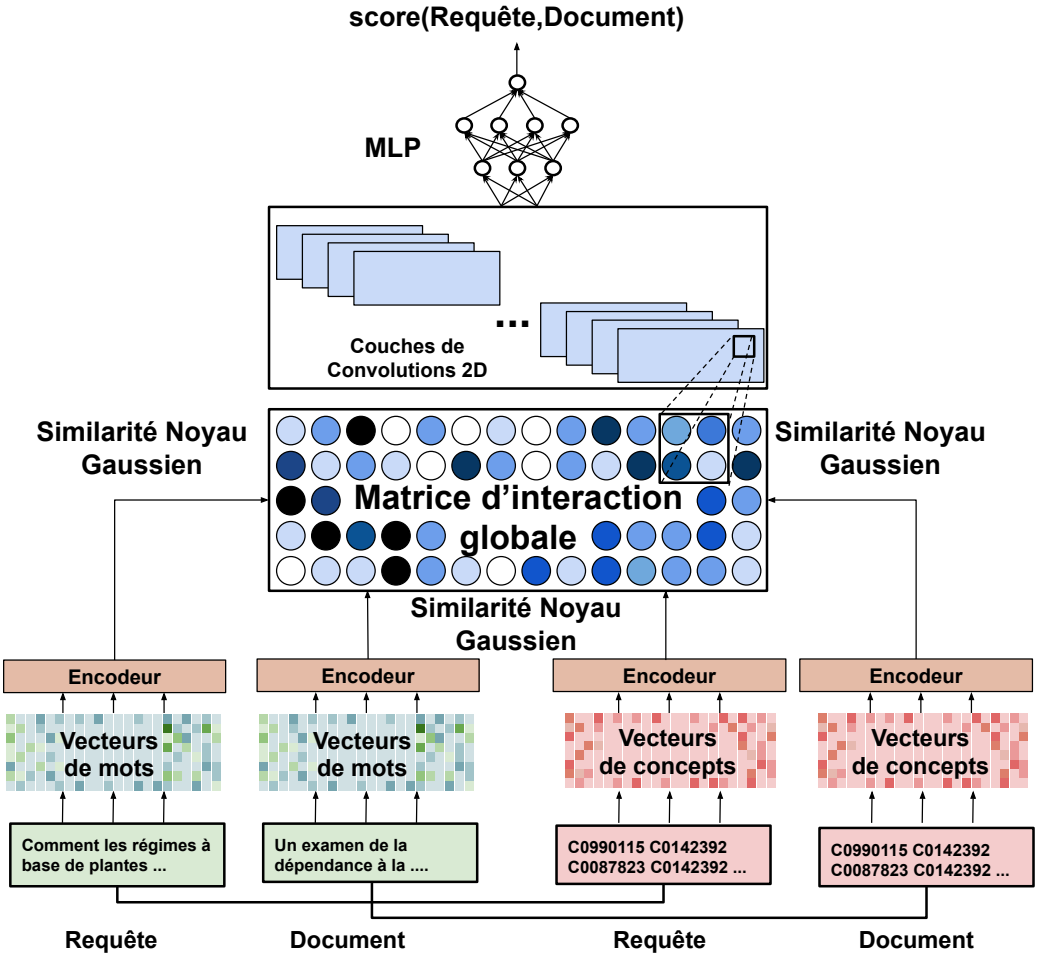

FIGURE 6.4 – Architecture des modèles NLTR sémantiques basés sur une interaction globale

Nous utilisons les mêmes modèles de référence basés sur la correspondance exacte, les mêmes modèles de référence neuronaux et les mêmes protocoles que ceux décrits dans les sections 5.4.2, 5.4.3 et 5.4.4.

6.3.1 Collection NFcorpus

Nous évaluons l'incorporation de concepts médicaux sur la collection NFCorpus (Boteva et al., 2016) : une collection en langue anglaise et libre d'accès. NFCorpus a été proposé pour développer des méthodes de type *learning-to-rank* dans le domaine médical. Elle est composée de 5276 requêtes rédigées en langage courant, tirées du site Web http://nutritionfacts.org/ et de 3633 documents composés des titres et des résumés d'articles de PubMed et de PMC rédigés dans un vocabulaire hautement technique. Des exemples de requêtes et de document pertinents associés tirés de la collection NFCorpus sont illustrés sur le Tableau 6.1.

Requête	**Document pertinent associé**
ultra-processed foods	manufactured uncertainty ...
coffee and artery function	caffeine enhances endothelial ...
the actual benefit of diet vs. drugs	... placebo study diet drug ...
a dairy-free diet may stop ...	significant dietary intake ...

TABLE 6.1 – Exemples de requêtes et d'un document pertinent associé provenant du NFCorpus. Le corpus étant déjà pré-traité, les exemples ci-dessus ne contiennent ni majuscules, ni ponctuation.

6.3.2 Ressource sémantique dans le domaine médical : UMLS

Dans le cadre de cette thèse, nous utilisons la ressource sémantique UMLS. UMLS (*Unified Medical Language System*) est une ressource du domaine médical (Bodenreider, 2004) qui est composée d'un méta-thésaurus, d'un réseau sémantique et de différents outils pour extraire des concepts depuis le texte (*MetaMap*), de gestion du méta-thésaurus (*MetamorphoSys*) et pour générer des variantes lexicales de noms de concepts (*lvg*). Dans ce travail de thèse, nous avons utilisé le méta-thésaurus UMLS et l'outil de gestion *MetamorphoSys*.

Méta-thésaurus

Le méta-thésaurus est une base de données de vocabulaire dans le domaine médical, extraite de nombreux vocabulaires sources [1]. Le méta-thésaurus est organisé en concepts, qui représentent la signification commune d'un ensemble de chaînes de caractères extraites des différents vocabulaires sources. Une structure spécifique relie les concepts à leur vocabulaire source (Voir Figure 6.5). Cette structure est composée de :

— Chaînes de caractères : les représentants des différentes formes du même concept.

— Atomes : ensemble associé à chaque chaînes de caractères indiquant le ou les vocabulaires sources d'où la chaîne de caractères est tiré.

— Termes : ensemble des chaînes de caractères qui sont des variantes lexicales les unes des autres.

Concept	Termes	Chaînes de caractères	Atomes
C0004238 Atrial Fibrillation (preferred) Atrial Fibrillations Auricular Fibrillation Auricular Fibrillations	**L0004238** Atrial Fibrillation (preferred) Atrial Fibrillations	**S0016668** Atrial Fibrillation (preferred)	**A0027665** Atrial Fibrillation (from MSH) **A0027667** Atrial Fibrillation (from PSY)
		S0016669 (plural variant) Atrial Fibrillations	**A0027668** Atrial Fibrillations (from MSH)
	L0004327 (synonym) Auricular Fibrillation Auricular Fibrillations	**S0016899** Auricular Fibrillation (preferred)	**A0027930** Auricular Fibrillation (from PSY)
		S0016900 (plural variant) Auricular Fibrillations	**A0027932** Auricular Fibrillations (from MSH)

FIGURE 6.5 – Illustration du concept C0004238 tiré du méta-thésaurus UMLS ainsi que des *terms*, *strings* et *atoms* qui lui sont associés.

En plus des Chaînes de caractères, Atomes et Termes, les concepts du méta-thésaurus sont liés entre eux par un ensemble de relations (voir Figure 6.6).

1. 60 vocabulaires sources utilisés pour la première version d'UMLS et plus de 200 pour la version de 2018

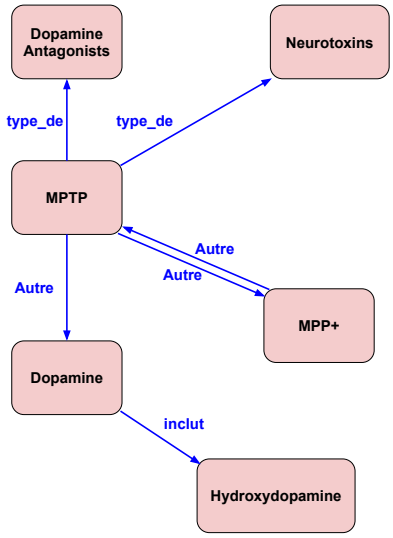

FIGURE 6.6 – Illustration d'un sous ensemble du graphe de relations entre concepts UMLS.

Nous utilisons des concepts médicaux de la version *2018AA* du Metathesaurus UMLS. Nous avons choisi UMLS principalement en raison de sa couverture : 3,67 millions de concepts tirés de 203 vocabulaires sources.

Extraction du graphe de relations

Nous avons utilisé *MetamorphoSys* pour extraire le graphe relationnel des concepts médicaux à partir d'UMLS. Nous avons éliminé les concepts qui n'appartiennent pas à un type sémantique médical (par exemple, les concepts quantitatifs). Par conséquent, le graphe relationnel que nous extrayons comprend 764 139 concepts médicaux, 26 309 368 relations et 3 types de relations : type_de, inclut et autre.

Annotation du texte

Le texte est annoté avec les concepts médicaux en utilisant *QuickUMLS* (Soldaini et Goharian, 2016) avec des valeurs de paramètres par défaut. Nous avons utilisé *QuickUMLS* au lieu de *MetaMap*, l'outil d'annotation fourni par UMLS en raison de sa vitesse d'annotation supérieur (jusqu'à 135 fois plus importante). Comme l'ont fait Shen et al. (2018), le nombre maximal de concepts candidats par

n-gramme de caractères annoté est fixé à 8.

6.3.3 Ressource sémantique dans le domaine Général

DBpedia est une ontologie construite automatiquement à partir des articles *Wikipédia* (Auer et al., 2007). DBpedia est construite en exploitant :

1. Les infoboîtes (*infoboxes*) associées aux articles contenant des informations sur le sujet de l'article
2. Le texte des articles
3. Les liens internes entre articles
4. Les relations déjà explicitées par *MediaWiki*, le moteur de wiki pour le Web utilisé, entre autres, par *Wikipédia*.

Les algorithmes d'extractions de relations utilisés pour la construction de DBpedia sont libres d'accès [1]. DBpedia est disponible en format RDF qui est composé d'un ensemble de triplets (sujet, prédicat, objet). Dans le cas de DBpedia les sujets et objets sont des concepts et les prédicats sont des relations entre concepts (*type*, *subject*, *birthPlace*, etc ...). Nous avons utilisé la version anglaise de DBpedia 2014 [2] composée de 580 millions de triplets contenants 4,5 millions de concepts. Nous annotons le texte avec des concepts de DBpedia à l'aide de *pyspotlight* [3], en utilisant les valeurs des paramètres par défaut.

6.3.4 Vecteurs de mots

Nous utilisons différents vecteurs de mots en fonction des collection sur lesquelles les modèles NLTR sémantiques sont entraînés et évalués. Sur la collection médicale NFCorpus, nous utilisons des vecteurs de mots entraînés sur une combinaison de textes de *PubMed* [4] et *PMC* [5]. Nous comparons deux méthodes de vecteurs de mots :

— *word2vec*, dont les vecteurs entraînés sur *PubMed* et *PMC*, nommé *bio-word2vec* sont disponibles à l'adresse : http://bio.nlplab.org

1. https://github.com/dbpedia/extraction-framework/wiki
2. https://wiki.dbpedia.org/Downloads2014
3. https://pypi.org/project/pyspotlight/
4. Composé de plus de 30 millions d'articles de recherche dans le domaine biomédical
5. Composé de 6.4 millions d'articles de recherche dans le domaine biomédical

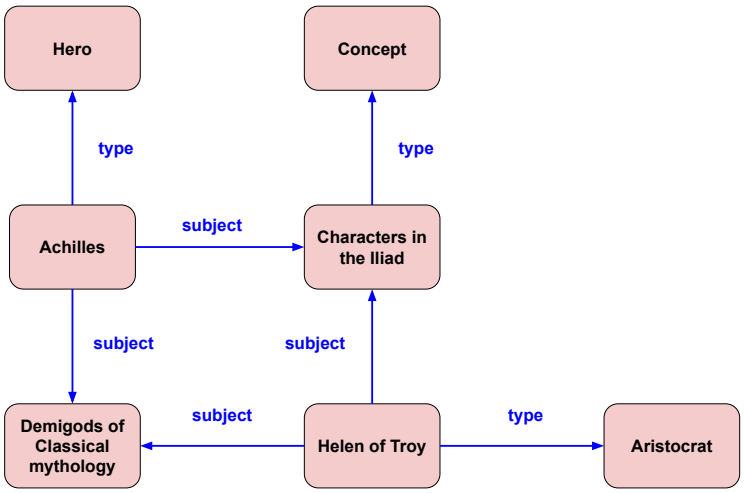

FIGURE 6.7 – Illustration d'un sous ensemble du graphe relationnel DBpedia

— BERT, dont le modèle entraîné sur *PubMed* et *PMC*, nommé BioBert est disponible à l'adresse suivante : https://github.com/dmis-lab/biobert.

Sur les collections *Wikipédia* et TREC, nous utilisons des vecteurs de mots non-spécifiques à un domaine particulier. Nous comparons deux méthodes de vecteurs de mots :

— *fast*Text, dont les vecteurs entraînés sur la version anglaise de *Common Crawl* [1] sont disponibles à cette adresse : https://fasttext.cc/docs/en/english-vectors.html.

— BERT, dont le modèle entraîné sur la version anglaise de *Wikipédia* et le jeu de données BooksCorpus (Zhu et al., 2015), nommé $\text{BERT}_{\text{BASE}}$ est disponible à l'adresse suivante : https://github.com/huggingface/transformers.

Les vecteurs *bio-word2vec* et *fast*Text sont mis à jour pendant l'entraînement des modèles NLTR sémantiques. La dernière couche des modèles $\text{BERT}_{\text{BASE}}$ et BioBert est également mise à jour lors de l'entraînement des modèles NLTR sémantiques.

6.3.5 Vecteurs de concepts

Les modèles NLTR sémantiques entraînés et évalués sur la collection médicale NFCorpus, utilisent des vecteurs de concepts de dimension 200, entraînés sur le

1. Un agrégat de pages Web composé de 600 milliards de mots

graphe relationnel UMLS. Les modèles NLTR sémantiques entraînés et évalués sur les collections *Wikipédia* et TREC utilisent des vecteurs de concepts de dimension 200, entraînés sur le graphe relationnel DBpedia. Les vecteurs de concepts sont entraînés avec l'algorithme TransE (section 3.2) et sont mis à jour pendant l'entraînement des modèles NLTR sémantiques.

6.3.6 Implémentation

Les modèles NLTR sémantiques sont implémentés avec *pytorch* [1]. Les nombres de paramètres associés aux réseaux de neurones sémantiques sont représentés sur le tableau 6.2. Puisque le nombre de paramètres d'un réseaux de neurone dépend des valeurs de ses hyperparamètres, nous reportons les nombres de paramètres associés aux valeurs des hyperparamètres donnant les meilleurs performances sur la collection NFCorpus

Modèle	Nombre de paramètres
ID-Rep	2
CNN-Rep	361k
LSTM-Rep	960k
Transf-Rep	720k
ID-SInter	34k
CNN-SInter	315k
LSTM-SInter	814k
Transf-SInter	564k
ID-GInter	22k
CNN-GInter	290k
LSTM-GInter	889k
Transf-GInter	659k

TABLE 6.2 – Nombre de paramètres des modèles NLTR sémantiques. Les nombres de paramètres reportés dans ce tableaux correspondent aux valeurs des hyperparamètres donnant les meilleurs performances sur la collection NFCorpus.

Entraînement

La méthode d'optimisation Adam (Bolukbasi et al., 2016) est utilisée avec un taux d'apprentissage égal à 0.001 sur les collection NFCorpus et *Wikipédia* et égal

1. https://pytorch.org/

à 0.0001 sur les collections TREC. Nous utilisons la fonction objectif *pairwise* de Hinge avec une marge égale à 1.

Hyperparamètres

Les valeurs des hyperparamètres renseignées ci-dessous sont choisies par recherche aléatoire sur 20 exécutions pour maximiser la ndcg@5 sur les ensembles de validations. Nous n'avons pas exploré les valeurs des hyperparamètres avec la méthode de recherche en grille en raison des ressources et du temps importants nécessaires à une exploration exhaustive.

Encodeur CNN. La taille des filtres, le nombre de filtres et le nombre de couches sont sélectionnés parmi $\{2, 3, 4, 5\}$, $\{50, 100, 200, 500\}$ et $\{1, 2, 3\}$ respectivement. Nous avons utilisé la fonction d'activation ReLU.

Encodeur LSTM. La dimension des vecteurs d'état interne [1] ainsi que le nombre de couches sont sélectionnés parmi $\{50, 100, 200, 500\}$ et $\{1, 2, 3\}$ respectivement. Nous avons utilisé les mêmes fonctions d'activations que celles décrites dans la section 3.4.2.

Encodeur transformer. Le nombre de têtes d'attention, le nombre de couches et la dimension du réseau *feed forward* sont sélectionnés parmi $\{1, 2, 5, 10\}$, $\{1, 2, 3\}$ et $\{50, 100, 200, 500\}$ respectivement. Nous avons utilisé la fonction d'activation ReLU.

Convolutions 2D. La taille des filtres, le nombre de filtres et le nombre de couches sont sélectionnés parmi $\{2 \times 2, 3 \times 3, 4 \times 4, 5 \times 5\}$, $\{50, 100, 200, 500\}$ et $\{1, 2, 3\}$ respectivement. Nous avons utilisé la fonction d'activation ReLU et des couches *max-pooling*.

MLP. Nous avons implémenté des MLP à une couche cachée. L'entrée des MLP est de même dimension que la sortie de la dernière couche de convolution 2D, la sortie du MLP est un scalaire (donc de dimension 1) et la dimension de la couche intermédiaire est sélectionnée parmi $\{50, 100, 200, 500\}$. Nous avons utilisé la fonction d'activation ReLU.

Évaluation

Comme le NFCorpus ne possède que 3633 documents, nous pouvons évaluer chaque paire (requête, document) dans un délai raisonnable afin d'éviter de dépendre d'une stratégie de reclassement. Par conséquent, nous pouvons comparer le rappel des modèles NLTR avec le rappel des modèles basés sur la correspondance exacte.

1. Les vecteurs d'états interne les vecteurs de mémoire ont la même dimension.

6.4 Résultats

6.4.1 Modèles NLTR sémantiques dans le domaine médical

Analyse des performances des modèles NLTR sémantiques

Les performances des modèles NLTR sémantiques utilisant les vecteurs de mots *bio-word2vec* sur la collection NFCorpus sont indiquées sur le Tableau 6.3. Les résultats de l'ensemble des modèles NLTR de référence et de BM25 sont indiqués sur le Tableau C.1 en annexes.

Modèle	P@5	P@10	ndcg@5	ndcg@10	rappel@1000
BM25-RM3	0.3056	0.2603	0.3664	0.3431	0.6249
CKNRM	0.3146	0.2865	0.3010^{-}	0.3090	0.8143^{+}
ID-Rep	0.0572^{-}	0.0442^{-}	0.0832^{-}	0.0547^{-}	0.2701^{-}
CNN-Rep	0.3265	0.2976^{+}	0.3362	0.3287	0.8133^{+}
LSTM-Rep	0.2762	0.2485	0.2856^{-}	0.2706^{-}	0.7986^{+}
Transf-Rep	$\mathbf{0.3554}^{+}$	$\mathbf{0.3294}^{+}$	**0.3708**	**0.3584**	$\mathbf{0.8520}^{+}$
ID-SInter	0.3466^{+}	0.3161^{+}	0.3575	0.3535	0.8464^{+}
CNN-SInter	0.3090	0.2895	0.3232^{-}	0.3079^{-}	0.8321^{+}
LSTM-SInter	0.2795	0.2589	0.2912^{-}	0.2743^{-}	0.7877^{+}
Transf-SInter	0.3300^{+}	0.3129^{+}	0.3481	0.3312	0.8322^{+}
ID-GInter	0.3290	0.3033^{+}	0.3474	0.3287	0.8305^{+}
CNN-GInter	0.2893	0.2718	0.3044^{-}	0.2871^{-}	0.8094^{+}
LSTM-GInter	0.2601^{-}	0.2417	0.2708^{-}	0.2539^{-}	0.7708^{+}
Transf-GInter	0.3114	0.2957^{+}	0.3305	0.3124^{-}	0.8114^{+}

TABLE 6.3 – Comparaison des performances de BM25-RM3, CKNRM et des modèles NLTR sémantiques avec les vecteurs de mots *word2vec* sur le NFCorpus. Les amélioration/dégradations statistiquement significatives par rapport à BM25-RM3 sont notées par (+/-) avec une p-value < 0.01.

Modèles basés sur la représentation. Le modèle Transf-Rep, surpasse les deux modèles de référence BM25-RM3 et CKNRM sur toutes les mesures. Le fait que les modèles CNN-Rep et LSTM-Rep ne parviennent pas à des niveaux de performances similaires à Transf-Rep suggère que les encodeurs *Transformers* sont mieux adaptés à la RI textuelle que les LSTM et que les couches de convolution 1D. De plus, les faibles performances du modèle ID-Rep suggèrent que pour les modèles NLTR

basés sur la représentation, il est nécessaire de traiter les vecteurs pour avoir des représentations utiles pour la RI.
Modèles basés sur l'interaction. De façon intéressante, les modèles basés sur l'interaction les plus performants sont ceux qui n'appliquent pas de transformation aux vecteurs en entrée : ID-SInter et ID-GInter. Ces observations suggèrent que les matrices d'interactions doivent être calculées directement à partir des vecteurs de mots et/ou de concepts, sans transformation. En effet, le fait de ne pas transformer les vecteurs permet aux modèles de prendre en compte de façon explicite les signaux de correspondance exacte : deux vecteurs du même mot/concepts auront une similarité égale à 1 tandis que si les vecteurs passent par une couche CNN, LSTM ou *Transformers*, alors ils ne sont pas garantis d'avoir une similarité égale à 1, et ce, même s'ils sont issus du même mot/concept. Ces résultats confirment l'importance de la prise en compte des signaux de correspondance exacte pour les modèles neuronaux basés sur l'interaction. Finalement, nous constatons que les modèles à interaction globale obtiennent de moins bonnes performances que leurs homologues à interactions séparées. Cela suggère qu'il n'est pas pertinent de comparer directement les vecteurs de mots aux vecteurs de concepts et que les interactions mots/mots et les interactions concepts/concepts doivent être traitées séparément.
Rappel des modèles NLTR. Les résultats indiquent que le gain de tous les modèles NLTR (excepté ID-Rep) en terme de rappel est significatif par rapport à BM25-RM3 (+36.3% pour Transf-Rep). Ce gain s'explique par le fait que les modèles basés sur la correspondance exacte ne peuvent récupérer que les documents qui contiennent au moins un terme de la requête alors que les modèles NLTR n'ont pas cette restriction.

Analyse des gains apportés par les concepts médicaux

Les gains de performance apportés par l'incorporation de concepts médicaux sont indiqués sur le Tableau 6.4. Nous constatons que le fait d'incorporer des vecteurs de concepts médicaux permet d'augmenter les performances des modèles NLTR sémantiques basés sur la représentation (excepté ID-Rep) et basés sur des interactions séparées. Cela n'est néanmoins pas le cas pour les modèles à interaction globale. Ces résultats suggèrent que les interactions mots/concepts ne constituent pas un signal pertinent pour la RI et que leur incorporation peuvent dégrader les performances des modèles de RI.

Comparaison des vecteurs *bio-word2vec* avec les vecteurs BioBERT

Les comparaisons des performances des modèles NLTR sémantiques utilisant des vecteurs *bio-word2vec* avec ceux utilisant les vecteurs BioBERT sont décrites

sur le Tableau 6.5. Les comparaisons sur la totalité des modèles sont décrites sur le Tableau C.2 en annexes.

Modèles basés sur la représentation. Nous constatons que pour les modèles basés sur la représentation, l'utilisation des vecteurs BioBERT entraîne de moins bonnes performances que les vecteurs *bio-word2vec*. L'exception étant le modèle ID-Rep avec BioBERT qui a des performances proches de celles de Transf-Rep avec *bio-word2vec*. Ces résultats montrent que les vecteurs BioBERT ne doivent pas être traités par une couche intermédiaire de type CNN, LSTM ou *Transformers* et doivent être utilisés directement pour calculer les représentations textuelles des requêtes et des documents. Le fait que ID-Rep_{BB} ne soit pas aussi performant que Transf-Rep_{bw} est probablement dû au fait que le NFCorpus n'a pas assez de requêtes résolues pour affiner correctement l'encodeur BERT pour la RI dans le domaine médical.

Modèles basés sur l'interaction. Les vecteur contextualisés BioBERT donnent de moins bonne performances que les vecteurs *bio-word2vec* puisque, comme décrit précédemment, ils ne permettent pas aux modèles basés sur l'interaction de prendre en compte le signal de correspondance exacte.

modèles	P@5	P@10	ndcg@5	ndcg@10	rappel@1000
ID-Rep	+1.0%	– 0.8%	+0.7%	–2.8%	–4.6%
CNN-Rep	+14.0%	+8.9%	+3.6%	+5.3%	+4.8%
LSTM-Rep	+9.8%	+8.2%	+5.8%	+7.3%	+5.3%
Transf-Rep	+10.9%	+9.5%	+7.0%	+6.3%	+4.0%
ID-SInter	+10.4%	+11.3%	+9.2%	+8.0%	+4.8%
CNN-SInter	+9.2%	+8.9%	+5.0%	+7.4%	+3.8%
LSTM-SInter	+7.7%	+5.4%	+4.8%	+4.4%	+0.8%
Transf-SInter	+9.3%	+6.2%	+2.8%	+3.1%	+3.0%
ID-GInter	+0.8%	+1.1%	– 0.3%	– 0.4%	+2.2%
CNN-GInter	+2.1%	+2.5%	– 0.9%	+0.3%	+2.2%
LSTM-GInter	+0.0%	+1.8%	+1.5%	– 0.1%	+0.3%
Transf-GInter	+0.3%	+0.0%	+0.3%	+0.1%	+1.2%

TABLE 6.4 – Gains de performance obtenus avec l'incorporation de concepts sur la collection NFCorpus.

modèles	P@5	P@10	ndcg@5	ndcg@10	rappel@1000
ID-Rep$_{bw}$	0.0572	0.0442	0.0832	0.0547	0.2701
ID-Rep$_{BB}$	0.3476	0.3013	0.3451	0.3469	0.8494
CNN-Rep$_{bw}$	0.3265	0.2976	0.3362	0.3287	0.8133
CNN-Rep$_{BB}$	0.2671	0.2379	0.2785	0.2645	0.7813
LSTM-Rep$_{bw}$	0.2762	0.2485	0.2856	0.2706	0.7986
LSTM-Rep$_{BB}$	0.2318	0.1938	0.2478	0.2272	0.7930
Transf-Rep$_{bw}$	**0.3554**	**0.3294**	**0.3708**	**0.3584**	**0.8520**
Transf-Rep$_{BB}$	0.2542	0.1982	0.2557	0.2553	0.7891
ID-SInter$_{bw}$	0.3466	0.3161	0.3575	0.3535	0.8464
ID-SInter$_{BB}$	0.1877	0.1571	0.1984	0.1864	0.7462
ID-GInter$_{bw}$	0.3290	0.3033	0.3474	0.3287	0.8305
ID-GInter$_{BB}$	0.1974	0.1570	0.2077	0.1902	0.7582

TABLE 6.5 – Comparaison de l'utilisation des vecteurs *bio-word2vec* (bw) avec les vecteurs BioBERT (BB) sur le NFCorpus.

Affinage sur des collections de RI médicales

De la même façon qu'avec les collections *Wikipédia*, nous avons utilisé les modèles neuronaux entraînés sur le NFCorpus et les avons affinés sur deux collections de RI médicales standards : Ohsumed (Hersh et al., 1994) et CLEF2014 (Kelly et al., 2014). Des expériences préliminaires ont néanmoins montré que cette approche ne permet pas de compenser la faible quantité de requêtes résolues des collections Ohsumed et CLEF 2014 (respectivement 63 et 50 requêtes résolues). Nous pensons que cela est dû au faible nombre de requêtes résolues du NFCorpus comparé à celui des collections *Wikipédia*.

6.4.2 Modèles NLTR sémantiques dans le domaine général

Analyse des performances des modèles NLTR sémantiques

Les performances des modèles NLTR sémantiques avec les vecteurs de mots *fast*Text sur les collections *Wikipédia* sont indiquées sur le Tableau 6.6. Les résultats de l'ensemble des modèles NLTR de référence et de BM25 sont indiqués sur les Tableaux C.3 et C.4 en annexes.

WikIR78k. Comme on peut le voir sur le Tableau 6.6, les modèles NLTR sémantiques atteignent des performances similaires, voire supérieures aux modèles de référence sur WikIR78k. De plus, les modèles basés sur l'interaction atteignent les

meilleures performances, ce qui confirme l'importance du signal de correspondance exacte dans le cas de requêtes courtes et bien définies. Tout comme sur le NFCorpus, le fait que le modèle à interactions séparées obtienne de meilleures performances que le modèle à interaction globale, suggère qu'il n'est pas pertinent de comparer directement les vecteurs de mots aux vecteurs de concepts et que les interaction mots/mots et les interactions concepts/concepts doivent être traitées séparément.

WikIRS78k. Nous constatons que les modèles NLTR sémantiques que nous proposons atteignent également des performances similaires, voire supérieures aux modèles de référence sur la collection WikIRS78k. Le modèle basé sur la représentation est néanmoins le plus performant sur WikIRS78k parmi les modèles NLTR sémantiques. Nous pouvons en conclure que les modèles NLTR basés sur la représentation sont mieux adaptés à gérer les requêtes longues et bruitées que les modèles basés sur l'interaction.

wikIR78k

Modèle	P@5	P@10	ndcg@5	ndcg@10	ndcg	MAP
BM25-RM3	0.2844	0.2235	0.3442	0.3238	0.3813	0.1710
DRMM	0.2941^{+}	0.2312^{+}	0.3659^{+}	0.3394^{+}	0.3861^{+}	$\mathbf{0.1773}^{+}$
DUET	0.2819	0.2243	0.3520^{+}	0.3239	0.3746^{-}	0.1656^{-}
CKNRM	0.2792	0.2268	0.3285^{-}	0.3114^{-}	0.3622^{-}	0.1599^{-}
Transf-Rep	0.2716^{-}	0.2173	0.3408	0.3173	0.3715	0.1516^{-}
ID-SInter	$\mathbf{0.3150}^{+}$	$\mathbf{0.2429}^{+}$	$\mathbf{0.3962}^{+}$	$\mathbf{0.3652}^{+}$	$\mathbf{0.4290}^{+}$	0.1769^{+}
ID-GInter	0.2874	0.2201	0.3601^{+}	0.3226	0.3841	0.1581

wikIRS78k

Modèle	P@5	P@10	ndcg@5	ndcg@10	ndcg	MAP
BM25-RM3	0.2237	0.1698	0.3015	0.2733	0.3158	0.1216
DRMM	0.2486^{+}	0.1867^{+}	0.3306^{+}	0.2991^{+}	0.3288^{+}	0.1361^{+}
DUET	0.2601^{+}	0.1998^{+}	0.3318^{+}	0.3051^{+}	0.3281^{+}	0.1367^{+}
CKNRM	0.2779^{+}	0.2148^{+}	0.3361^{+}	0.3134^{+}	0.3369^{+}	0.1465^{+}
Transf-Rep	$\mathbf{0.2934}^{+}$	$\mathbf{0.2287}^{+}$	0.3495^{+}	$\mathbf{0.3332}^{+}$	0.3356^{+}	$\mathbf{0.1548}^{+}$
ID-SInter	0.2657^{+}	0.2093^{+}	$\mathbf{0.3570}^{+}$	0.3281^{+}	$\mathbf{0.3492}^{+}$	0.1474^{+}
ID-GInter	0.2407^{+}	0.1828^{+}	0.3228^{+}	0.2871^{+}	0.3296	0.1323

TABLE 6.6 – Comparaison des performances des modèles NLTR sémantiques sur wikIR78k et wikIRS78k. Les amélioration/dégradations statistiquement significatives par rapport à BM25-RM3 sont notées par (+/-) avec une p-value < 0.01.

Analyse des gains apportés par les concepts

Les gains de performance apportés par l'incorporation de concepts sont indiqués sur le Tableau 6.7. Les résultats de l'ensemble des modèles NLTR sémantiques sont indiqués sur les Tableaux C.5 et C.6 en annexes. Nous constatons que l'incorporation de concepts DBpedia a un faible effet sur les performances des modèles neuronaux que nous avons développés. Nous en concluons que : **(1)** notre utilisation des vecteurs de concepts n'apporte pas d'information utile aux modèles neuronaux pour la RI sur nos collections *Wikipédia* ; **(2)** les performances supérieures comparées aux modèles NLTR de référence ne sont donc pas dues aux vecteurs de concept, mais probablement à l'utilisation d'encodeur *Transformers* pour le modèle basé sur la représentation et à la similarité par noyau Gaussien pour les modèles basés sur l'interaction puisqu'il s'agit d'attributs spécifiques qui ne sont pas présents dans les modèles NLTR de référence.

wikIR78k

Modèle	P@5	P@10	ndcg@5	ndcg@10	ndcg	MAP
Transf-Rep	- 0.4%	+ 0.1%	+ 0.1%	– 0.6%	+ 0.9%	+ 0.2%
ID-SInter	+ 0.1%	+ 1.0%	– 0.1%	– 0.1%	– 0.7%	+ 0.7%
ID-GInter	+ 0.6%	– 0.3%	+ 0.0%	+ 0.0%	+ 0.4%	+ 1.2%

wikIRS78k

Modèle	P@5	P@10	ndcg@5	ndcg@10	ndcg	MAP
Transf-Rep	+ 0.8%	+ 0.6%	+ 0.1%	+ 0.1%	+ 0.7%	+ 0.4%
ID-SInter	– 0.7%	+ 0.2%	– 0.2%	+ 0.4%	+ 0.3%	+ 0.0%
ID-GInter	– 0.2%	+ 0.1%	+ 0.0%	+ 0.3%	+ 0.1%	+ 0.5%

TABLE 6.7 – Gains de performance obtenus avec l'incorporation de concepts sur les WikIR78k et WikIRS78k.

Comparaison des vecteurs *fast*Text avec les vecteurs BERT

Les comparaisons des performances des modèles NLTR sémantiques utilisant des vecteurs *fast*Text avec ceux utilisant les vecteurs BERT sont décrites sur le Tableau 6.8. Les comparaisons sur la totalité des modèles sont décrites sur les Tableaux C.7 et C.8 en annexes. De la même façon que sur le NFCorpus, nous constatons que les vecteurs contextualisés BERT ne sont pas adaptés aux modèles basés sur l'interaction et que afin d'en tirer les meilleures performances avec des modèles basés sur la représentation, il faut utiliser les vecteurs BERT directement sans les

traiter par un encodeur supplémentaire. Nous pouvons également constater que le modèle ID-Rep$_{\text{BERT}}$ atteint les meilleurs performances sur wikIRS78k. Cela signifie que : **(1)** contrairement au NFCorpus, nos collections *Wikipédia* ont assez de requêtes résolues pour tirer avantage des vecteurs contextualisés de BERT ; **(2)** les vecteurs de mots contextualisés permettent aux modèles NLTR d'être plus robustes aux requêtes bruitées.

wikIR78k

modèles	P@5	P@10	ndcg@5	ndcg@10	ndcg	MAP
ID-Rep$_{\text{fT}}$	0.1342	0.0978	0.1546	0.1479	0.1720	0.0721
ID-Rep$_{\text{BERT}}$	0.3070	0.2410	0.3790	0.3518	0.3965	**0.1804**
Transf-Rep$_{\text{fT}}$	0.2716	0.2173	0.3408	0.3173	0.3715	0.1516
Transf-Rep$_{\text{BERT}}$	0.2259	0.1786	0.2696	0.2528	0.3004	0.1234
ID-SInter$_{\text{fT}}$	**0.3150**	**0.2429**	**0.3962**	**0.3652**	**0.4290**	0.1769
ID-SInter$_{\text{BERT}}$	0.1820	0.1423	0.2404	0.2394	0.2575	0.1029
ID-GInter$_{\text{fT}}$	0.2874	0.2201	0.3601	0.3226	0.3841	0.1581
ID-GInter$_{\text{BERT}}$	0.1891	0.1612	0.2342	0.2113	0.2465	0.1001

wikIRS78k

modèles	P@5	P@10	ndcg@5	ndcg@10	ndcg	MAP
ID-Rep$_{\text{fT}}$	0.1281	0.0761	0.1257	0.1270	0.1513	0.0605
ID-Rep$_{\text{BERT}}$	**0.3089**	**0.2437**	**0.3655**	**0.3449**	**0.3651**	**0.1635**
Transf-Rep$_{\text{fT}}$	0.2934	0.2287	0.3495	0.3332	0.3356	0.1548
Transf-Rep$_{\text{BERT}}$	0.2643	0.2037	0.3114	0.2999	0.3151	0.1382
ID-SInter$_{\text{fT}}$	0.2657	0.2093	0.3570	0.3281	0.3492	0.1474
ID-SInter$_{\text{BERT}}$	0.2396	0.1992	0.3255	0.3033	0.3284	0.1427
ID-GInter$_{\text{fT}}$	0.2407	0.1828	0.3228	0.2871	0.3296	0.1323
ID-GInter$_{\text{BERT}}$	0.1968	0.1473	0.2586	0.2341	0.2718	0.1017

TABLE 6.8 – Comparaison de l'utilisation des vecteurs *fast*Text (fT) avec les vecteurs BERT sur WikIR78k et WikIRS78k.

Bien que l'utilisation de vecteurs contextualisés entraîne des gains de performances, ces derniers ne sont pas aussi importants que ceux que l'on peut obtenir dans d'autres domaines du TALN (Devlin et al., 2019) et sur d'autres collections de RI (Dai et Callan, 2019). Nous pensons que cela est principalement dû à deux facteurs qui sont à explorer dans des travaux futurs :

1. **L'absence de signal de correspondance exacte.** Nous n'avons pas utilisé

les vecteurs contextualisés en combinaison avec un signal de correspondance exacte de la même façon que le modèle DUET.

2. **L'absence de prise en compte des passages.** Nous n'avons pas séparé les documents en ensemble de passages et encodé chacun des passages à l'aide de vecteurs contextualisés. En effet, ces derniers ne sont pas développés pour encoder de longues séquences (maximum de 512 pour BERT) par conséquent, il vaudrait mieux utiliser les vecteurs contextualisés pour l'encodage de passages plutôt que pour l'encodage de documents (Dai et Callan, 2019; Boualili et al., 2020).

Affinage sur des collections TREC

Nous avons également affiné sur les collections TREC les modèles NLTR sémantiques pré-entraînés sur les collections *Wikipédia* dont les performances sont indiquées sur le Tableau 6.9. Les résultats de l'ensemble des modèles NLTR de référence et de BM25 sont indiqués sur les Tableaux C.9 et C.10 et C.11 en annexes. Nous constatons que la stratégie de pré-entraînement et d'affinage fonctionne également sur les modèles NLTR que nous avons développés. De la même façon que pour les modèles NLTR de référence, sans pré-entraînement, les modèles NLTR sémantiques n'atteignent pas les mêmes performances que BM25 et BM25-RM3. Nous constatons néanmoins que le modèle basé sur l'interaction ID-Sinter, est proche des performances de BM25 et BM25-RM3 sur les collection LA et FT91-94 malgré le manque de requêtes résolues. Cela est probablement dû au fait qu'il n'y a pas besoin de beaucoup de données d'entraînement pour exploiter le signal de correspondance exacte pour la RI. De la même façon que pour les modèles NLTR de référence, le fait de pré-entraîner sur wikIRS78k plutôt que sur wikIR78k permet d'avoir des modèles plus robustes et donc plus performants. Le fait que ID-$\text{Rep}_{\text{BERT}_{\text{WikIRS}}}$ obtienne les meilleures performances sur toutes les mesures nous confirme que les vecteurs contextualisés permettent d'obtenir de bonnes représentations de requêtes et de documents pour la RI.

	AP88-89			**LA**			**FT91-94**		
Modèle	P@5	ndcg@5	MAP	P@5	ndcg@5	MAP	P@5	ndcg@5	MAP
BM25	0.4573	0.4470	0.2755	0.3093	0.3498	0.2221	0.3330	0.3531	0.2351
BM25-RM3	0.4720	0.4599	0.2908	0.3240	0.3626	0.2357	0.3473	0.3667	0.2513
$\text{DUET}_{\text{WikIRS}}$	0.5332^{+}	0.5461^{+}	0.3551^{+}	0.3863^{+}	0.3965^{+}	0.2743^{+}	0.3511	0.3719	0.2704
$\text{CKNRM}_{\text{WikIRS}}$	0.5450^{+}	0.5507^{+}	0.3483^{+}	0.4021^{+}	0.4105^{+}	0.2824^{+}	0.4243^{+}	0.4431^{+}	0.2898^{+}
Transf-Rep	0.1715^{-}	0.1747^{-}	0.1120^{-}	0.1138^{-}	0.1172^{-}	0.0761^{-}	0.1090^{-}	0.1229^{-}	0.0224^{-}
ID-Sinter	0.4072^{-}	0.3941^{-}	0.2412^{-}	0.2930^{-}	0.3327^{-}	0.1981^{-}	0.3160^{-}	0.3285^{-}	0.2266^{-}
$\text{Transf-Rep}_{\text{WikIR}}$	0.4649	0.4518	0.2820	0.3135	0.3516	0.2365	0.3390	0.3631	0.2444
$\text{ID-Sinter}_{\text{WikIR}}$	0.4863	0.4613	0.3073	0.3410	0.3853	0.2374	0.3575	0.3791	0.2675
$\text{ID-Rep}_{\text{BERT}_{\text{WikIRS}}}$	$\mathbf{0.5727}^{+}$	$\mathbf{0.5804}^{+}$	$\mathbf{0.3632}^{+}$	$\mathbf{0.4231}^{+}$	$\mathbf{0.4311}^{+}$	$\mathbf{0.2911}^{+}$	$\mathbf{0.4444}^{+}$	$\mathbf{0.4546}^{+}$	$\mathbf{0.3029}^{+}$
$\text{Transf-Rep}_{\text{WikIRS}}$	0.5619^{+}	0.5692^{+}	0.3522^{+}	0.4121^{+}	0.4192^{+}	0.2800^{+}	0.4333^{+}	0.4448^{+}	0.2928^{+}
$\text{ID-Sinter}_{\text{WikIRS}}$	0.5089^{+}	0.4768	0.3188	0.3469	0.4073^{+}	0.2498	0.3744	0.3840	0.2789

TABLE 6.9 – Comparaisons des performances sur les collections TREC de nos modèles pré-entraînés sur wikIR78k et wikIRS78k avec des modèles de référence. À l'exception du modèle $\text{ID-Rep}_{\text{BERT}_{\text{WikIRS}}}$ qui utilise les vecteurs de mots BERT, les résultats reportés sont ceux des modèles utilisant les vecteurs *fast*Text. Les amélioration/dégradations statistiquement significatives par rapport à BM25-RM3 sont notées par (+/-) avec une p-value < 0.01.

6.5 Conclusions

Dans ce chapitre, nous proposons des modèle NLTR sémantiques pour la RI utilisant à la fois des vecteurs de mots et des vecteurs de concepts. Nous avons proposé 3 familles de modèles : les modèles Rep calculant des représentations basées sur le texte et sur les concepts, les modèles SInter calculant une matrice d'interaction textuelle et une matrice d'interaction conceptuelle ; les modèles GInter calculant une matrice d'interaction globale.

Nous avons utilisé la collection NFCorpus pour évaluer les modèles NLTR sémantiques pour la RI dans le domaine médical et les collections *Wikipédia* et TREC pour évaluer les modèles NLTR sémantiques pour la RI dans le domaine général. Nous avons empiriquement montré que notre utilisation des vecteurs de concepts est utile pour la RI dans le domaine médical, mais pas pour la RI dans le domaine général. Nous montrons également que les encodeurs *Transformers* et les vecteurs contextualisés BERT permettent de surpasser les modèles NLTR de référence pour la RI.

Chapitre 7

Réseaux de Neurones Efficients pour la Recherche d'Information

7.1 Introduction

L'indexation de documents en Recherche d'Information (RI) consiste généralement à associer chaque document d'une collection à un ensemble de termes pondérés reflétant leur contenu informatif. Les modèles traditionnels de RI associent à tous les termes une valeur de discrimination (TDV de l'anglais *term discrimination value*) dont le but est de refléter l'utilité d'un terme afin de discriminer les documents (Salton, 1975). Par exemple le terme "*le*" est considéré comme peu discriminant puisque sa présence dans un document ne nous renseigne pas sur son contenu informatif et par conséquent ne permet pas de distinguer un document pertinent d'un document non-pertinent pour une requête donnée.

Cependant, les systèmes de RI utilisent peu les TDV lors de l'indexation. La seule exception est le filtrage par un antidictionnaire. Cette méthode suppose que les mots vides ont une TDV nulle et les supprime donc des représentations des documents indexés. Ce processus conduit à des représentations de documents plus discriminantes et permet également d'accélérer le processus de recherche lors de l'utilisation d'un index inversé. En effet, puisque l'utilisation d'un index inversé permet d'avoir rapidement accès aux documents contentant les termes d'une requête donnée, le fait de supprimer les mots vides de l'index inversé réduit le nombre de documents considérés par le système et ainsi accélère la recherche. Cette accélération est d'autant plus importante que certains mots vides sont présents dans la majorité des documents et par conséquent le fait de les supprimer permet de grandement réduire le nombre de documents considérés par le système.

Plusieurs méthodes ont été proposées pour calculer les TDV, telles que l'utilisation de la densité de l'espace vectoriel des documents (Salton et al., 1975) ou le coefficient de couverture des documents (Can et Ozkarahan, 1987). Aujourd'hui, les approches les plus courantes dans les modèles de RI basés sur la correspondance exacte calculent des TDV en utilisant soit la fréquence inverse des documents (*idf*) (Robertson et Zaragoza, 2009) soit une méthode de lissage telle que le lissage de Dirichlet (Zhai et Lafferty, 2001). Récemment, Roy et al. (2019) ont proposé de sélectionner des termes discriminants pour améliorer les méthodes d'expansion de requêtes basées sur une boucle de rétro-pertinence en aveugle (*pseudo relevance feedback*). Toutefois, ces approches n'utilisent les TDV que lors de la recherche et non pendant l'indexation.

En nous inspirant du filtrage par un antidictionnaire, nous suggérons de supprimer les termes non-discriminants au moment de l'indexation afin d'accélérer le processus de recherche sans détériorer la qualité du classement des documents. Nous proposons d'apprendre les TDV en utilisant des connaissances *a priori* sous forme de vecteurs de mots. Afin d'avoir des TDV adaptées aux modèles de RI basés sur la correspondance exacte, nous optimisons un modèle NLTR approximant le score des modèles basés sur la correspondance exacte. Cependant, certaines composantes de ces modèles telles que le nombre d'occurrences des termes (tf) ou la fréquence inverse des documents (idf) ne sont pas différentiables dans le cadre où les réseaux neuronaux sont couramment utilisés (séquences de vecteurs de mots traitées par des CNN, RNN ou des couches Transformers). Cette non-différentiabilité rend impossible l'utilisation des méthodes d'optimisation basées sur la descente de gradient requises par les réseaux de neurones.

Pour pallier ce problème, nous proposons :

1. De redéfinir les modèles de RI basés sur la correspondance exacte afin de faire en sorte que toutes les opérations qu'ils utilisent soient différentiables par rapport aux tf de la requête et du document
2. De multiplier les tf par les TDV calculées à l'aide d'un modèle NLTR

Par conséquent, il sera possible de propager le gradient de la fonction de perte jusqu'au modèle NLTR utilisé pour calculer les TDV. Nous proposons d'utiliser un modèle NLTR à une couche caché afin qu'il ait peu de paramètres et qu'il n'ait donc pas besoin de grandes quantités de requêtes résolues pour avoir des résultats similaires, voire meilleurs que les modèles de RI basés sur la correspondance exacte. En outre, nous proposons de supprimer de l'index inversé les *posting lists* associées aux termes ayant une TDV nulle afin d'augmenter la vitesse de recherche et de réduire la taille de l'index inversé en mémoire.

À notre connaissance, notre approche est la première à proposer d'approximer

de façon différentiable des modèles de RI basés sur la correspondance exacte à l'aide de réseaux de neurones (Frej et al., 2020a).

7.2 Apprentissage de valeur de discrimination

Pour apprendre des TDV adaptées aux modèles de RI basés sur la correspondance exacte en utilisant un modèle NLTR, nous proposons la stratégie suivante :

1. Rendre les modèles de RI basés sur la correspondance exacte compatibles avec des modèles NLTR en utilisant des opérations matricielles différentiables par rapport aux éléments de l'index inversé (Section 7.2.1) ;
2. Introduire un réseau de neurones pour calculer les TDV ainsi qu'une méthode pour inclure les TDV dans l'index inversé (Section 7.2.2) ;
3. Utiliser les modèles différentiables proposés dans la section 7.2.1 pour apprendre des TDV adaptées aux modèles de RI basés sur la correspondance exacte en optimisant un modèle NLTR de façon supervisée (Section 7.2.3) ;

7.2.1 RI traditionnelle différentiable

Toutes les opérations utilisées par les fonctions de classement de RI traditionnelles peuvent être exprimées à partir d'éléments de l'index inversé. Étant donné un vocabulaire V et une collection C, l'index inversé peut être représenté sous la forme d'une matrice creuse $S \in \mathbb{R}^{|V| \times |C|}$. Chaque élément de S correspond au tf d'un terme $t \in V$ par rapport à un document $d \in C$: $S_{t,d} = \text{tf}_{t,d}$. Les colonnes de S (dénotées $S_{:,d}$) correspondent aux représentations des documents en C en BoW et les lignes de S (dénotées $S_{t,:}$) correspondent aux *posting lists* des termes de V. $R \in \mathbb{R}^{|V|}$ dénote la représentation en BoW d'une requête r.

TF-IDF

En utilisant des opérations matricielles sur S, la fonction de classement TF-IDF entre une requête r et un document d peut être formulée comme suit :

$$\text{TF-IDF}(r, d) = \sum_{t \in r} \text{tf}_{t,d}\, \text{idf}_t = R^{\intercal} \cdot (S_{:,d} \odot \text{IDF}), \tag{7.1}$$

avec $\odot$ le produit élément par élément (ou produit de Hadamard) et $\text{IDF} \in \mathbb{R}^{|V|}$ le vecteur contenant l'idf de tous les termes du vocabulaire V. Les idf peuvent être

dérivés de S en utilisant la norme ℓ_0 pour calculer les fréquences de documents (df) :

$$\mathrm{idf}_t = \log \frac{|C| + 1}{\mathrm{df}_t} = \log \frac{|C| + 1}{\ell_0(S_{t,:})}. \tag{7.2}$$

Comme mentionné précédemment, pour pouvoir adapter les TDV aux modèles de RI basés sur la correspondance exacte, ces derniers doivent utiliser des opérations différentiables par rapport aux éléments de S. Cependant, la norme ℓ_0 n'est pas différentiable. Par conséquent, nous proposons de redéfinir l'idf en utilisant la norme ℓ_1 qui est une bonne approximation de la norme ℓ_0 (Ramirez et al., 2013). En remplaçant ℓ_0 par ℓ_1, l'idf obtenu est négatif pour les termes tels que $\ell_0(S_{t,:}) > |C| + 1$ ce qui viole la contrainte de fréquence des termes, une propriété désirable des fonctions de classement de RI (Fang et al., 2004). Pour garantir des idf positifs, nous proposons une normalisation maximum :

$$\widetilde{\mathrm{idf}}_t = \log \frac{\max_{\{t' \in V\}} \ell_1(S_{t',:}) + 1}{\ell_1(S_{t,:})}. \tag{7.3}$$

En utilisant $\widetilde{\mathrm{idf}}_t$, nous avons l'approximation différentiable suivante de la formule TF-IDF :

$$\widetilde{\text{TF-IDF}}(r, d) = R^\intercal \cdot \left(S_{:,d} \odot \widetilde{\mathrm{IDF}} \right). \tag{7.4}$$

avec $\widetilde{\mathrm{IDF}} \in \mathbb{R}^{|V|}$ le vecteur contenant l'$\widetilde{\mathrm{idf}}$ de tous les termes de V.

BM25

Considérons la formule de classement BM25 (Robertson et Zaragoza, 2009) :

$$\mathrm{BM25}(r, d) = \sum_{t \in r} \mathrm{idf}_t \frac{\mathrm{tf}_{t,d}(k_1 + 1)}{\mathrm{tf}_{t,d} + k_1 \left(1 - b + b \frac{|d|}{avgdl}\right)}, \tag{7.5}$$

avec k_1 et b des paramètres de BM25 et $avgdl$ la longueur moyenne des documents de C. En utilisant la même stratégie que précédemment nous pouvons également définir une approximation différentiable de la formule de classement de BM25 en utilisant $\widetilde{\mathrm{IDF}}$:

$$\widetilde{\mathrm{BM25}}(r,d) = R^{\intercal} \cdot \left[\widetilde{\mathrm{IDF}} \odot \left(S_{:,d} \left(k_1 + 1 \right) \right) ./ \left(S_{:,d} + k_1 \left(1 - b + b \frac{|d|}{avgdl} \right) \mathbb{1}_{|V|} \right) \right], \tag{7.6}$$

avec ./ la division éléments par éléments (ou division de Hadamard) et $\mathbb{1}_{|V|}$ est un vecteur de dimension $|V|$ avec tous ses éléments égaux à un. $|d|$ et *avgdl* sont tous deux différentiables par rapport aux éléments de S :

$$|d| = \ell_1(S_{:,d}), \tag{7.7}$$

$$avgdl = \sum_{d \in C} \ell_1(S_{:,d}) / |C|. \tag{7.8}$$

Modèle de langue de Jelinek-Mercer

Nous proposons la formulation différentiable du modèle de langue avec lissage de Jelinek-Mercer suivante :

$$\begin{aligned} \mathrm{JM}(r,d) &= \sum_{t \in r} \log \left(1 + \frac{\lambda \mathbb{P}(t|\theta_d)}{(1-\lambda)\mathbb{P}(t|\theta_C)} \right) + |r| \log \lambda \\ &= R^{\intercal} \cdot \left[\log \left(\mathbb{1}_{|V|} + \frac{\lambda}{(1-\lambda)} \left(\frac{S_{:,d}}{|d|} ./ P_C \right) \right) + |r| \log(\lambda) \mathbb{1}_{|V|} \right], \end{aligned} \tag{7.9}$$

avec $P_C \in \mathbb{R}^{|V|}$ le vecteur contenant les probabilités de présence de chacun des termes du vocabulaire dans la collection :

$$\forall t \in V, P_{C_t} = \mathbb{P}(t|\theta_C) = \sum_{d \in C} S_{t,d} / \sum_{t' \in V} \sum_{d \in C} S_{t',d}. \tag{7.10}$$

P_C est bien différentiable par rapport aux éléments de S.

Modèle de langue avec lissage de Dirichlet

Nous proposons également une formulation différentiable du modèle de langue avec lissage de Dirichlet :

$$
\begin{aligned}
\mathrm{Dir}(r,d) &= \sum_{t\in r} \log\left(1 + \frac{\mathrm{tf}_{t,d}}{\mu \mathbb{P}(t|\theta_C)}\right) + |r| \log \frac{\mu}{|d|+\mu} \\
&= R^\intercal \cdot \left[\log\left(\mathbb{1}_{|V|} + (S_{:,d}./(\mu P_C))\right) + |r| \log\left(\frac{\mu}{|d|+\mu}\right) \mathbb{1}_{|V|}\right],
\end{aligned}
\tag{7.11}
$$

7.2.2 Réseau de neurones peu profond pour l'apprentissage des TDV

Nous proposons de calculer les TDV en utilisant un réseau de neurones peu profond composé d'une seule couche linéaire et de l'opération non-linéaire ReLU : $\mathrm{tdv}_t = \mathrm{ReLU}(x_t^\intercal \cdot w + b_{nn}) = \max(0, x_t^\intercal \cdot w + b_{nn})$ où x_t est le vecteur de mot associé au terme t, et $w \in \mathbb{R}^d$ et $b_{nn} \in \mathbb{R}$ sont des paramètres du réseau de neurones. Nous n'utilisons qu'une couche cachée afin de limiter le nombre de paramètres et donc de limiter le nombre de requêtes résolues nécessaire pour l'entraînement. Nous utilisons la fonction d'activation ReLU pour nous assurer que les TDV soient positives (car les TDV négatives peuvent violer la contrainte de fréquence des termes) ainsi que pour pouvoir avoir des termes avec une TDV nulle que nous pourrons retirer de l'index inversé. Nous redéfinissons l'index inversé S en utilisant les TDV de la manière suivante :

$$
S'_{t,d} = \mathrm{tf}_{t,d}\ \mathrm{tdv}_t = \mathrm{tf}_{t,d}\ \mathrm{ReLU}(x_t^\intercal \cdot w + b_{nn}). \tag{7.12}
$$

Avec cette définition, nous nous assurons que si un terme t a une valeur de discrimination nulle ($\mathrm{tdv}_t = 0$), la ligne dans S associée à t sera remplie de zéros et par conséquent sa *posting list* sera vide et pourra être retirée de l'index inversé.

7.2.3 Apprendre les TDV avec de la RI différentiable

Pour apprendre des TDV qui optimisent le score des modèles de RI basés sur la correspondance exacte, il suffit de remplacer S dans les équations (7.4), (7.6), (7.9) et (7.11) par S' :

$$\text{TDV-TF-IDF}(r,d) = R^{\intercal} \cdot \left(S'_{:,d} \odot \widetilde{\text{IDF}'} \right); \tag{7.13}$$

$$\begin{aligned}\text{TDV-BM25}(r,d) &= R^{\intercal} \cdot \widetilde{\text{IDF}'} \odot \left(S'_{:,d} \left(k_1 + 1\right)\right) \\ &\quad ./ \left(S'_{:,d} + k_1 \left(1 - b + b\frac{|d|'}{avgdl'}\right) \mathbb{1}_{|V|} \right);\end{aligned} \tag{7.14}$$

$$\text{TDV-JM}(r,d) = R^{\intercal} \cdot \left[\log \left(\mathbb{1}_{|V|} + \frac{\lambda}{(1-\lambda)} \left(\frac{S'_{:,d}}{|d|'} ./ P'_C \right) \right) \right]; \tag{7.15}$$

$$\text{TDV-Dir}(r,d) = R^{\intercal} \cdot \left[\log \left(\mathbb{1}_{|V|} + \left(S'_{:,d} ./ (\mu P'_C) \right) \right) + |r| \log \left(\frac{\mu}{|d|' + \mu} \right) \mathbb{1}_{|V|} \right]; \tag{7.16}$$

où $\widetilde{\text{IDF}'}$, $|d|'$, $avgdl'$ et P'_C désignent respectivement $\widetilde{\text{IDF}}$, $|d|$, $avgdl$ et P_C calculés avec S' au lieu de S. Les scores calculés par ces fonctions sont différentiables par rapport aux paramètres w et b_{nn} (voir figure 7.1). Par conséquent, nous pouvons utiliser des méthodes d'optimisation basées sur la descente de gradient pour mettre à jour w et b, ainsi que les paramètres spécifiques aux modèles k_1, b, λ et μ.

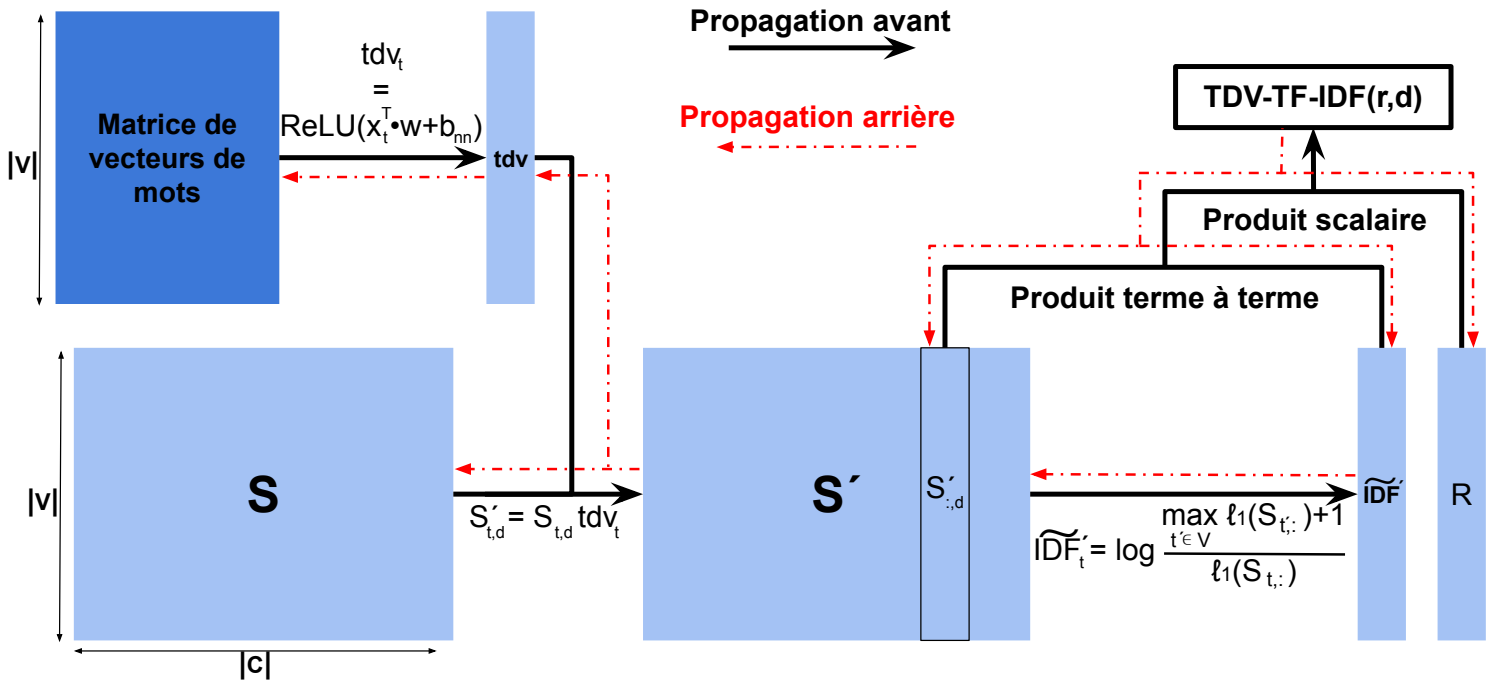

FIGURE 7.1 – Architecture du modèle TDV-TF-IDF. Toutes les opérations sont différentiables et les gradients peuvent être rétro-propagés du score finale jusqu'à w et b.

7.3 Protocole expérimental

Nous évaluons les modèles NLTR efficients sur les collections *Wikipédia* et TREC (avec et sans affinage) selon 3 critères :

1. Les mesures d'évaluation traditionnelles de RI
2. Le temps moyen de recherche pour une requête
3. La réduction de l'empreinte mémoire de l'index inversé

Nous utilisons les mêmes protocoles que ceux décrits dans les sections 5.4.2, 5.4.3 et 5.4.4.

7.3.1 Vecteurs de mots

Nous utilisons les vecteurs *fast*Text de dimension 300 entraînés sur la version anglaise de *Common Crawl* disponibles à cette adresse : https://fasttext.cc/docs/en/english-vectors.html. Les vecteurs de mots ne sont pas mis à jour pour réduire le nombre de paramètres des modèles et donc réduire la quantité de requêtes résolues pour l'entraînement.

7.3.2 Nombre de paramètres

Puisque nous utilisons des vecteurs de mots de dimension 300, le nombre de paramètres associé au réseau de neurones à une couche calculant les TDV est égal à 301 (w de même dimension que les vecteurs de mots et b_{nn} de dimension 1). Le nombre de paramètres de chacun des modèles NLTR efficients est illustré sur le tableau 7.1.

Modèle	Paramètres	Nombre total de paramètres
TDV-TF-IDF	w, b_{nn}	301
TDV-JM	w, b_{nn}, λ	302
TDV-Dir	w, b_{nn}, μ	302
TDV-BM25	w, b_{nn}, k_1, b	303

TABLE 7.1 – Nombre de paramètres des modèles NLTR efficients.

7.3.3 Implémentation

Les modèles neuronaux sont implémentés avec *Tensorflow* [1]. Pour des raisons d'efficience, et pour réduire l'utilisation de la mémoire, nous avons utilisé des représentations tensorielles creuses (*sparse tensors*).

Entraînement

La méthode d'optimisation Adam (Bolukbasi et al., 2016) est utilisée avec un taux d'apprentissage égal à 0.001 sur les collections *Wikipédia* et TREC sans affinage et avec un taux d'apprentissage égal à 0.0001 sur les collections TREC avec affinage. Nous utilisons la fonction objectif *pairwise* de Hinge avec une marge égale à 1. Afin de favoriser que les fonctions de classement produisant des termes avec des valeurs de discrimination zéro, nous ajoutons à la fonction objectif un objectif de *sparsité*. Pour ce faire, nous minimisons la norme ℓ_1 de la représentation des documents comme suggéré par Zamani et al. (2018). La fonction objectif que nous utilisons est définie comme suit :

$$(1-\lambda)\mathcal{L}_{\text{Hinge}}(f, r, d^+, d^-) + \lambda(\ell_1(S f'_{:,d^+}) + \ell_1(S f'_{:,d^-})), \tag{7.17}$$

avec $\lambda \in [0, 1]$ et $S f'$ la matrice index inversée calculée par f.

Initialisation

Afin de garantir que les TDV de tous les termes ne soient pas nuls au début de l'entraînement, nous avons initialisé le biais b_{nn} à 1. Le vecteur de poids w est initialisé avec l'initialisation par défaut de *Tensorflow*.

7.4 Résultats

7.4.1 Performances des modèles NLTR efficients

Collections TREC et *Wikipédia*

Le Tableau 7.2 montre la ndcg@5 des modèles de référence et des modèles NLTR efficients sur les collections TREC et *Wikipédia*. Notre principale observation est que l'apprentissage et l'incorporation des TDV dans les fonctions de RI basés sur la correspondance exacte améliorent les performances de ces dernières sur

1. https://www.tensorflow.org

Modèle	AP88-89		LA		FT91-94		wikIR78k		wikIRS78k	
	Réf.	TDV	Réf.	TDV	Réf.	TDV	Réf.	TDV	Réf.	TDV
TF-IDF	0.2638	0.3028^{+}	0.1792	0.2354^{+}	0.1782	0.2511^{+}	0.1629	0.2466^{+}	0.1452	0.2321^{+}
JM	0.4054	0.4144	0.3156	0.3333^{+}	0.3323	0.3459	0.2988	0.3156^{+}	0.2698	0.2963^{+}
Dir	0.4464	0.4630^{+}	0.3450	0.3616	0.3515	$\mathbf{0.3778}^{+}$	0.3252	0.3391^{+}	0.2931	0.3174^{+}
BM25	0.4470	$\mathbf{0.4709}^{+}$	0.3498	$\mathbf{0.4004}^{+}$	0.3531	0.3698^{+}	0.3269	0.3402^{+}	0.2944	0.3226^{+}
DRMM	0.4405	\	0.3622	\	0.3595	\	0.3659	\	0.3306	\
DUET	0.4386	\	0.1526	\	0.1688	\	0.3520	\	0.3318	\
CKNRM	0.4413	\	0.2265	\	0.3795	\	0.3285	\	0.3361	\
Transf-Rep	0.1747	\	0.1172	\	0.1229	\	0.3408	\	0.3495	\
ID-Sinter	0.3941	\	0.3327	\	0.3285	\	**0.3962**	\	**0.3570**	\

TABLE 7.2 – ndcg@5 des modèles NLTR efficients sur les collections TREC et *Wikipédia*. Les amélioration/dégradations statistiquement significatives par rapport au modèle basé sur la correspondance exacte associé sont notées par (+/-) avec une p-value < 0.01.

toutes les collections. Le fait que ces améliorations soient également observées sur les collections TREC montre que les modèles NLTR efficients n'ont pas besoin de beaucoup de requêtes résolues pour atteindre de meilleurs performances que les modèles de référence basés sur la correspondance exacte. Nous pensons que cette faible quantité de requêtes résolues requise par les modèles NLTR efficients est due à deux facteurs :

1. Le faible nombre de paramètres à optimiser (voir section 7.3.2)
2. L'initialisation du biais égal à 1 implique que les performances des modèles NLTR efficients sont déjà proches de celles des modèles de références avant même de commencer l'entraînement

Bien que plus performant que les modèles de références basés sur la correspondance exacte, les modèles NLTR efficients ne parviennent pas à atteindre les mêmes performances que les modèles NLTR de référence sur les collections *Wikipédia*. Ces résultats suggèrent que sur les collections ayant suffisamment de requêtes résolues, les modèles NLTR de référence sont les plus performants.

Affinage sur les collections TREC

Le Tableau 7.3 montre la ndcg@5 des modèles NLTR efficients pré-entraînés sur les collections *Wikipédia* et affinés sur les collections TREC. Comme nous pouvons le constater, le fait de pré-entraîner les modèles NLTR efficients sur les collections *Wikipédia* ne conduit pas à des gains de performance en terme de ndcg@5. Nous pensons qu'à cause du faible nombre de paramètres des modèles NLTR efficients, ces derniers ne sont pas capables de bénéficier des connaissances *a priori* présentes dans les collections *Wikipédia* afin d'améliorer leurs performances sur les collections TREC.

7.4.2 Accélération de la recherche

Collections TREC

Le Tableau 7.4 indique le temps moyen de recherche par requête des modèles sur les trois collections de TREC. Tout d'abord, comme nous pouvions nous y attendre, nous remarquons que les modèles de référence NLTR sont considérablement plus lents pour retrouver les documents que les autres modèles qui utilisent des index inversés. Deuxièmement, en supprimant de l'index inversé les termes ayant une valeur de discrimination nulle, nous sommes en mesure d'accélérer considérablement le processus de récupération de tout les modèles sur toutes les collections. Sur

Modèle	AP88-89		LA		FT91-94	
	Réf.	TDV	Réf.	TDV	Réf.	TDV
TF-IDF	0.2638	0.3028	0.1792	0.2354	0.1782	0.2511
$\text{TF-IDF}_{\text{wikIR}}$	\	0.3153	\	0.2310	\	0.2545
$\text{TF-IDF}_{\text{wikIRS}}$	\	0.3013	\	0.2413	\	0.2532
JM	0.4054	0.4144	0.3156	0.3333	0.3323	0.3459
JM_{wikIR}	\	0.4115	\	0.3442	\	0.3558
$\text{JM}_{\text{wikIRS}}$	\	0.4085	\	0.3320	\	0.3484
Dir	0.4464	0.4630	0.3450	0.3616	0.3515	0.3778
$\text{Dir}_{\text{wikIR}}$	\	0.4669	\	0.3536	\	0.3644
$\text{Dir}_{\text{wikIRS}}$	\	0.4657	\	0.3605	\	0.3845
BM25	0.4470	0.4709	0.3498	0.4004	0.3531	0.3698
$\text{BM25}_{\text{wikIR}}$	\	0.4858	\	0.4276	\	0.3523
$\text{BM25}_{\text{wikIRS}}$	\	0.4438	\	0.3915	\	0.3884

TABLE 7.3 – ndcg@5 des modèles NLTR efficients pré-entraînés sur les collections *Wikipédia* et affinés sur les collections TREC.

les collections LA et FT91-94 TREC, le temps de recherche de BM25 est presque divisé par deux et sur AP88-89, le temps de recherche de BM25 est divisé par 3. Il est intéressant de noter que les modèles de langue prennent systématiquement plus de temps pour récupérer les documents que les autres modèles. En effet, les modèles le langue doivent calculer des logarithmes (qui sont coûteux en termes de calcul) au moment de la recherche (voir équations 7.9 et 7.11), alors que les opérations coûteuses en temps de calcul de BM25 et TF-IDF qui sont nécessaires pour le calcul de l'idf et de l'$\widetilde{\text{IDF}}$ sont effectuées "hors-ligne" et une table de correspondance est utilisée au moment de la recherche.

Collections *Wikipédia*

Le Tableau 7.5 indique le temps moyen de recherche par requête des modèles sur les deux collections *Wikipédia*. De la même façon que sur les collections TREC, les modèles de référence NLTR sont considérablement plus lents à retrouver les documents que les modèles utilisant des index inversés et le fait de supprimer les *posting lists* associées au termes non-discriminant de l'index inversé permet d'accélérer le processus de recherche. De plus, le temps de recherche moyen est plus important sur la collection wikIRS78k que sur la collection wikIR78k. Cela s'explique par le fait que, en moyenne, les requêtes de wikIRS78k sont 4 fois plus longues que les

Modèle	AP88-89		LA		FT91-94	
	Réf.	TDV	Réf.	TDV	Réf.	TDV
TF-IDF	147.4	29.2	26.3	4.6	56.6	15.1
JM	557.5	154.9	121.3	53.5	226.1	108.7
Dir	683.8	180.1	139.6	54.3	270.9	122.6
BM25	207.0	61.2	29.2	16.6	83.1	42.8
DRMM	$>10^4$	\	$>10^4$	\	$>10^4$	\
DUET	$>10^5$	\	$>10^5$	\	$>10^5$	\
CKNRM	$>10^5$	\	$>10^5$	\	$>10^5$	\
Transf-Rep	$>10^6$	\	$>10^6$	\	$>10^6$	\
ID-Sinter	$>10^5$	\	$>10^5$	\	$>10^5$	\

TABLE 7.4 – Comparaison du temps moyen (en millisecondes) de recherche par requête des modèles NLTR efficients sur les collections TREC.

requêtes de wikIR78k, ce qui se traduit par un nombre plus grand de *posting lists* considérées pour résoudre une requête.

Modèle	wikIR78k		wikIRS78k	
	Réf.	TDV	Réf.	TDV
TF-IDF	356.0	89.0	1086.0	183.7
JM	1023.1	403.2	2698.8	772.4
Dir	1359.8	489.5	3105.9	886.1
BM25	426.7	102.8	1256.3	218.1
DRMM	$>10^5$	\	$>10^5$	\
DUET	$>10^6$	\	$>10^7$	\
CKNRM	$>10^6$	\	$>10^7$	\
Transf-Rep	$>10^7$	\	$>10^7$	\
ID-Sinter	$>10^6$	\	$>10^7$	\

TABLE 7.5 – Comparaison du temps moyen (en millisecondes) de recherche par requête des modèles NLTR efficients sur les collections *Wikipédia*.

Affinage sur les collections TREC

Le Tableau 7.6 indique le temps moyen de recherche par requête des modèles NLTR efficients pré-entraînés sur les deux collections *Wikipédia* et affinés sur les collections TREC. Comme observé précédemment sur la ndcg@5, le fait de pré-

entraîner les modèles NLTR efficients sur les collections *Wikipédia* ne conduit pas à une amélioration de la vitesse de recherche sur les collections TREC.

Modèle	AP88-89		LA		FT91-94	
	Réf.	TDV	Réf.	TDV	Réf.	TDV
TF-IDF	147.4	29.2	26.3	4.6	56.6	15.1
$\text{TF-IDF}_{\text{wikIR}}$	\	29.1	\	4.5	\	15.0
$\text{TF-IDF}_{\text{wikIRS}}$	\	28.9	\	4.4	\	15.4
JM	557.5	154.9	121.3	53.5	226.1	108.7
JM_{wikIR}	\	149.4	\	51.6	\	109.1
$\text{JM}_{\text{wikIRS}}$	\	154.1	\	53.2	\	109.0
Dir	683.8	180.1	139.6	54.3	270.9	122.6
$\text{Dir}_{\text{wikIR}}$	\	179.0	\	53.9	\	121.8
$\text{Dir}_{\text{wikIRS}}$	\	184.5	\	55.6	\	125.6
BM25	207.0	61.2	29.2	16.6	83.1	42.8
$\text{BM25}_{\text{wikIR}}$	\	63.0	\	17.1	\	44.1
$\text{BM25}_{\text{wikIRS}}$	\	61.9	\	16.7	\	43.2

TABLE 7.6 – Comparaison du temps moyen (en millisecondes) de recherche par requête des modèles NLTR efficients pré-entraînés sur les collections *Wikipédia* et affinés sur les collections TREC.

7.4.3 Réduction de l'empreinte mémoire

Collections TREC et *Wikipédia*

Le Tableau 7.7 décrit la réduction de l'empreinte mémoire de l'index inversé sur les collections TREC et *Wikipédia* obtenue en supprimant les *posting lists* des termes ayant une valeur de discrimination nulle selon les modèles NLTR efficients. Nous constatons que pour toutes les collections, nous sommes en mesure de supprimer une partie importante de l'index inversé tout en ayant de meilleurs performances en terme de ndcg@5 et de vitesse de recherche.

Affinage sur les collections TREC

Le Tableau 7.8 décrit la réduction de l'empreinte mémoire de l'index inversé obtenue en supprimant les *posting lists* des termes ayant une valeur de discrimination nulle selon les modèles NLTR efficients pré-entraînés sur les collections *Wikipédia*

Modèle	AP88-89	LA	FT91-94	wikIR78k	wikIRS78k
TDV-TF-IDF	-45.00%	-39.67%	-45.77%	-38.50%	-49.42%
TDV-JM	-48.65%	-31.35%	-45.82%	-39.80%	-47.51%
TDV-Dir	-46.06%	-34.03%	-40.25%	-36.97%	-51.93%
TDV-BM25	-46.91%	-32.35%	-44.42%	-40.81%	-49.09%

TABLE 7.7 – Réduction de l'empreinte mémoire de l'index inversé.

et affinés sur les collections TREC. Tout comme pour la ndcg@5 et la vitesse de recherche, pré-entraîner les modèles NLTR efficients sur les collections *Wikipédia* ne conduit pas à une réduction de l'empreinte mémoire plus importante que les modèles sans pré-entraînement.

Modèle	AP88-89	LA	FT91-94
TDV-TF-IDF	-45.00%	-39.67%	-45.77%
TDV-TF-IDF$_{\text{wikIR}}$	-45.12%	-39.54%	-45.69%
TDV-TF-IDF$_{\text{wikIRS}}$	-45.64%	-41.16%	-47.09%
TDV-JM	-48.65%	-31.35%	-45.82%
TDV-JM$_{\text{wikIR}}$	-50.04%	-30.62%	-44.25%
TDV-JM$_{\text{wikIRS}}$	-50.20%	-31.86%	-47.83%
TDV-Dir	-46.06%	-34.03%	-40.25%
TDV-Dir$_{\text{wikIR}}$	-48.73%	-33.83%	-41.54%
TDV-Dir$_{\text{wikIRS}}$	-45.27%	-32.68%	-37.48%
TDV-BM25	-46.91%	-32.35%	-44.42%
TDV-BM25$_{\text{wikIR}}$	-47.32%	-31.08%	-45.63%
TDV-BM25$_{\text{wikIRS}}$	-46.68%	-33.62%	-42.65%

TABLE 7.8 – Réduction de l'empreinte mémoire de l'index inversé.

7.5 Conclusion

Dans ce chapitre, nous avons proposé d'apprendre des TDV en utilisant de l'apprentissage supervisé. Afin d'avoir des TVD spécifiques aux modèles de RI basés sur la correspondance exacte et afin de pouvoir utiliser des méthodes d'optimisation basées sur la descente de gradient, nous avons développé des modèles NLTR approximant les fonctions de classement des modèles basés sur la correspondance exacte avec des opérations différentiables. Les TDV calculées sont intégrées aux

index inversés en multipliant les tf des termes par les TDV et les *posting lists* des termes non-discriminant (tdv = 0) sont supprimées.

Nous constatons que le fait de supprimer les termes ayant une TDV nulle de l'index inversé permet d'accélérer considérablement la recherche avec une légère amélioration des performances sur les trois collections TREC et les deux collections *Wikipédia* utilisées dans nos expériences. De plus, le fait que les modèles NLTR efficients n'aient que peu de paramètres leur permet de ne nécessiter que peu de requêtes résolues pour avoir de meilleures performances que les modèles de référence basés sur la correspondance exacte. En revanche, ce faible nombre de paramètres les empêche de bénéficier de l'apport de connaissances *a priori* via un pré-entraînement sur les collections *Wikipédia*.

Chapitre 8

Conclusion générale

Dans ce travail de thèse, nous avons étudié l'apport de connaissances *a priori* à la recherche d'information textuelle neuronale. Nous avons présenté le fonctionnement d'un système de recherche d'information (RI) textuel utilisant des modèles basés sur la correspondance exacte, les modèles *neural learning-to-rank* (NLTR) pour la RI textuelle, puis nous avons décrit nos contributions consistant à l'utilisation de différentes formes de connaissances *a priori* pour améliorer la performance ou l'efficience des modèles NLTR pour la RI textuelle.

Un système de RI utilisant des modèles basés sur la correspondance exacte est dans la capacité de classer des documents en fonction de leur pertinence par rapport au besoin de l'utilisateur de façon efficiente. Néanmoins, les modèles basés sur la correspondance exacte souffrent du problème de disparité des termes. Une solution à la disparité des termes est d'utiliser des modèles NLTR pour la RI qui sont capables de modéliser des relations complexes entre termes. Ces modèles présentent également certaines limitations et nous proposons d'aborder 3 d'entre elles en utilisant des connaissances *a priori* : **(1)** la nécessité d'avoir de grandes quantités de requêtes résolues ; **(2)** l'absence de prise en compte de ressources sémantiques pour modéliser le langage ; **(3)** l'incompatibilité avec l'index inversé qui entraîne un manque d'efficience.

Dans un premier temps, nous avons utilisé des connaissances *a priori* issues d'une ressource semi-structurée, *Wikipédia*, afin de réduire la quantité de requêtes résolues pour entraîner les modèles neuronaux. Pour ce, nous avons développé *WIKIR* : boîte à outils pour la construction de collections de RI à grande échelle à partir de *Wikipédia*. Nous avons proposé un cadre théorique pour la construction de collections de RI en utilisant des ressources satisfaisant 3 propriétés. Nous avons utilisé les collections construites à partir de *WIKIR* pour pré-entraîner des modèles neuronaux pour la RI. Ces derniers, et contrairement aux modèles sans pré-entraînement,

ont besoin de faibles quantités (de l'ordre de la centaine) de requêtes résolues pour atteindre des performances supérieures aux modèles basés sur la correspondance exacte.

Dans un second temps, nous avons utilisé des connaissances *a priori* sous forme de ressources sémantiques afin d'améliorer les performances des modèles NLTR. Nous avons développé des modèles NLTR sémantiques pour la RI utilisant des représentations vectorielles de concepts issues de ressources sémantiques. Nous avons exploré 3 architectures de modèles NLTR sémantiques ainsi que 4 encodeurs de séquences de vecteurs. Nous avons empiriquement montré que les modèles NLTR sémantiques atteignent des performances supérieures aux modèles NLTR pour la RI dans le domaine médical, mais pas dans le domaine général. Nous avons également montré que les encodeurs *Transformers* et les vecteurs contextualisés BERT, lorsqu'ils sont utilisés avec des modèles basés sur la représentation, permettent de surpasser les modèles NLTR de référence pour la RI.

Finalement, nous avons conceptualisé des modèles NLTR efficients, compatibles avec des index inversés et utilisant des connaissances *a priori* issues de texte non structuré. Nous utilisons un réseau de neurones à une couche pour calculer des valeurs de discrimination de terme (TDV) à l'aide de vecteurs de mots. Les TDV sont ensuite optimisées pour maximiser le score de modèles NLTR approximant de façon différentiable des modèles de RI basés sur la correspondance exacte. Les TDV apprises sont ensuite utilisées pour modifier ou supprimer des éléments des index inversés. Nous montrons empiriquement que les modèles NLTR efficients permettent d'accélérer considérablement la recherche avec une légère amélioration des performances et une réduction significative de l'empreinte mémoire de l'index inversé.

Perspectives

Les travaux effectués dans cette thèse peuvent être poursuivis de différentes façons telles que l'extension à d'autres langues que l'anglais, l'exploration d'autres architectures de réseaux de neurones sémantiques ou encore l'apprentissage de TDV de concepts.

Dans un premier temps, nous proposons d'utiliser des connaissances *a priori* issues de *Wikipédia* pour la RI neuronale textuelle en d'autres langues que l'anglais. Dans le cadre de ce travail de thèse, nous avons développé *MLWIKIR* (Frej et al., 2020c), une boîte à outils pour construire des collections de RI en 10 langues basées sur *Wikipédia*. Nous n'avons néanmoins pas eu l'occasion d'affiner les modèles pré-entraînés sur les collections *Wikipédia* sur des collections de RI des langues considé-

rées. Il s'agit là d'une des premières perspectives que nous proposons : l'adaptation de la stratégie d'affinage à d'autres langues que l'anglais. Nous pensons également à utiliser le cadre théorique pour la construction automatique de collections de RI que nous avons proposé sur d'autres ressources que *Wikipédia*. Nous pouvons par exemple utiliser des articles issus de PubMed ainsi que les termes MeSH associés pour construire une collection de RI dans le domaine médical.

Dans un second temps, nous pensons qu'il existe plusieurs stratégies pour améliorer les performances des modèles NLTR sémantiques que nous avons proposés. Nous pensons qu'une architecture hybride similaire à celle du modèle DUET utilisant un sous-modèle basé sur l'interaction modélisant de façon explicite le signal de correspondance exacte ainsi qu'un sous-modèle basé sur la représentation est un bon point de départ pour améliorer les performances des modèles NLTR sémantiques. De plus, nous pensons que l'utilisation de modèles de langues plus récents que BERT tels que XLM (Conneau et Lample, 2019), XLNet (Yang et al., 2019b), RoBERTa (Liu et al., 2019) ou Albert (Lan et al., 2020) peuvent apporter de meilleures représentations vectorielles de mots pour les modèles NLTR sémantiques. Nous pensons également que l'utilisation de GCN (*Graph Convolutional Network* (Kipf et Welling, 2017)) pour apprendre des vecteurs de graphe relationnel (Ye et al., 2019) peut conduire à des améliorations de performances des modèles NLTR sémantiques.

Finalement, concernant les modèles NLTR efficients, nous pensons qu'il est important d'explorer d'autres modèles pour calculer les TDV, notamment des modèles plus complexes avec plus de paramètres. En effet, le fait d'utiliser un réseau de neurones à une couche ayant peu de paramètres permet aux modèles de ne nécessiter que peu de requêtes résolues pour l'apprentissage, mais ne leur permet pas de bénéficier de l'apport de grandes quantités de requêtes résolues. De plus, dans ce travail de thèse, nous n'avons pas utilisé de vecteurs de mots contextualisés avec les modèles NLTR efficients proposés. Il s'agit d'une tâche non-triviale en raison du fait que les vecteurs contextualisés sont associés à un vocabulaire de n-gramme de caractères de 30000 éléments (Devlin et al., 2019), ce qui pose en problème pour la RI : un vocabulaire de taille réduite conduit à des *posting lists* plus longues dans l'index inversé et donc à une recherche peu efficiente, ce qui est à l'opposé de ce que nous recherchons avec les modèles NLTR efficients. Enfin, les modèles NLTR efficients souffrent du problème de disparité des termes. Pour remédier à cela, nous proposons d'annoter les requêtes et documents avec des concepts issus de ressources sémantiques et, de la même façon que pour les mots, d'apprendre des TDV associées aux concepts et de supprimer de l'index inversé les concepts non-discriminants. Plus généralement, nous pensons que le développement de réseaux

de neurones efficients est crucial pour la RI qui nécessite de comparer rapidement des milliards de documents à une requête. Nous pensons que l'incorporation de connaissances *a priori* sous forme de vecteurs de mots, de vecteurs de concepts, ou encore d'hypothèses sur la tâche de RI (Mitra et al., 2019) permettra de développer de tels modèles pour la recherche d'information.

Annexe A

Publications

A.1 Conférences internationales avec évaluation par les pairs

- Jibril Frej, Philippe Mulhem, Didier Schwab et Jean-Pierre Chevallet. Learning Term Discrimination. In Proceedings of the 43rd International conference on research and development in Information Retrieval (SIGIR 2020). 2020.
- Jibril Frej, Didier Schwab et Jean-Pierre Chevallet. WIKIR : A Python Toolkit for Building a Large-scale Wikipedia-based English Information Retrieval Dataset. In Proceedings of the 12th International Conference on Language Resources and Evaluation (LREC 2020). 2020
- Hang Le, Loïc Vial, Jibril Frej, Vincent Segonne, Maximin Coavoux, Benjamin Lecouteux, Alexandre Allauzen, Benoit Crabbé, Laurent Besacier et Didier Schwab. FlauBERT : Unsupervised Language Model Pre-training for French. In Proceedings of the 12th International Conference on Language Resources and Evaluation (LREC 2020). 2020

A.2 Conférences nationales avec évaluation par les pairs

- Jibril Frej, Didier Schwab et Jean-Pierre Chevallet. MLWIKIR : A Python Toolkit for Building Large-scale Wikipedia-based Information Retrieval Datasets in Chinese, English, French, Italian, Japanese, Spanish and More. In Proceedings of the

Joint Conference of the Information Retrieval Communities in Europe (CIRCLE 2020). 2020.

- Jibril Frej, Didier Schwab et Jean-Pierre Chevallet. Knowledge Based Transformer Model for Information Retrieval. In Proceedings of the Joint Conference of the Information Retrieval Communities in Europe (CIRCLE 2020). 2020.
- Hang Le, Loïc Vial, Jibril Frej, Vincent Segonne, Maximin Coavoux, Benjamin Lecouteux, Alexandre Allauzen, Benoit Crabbé, Laurent Besacier et Didier Schwab. FlauBERT : des modèles de langue contextualisés pré-entraînés pour le français. In 27ème conférence sur le Traitement Automatique des Langues Naturelles (TALN 2020). 2020.
- Jibril Frej, Philippe Mulhem, Didier Schwab et Jean-Pierre Chevallet. Combining Subword information and Language model for Information Retrieval. In 15ème COnférence en Recherche d'Informations et Application (CORIA 2018). 2018.

A.3 Autres

- Jibril Frej, Didier Schwab et Jean-Pierre Chevallet. Modèle Transformer à base de Connaissances pour la Recherche d'Information dans des Domaines Spécialisés. In Atelier Deep Learning pour le traitement automatique des langues (DL for NLP). 2020.
- Jibril Frej, Didier Schwab et Jean-Pierre Chevallet. Enhancing Translation Language Models with Word Embedding for Information Retrieval. In 9ème Atelier de Recherche d'Information SEmantique (RISE 2017). 2017.

Annexe B

Définitions

Recherche d'information : Domaine de l'informatique étudiant la récupération de documents issus d'une collection. Les documents sont récupérés dans le but de satisfaire un besoin d'information d'un utilisateur, exprimé sous forme de requête Baeza-Yates et Ribeiro-Neto (2011).

Recherche d'information *ad-hoc* : En recherche d'information *ad-hoc* il est supposé que "les documents de la collection restent relativement statiques tandis que de nouvelles requêtes sont soumises au système." Baeza-Yates et Ribeiro-Neto (2011).

Apprentissage automatique : L'apprentissage automatique est un domaine de l'intelligence artificielle qui utilise des approches statistiques pour donner aux ordinateurs la capacité d'apprendre à partir de données.

Apprentissage profond : Modèles informatiques composés de plusieurs couches de traitement qui apprennent des représentations de données avec plusieurs niveaux d'abstraction LeCun et al. (2015).

Apprendre à classer : "L'apprentissage de classement pour la recherche d'information est une tâche qui consiste à construire automatiquement un modèle de classement en utilisant des données d'entraînement, de sorte que le modèle puisse trier de nouveaux objets en fonction de leur degré de pertinence, de préférence ou d'importance" Liu (2009).

Réseau de neurones : Modèle généralement composé de plusieurs couches qui apprend à associer une entrée de à une sortie. Pour passer d'une couche à l'autre, un ensemble d'unités calculent une somme pondérée de leurs entrées et passent le

résultat par une fonction non linéaire. Un réseau est dit profond si il comporte un nombre élevé de couches (en général à partir d'une dizaine) LeCun et al. (2015).

Réseau de neurones récurrents : "Modèle traitant une séquence d'entrée de taille variable un élément à la fois, en maintenant un vecteur d'état qui contient implicitement des informations sur l'historique de tous les éléments passés de la séquence" LeCun et al. (2015).

Ressource sémantique : Formalisation informatique de la signification véhiculé par les mots pouvant prendre la forme d'un thésaurus, d'une base lexicale, d'une ontologie etc... (Zargayouna et al., 2015)

Ontologie peuplée d'instances : Ressource sémantique peuplée d'instances de type (classe + objet + propriété + relation entre objets) décrivant des faits qui sont typiquement le contenu d'une base de données. (Zargayouna et al., 2015)

Thésaurus : "Type de ressource sémantique constituée d'une liste de termes identifiés comme descripteurs ou non-descripteurs. Les termes non descripteurs renvoient aux descripteurs par une relation d'équivalence. [...] Les descripteurs sont reliés entre eux par au moins deux types de relations : une relation de spécialisation ou de généralisation et une relation associative sans signification explicite." (Zargayouna et al., 2015)

Annexe C

Résultats supplémentaires

modèles	P@5	P@10	ndcg@5	ndcg@10	rappel@1000
BM25	0.2846	0.2419	0.3524	0.3267	0.4740
BM25-RM3	0.3056	0.2603	0.3664	0.3431	0.6249
ArcI	0.1790^{-}	0.1566^{-}	0.1916^{-}	0.1669^{-}	0.7622^{+}
ArcII	0.1876^{-}	0.1544^{-}	0.1838^{-}	0.1638^{-}	0.7688^{+}
MatchPyramid	0.2320^{-}	0.2046^{-}	0.2209^{-}	0.1990^{-}	0.7903^{+}
DRMM	0.2940	0.2489	0.3540	0.3330	0.7051^{+}
DUET	0.1967^{-}	0.1840^{-}	0.1857^{-}	0.1883^{-}	0.7673^{+}
KNRM	0.2082^{-}	0.1887^{-}	0.1914^{-}	0.1914^{-}	0.7764^{+}
CKNRM	0.3146	0.2865	0.3010^{-}	0.3090	0.8143^{+}
ID-Rep	0.0572^{-}	0.0442^{-}	0.0832^{-}	0.0547^{-}	0.2701^{-}
CNN-Rep	0.3265	0.2976^{+}	0.3362	0.3287	0.8133^{+}
LSTM-Rep	0.2762	0.2485	0.2856^{-}	0.2706^{-}	0.7986^{+}
Transf-Rep	$\mathbf{0.3554}^{+}$	$\mathbf{0.3294}^{+}$	**0.3708**	**0.3584**	$\mathbf{0.8520}^{+}$
ID-SInter	0.3466^{+}	0.3161^{+}	0.3575	0.3535	0.8464^{+}
CNN-SInter	0.3090	0.2895	0.3232^{-}	0.3079^{-}	0.8321^{+}
LSTM-SInter	0.2795	0.2589	0.2912^{-}	0.2743^{-}	0.7877^{+}
Transf-SInter	0.3300^{+}	0.3129^{+}	0.3481	0.3312	0.8322^{+}
ID-GInter	0.3290	0.3033^{+}	0.3474	0.3287	0.8305^{+}
CNN-GInter	0.2893	0.2718	0.3044^{-}	0.2871^{-}	0.8094^{+}
LSTM-GInter	0.2601^{-}	0.2417	0.2708^{-}	0.2539^{-}	0.7708^{+}
Transf-GInter	0.3114	0.2957^{+}	0.3305	0.3124^{-}	0.8114^{+}

TABLE C.1 – Comparaison des performances des modèles de référence et de nos contributions sur le NFCorpus. Les amélioration/dégradations statistiquement significatives par rapport à BM25-RM3 sont notées par (+/-) avec une p-value $<$ 0.01.

modèles	P@5	P@10	ndcg@5	ndcg@10	rappel@1000
ID-Rep$_{bw}$	0.0572	0.0442	0.0832	0.0547	0.2701
ID-Rep$_{BB}$	0.3476	0.3013	0.3451	0.3469	0.8494
CNN-Rep$_{bw}$	0.3265	0.2976	0.3362	0.3287	0.8133
CNN-Rep$_{BB}$	0.2671	0.2379	0.2785	0.2645	0.7813
LSTM-Rep$_{bw}$	0.2762	0.2485	0.2856	0.2706	0.7986
LSTM-Rep$_{BB}$	0.2318	0.1938	0.2478	0.2272	0.7930
Transf-Rep$_{bw}$	0.3554	0.3294	0.3708	0.3584	0.8520
Transf-Rep$_{BB}$	0.2542	0.1982	0.2557	0.2553	0.7891
ID-SInter$_{bw}$	0.3466	0.3161	0.3575	0.3535	0.8464
ID-SInter$_{BB}$	0.1877	0.1571	0.1984	0.1864	0.7462
CNN-SInter$_{bw}$	0.3090	0.2895	0.3232	0.3079	0.8321
CNN-SInter$_{BB}$	0.2006	0.1623	0.2034	0.1863	0.7545
LSTM-SInter$_{bw}$	0.2795	0.2589	0.2912	0.2743	0.7877
LSTM-SInter$_{BB}$	0.1884	0.1481	0.1992	0.1828	0.7422
Transf-SInter$_{bw}$	0.3300	0.3129	0.3481	0.3312	0.8322
Transf-SInter$_{BB}$	0.2416	0.1958	0.2525	0.2347	0.7918
ID-GInter$_{bw}$	0.3290	0.3033	0.3474	0.3287	0.8305
ID-GInter$_{BB}$	0.2030	0.1540	0.2144	0.1925	0.7604
CNN-GInter$_{bw}$	0.2893	0.2718	0.3044	0.2871	0.8094
CNN-GInter$_{BB}$	0.1063	0.0691	0.1316	0.0995	0.6765
LSTM-GInter$_{bw}$	0.2601	0.2417	0.2708	0.2539	0.7708
LSTM-GInter$_{BB}$	0.1755	0.1311	0.1985	0.1815	0.7381
Transf-GInter$_{bw}$	0.3114	0.2957	0.3305	0.3124	0.8114
Transf-GInter$_{BB}$	0.1974	0.1570	0.2077	0.1902	0.7582

TABLE C.2 – Comparaison de l'utilisation des vecteurs *bio-word2vec* (bw) avec les vecteurs BioBERT (BB) sur le NFCorpus.

wikIR78k								
Modèle	P@5	P@10	P@20	nDCG@5	nDCG@10	nDCG@20	nDCG	MAP
BM25	0.2622	0.2039	0.1498	0.3269	0.3045	0.3098	0.3555	0.1498
BM25-RM3	0.2844	0.2235	0.1663	0.3442	0.3238	0.3305	0.3813	0.1710
DRMM	0.2941^{+}	0.2312^{+}	0.1745^{+}	0.3659^{+}	0.3394^{+}	0.3420^{+}	0.3861^{+}	$\mathbf{0.1773}^{+}$
DUET	0.2819	0.2243	0.1721^{+}	0.3520^{+}	0.3239	0.3297	0.3746^{-}	0.1656^{-}
CKNRM	0.2792	0.2268	0.1762^{+}	0.3285^{-}	0.3114^{-}	0.3199^{-}	0.3622^{-}	0.1599^{-}
ID-Rep	0.1342^{-}	0.0978^{-}	0.0726^{-}	0.1546^{-}	0.1479^{-}	0.1550^{-}	0.1720^{-}	0.0721^{-}
CNN-Rep	0.2545^{-}	0.1966^{-}	0.1450^{-}	0.3075^{-}	0.2896^{-}	0.2984^{-}	0.3434^{-}	0.1441^{-}
LSTM-Rep	0.2323^{-}	0.1750^{-}	0.1228^{-}	0.2761^{-}	0.2546^{-}	0.2643^{-}	0.3090^{-}	0.1261^{-}
Transf-Rep	0.2716^{-}	0.2173	0.1596	0.3408	0.3173	0.3166	0.3715	0.1516^{-}
ID-SInter	$\mathbf{0.3150}^{+}$	$\mathbf{0.2429}^{+}$	$\mathbf{0.1828}^{+}$	$\mathbf{0.3962}^{+}$	$\mathbf{0.3652}^{+}$	$\mathbf{0.3733}^{+}$	$\mathbf{0.4290}^{+}$	0.1769^{+}
CNN-SInter	0.2380^{-}	0.1719^{-}	0.1287^{-}	0.2752^{-}	0.2693^{-}	0.2726^{-}	0.3160^{-}	0.1283^{-}
LSTM-SInter	0.2108^{-}	0.1570^{-}	0.1175^{-}	0.2601^{-}	0.2437^{-}	0.2465^{-}	0.2772^{-}	0.1178^{-}
Transf-SInter	0.2440^{-}	0.1831^{-}	0.1426^{-}	0.3050^{-}	0.2891^{-}	0.3023^{-}	0.3365^{-}	0.1400^{-}
ID-GInter	0.2874	0.2201	0.1576	0.3601^{+}	0.3226	0.3430	0.3841	0.1581
CNN-GInter	0.2164^{-}	0.1658^{-}	0.1151^{-}	0.2396^{-}	0.2480^{-}	0.2561^{-}	0.2916^{-}	0.1187^{-}
LSTM-GInter	0.1955^{-}	0.1511^{-}	0.1083^{-}	0.2444^{-}	0.2184^{-}	0.2196^{-}	0.2395^{-}	0.0995^{-}
Transf-GInter	0.2238^{-}	0.1641^{-}	0.1254^{-}	0.2758^{-}	0.2622^{-}	0.2716^{-}	0.3063^{-}	0.1262^{-}

TABLE C.3 – Comparaison des performances des différents modèles sur wikIR78k. Les amélioration/dégradations statistiquement significatives par rapport à BM25-RM3 sont notées par (+/-) avec une p-value < 0.01.

wikIRS78k								
Modèle	P@5	P@10	P@20	nDCG@5	nDCG@10	nDCG@20	nDCG	MAP
BM25	0.2177	0.1634	0.1186	0.2944	0.2673	0.2695	0.3085	0.1163
BM25-RM3	0.2237	0.1698	0.1243	0.3015	0.2733	0.2753	0.3158	0.1216
DRMM	0.2486^{+}	0.1867^{+}	0.1404^{+}	0.3306^{+}	0.2991^{+}	0.2976^{+}	0.3288^{+}	0.1361^{+}
DUET	0.2601^{+}	0.1998^{+}	0.1479^{+}	0.3318^{+}	0.3051^{+}	0.3042^{+}	0.3281^{+}	0.1367^{+}
CKNRM	0.2779^{+}	0.2148^{+}	0.1567^{+}	0.3361^{+}	0.3134^{+}	0.3150^{+}	0.3369^{+}	0.1465^{+}
ID-Rep	0.1281^{-}	0.0761^{-}	0.0622^{-}	0.1257^{-}	0.1270^{-}	0.1483^{-}	0.1513^{-}	0.0605^{-}
CNN-Rep	0.2621^{+}	0.2046^{+}	0.1485^{+}	0.3178^{+}	0.2973^{+}	0.2964^{+}	0.3030^{+}	0.1356^{+}
LSTM-Rep	0.2409^{+}	0.1830^{+}	0.1365^{+}	0.2740^{+}	0.2633^{+}	0.2621^{+}	0.2799^{+}	0.1272^{+}
Transf-Rep	$\mathbf{0.2934}^{+}$	$\mathbf{0.2287}^{+}$	$\mathbf{0.1669}^{+}$	$\mathbf{0.3495}^{+}$	$\mathbf{0.3332}^{+}$	$\mathbf{0.3306}^{+}$	0.3556^{+}	0.1548^{+}
ID-SInter	0.2657^{+}	0.2093^{+}	0.1404^{+}	0.3570^{+}	0.3281^{+}	0.3244^{+}	$\mathbf{0.3592}^{+}$	$\mathbf{0.1574}^{+}$
CNN-SInter	0.2281	0.1636	0.1237	0.3061	0.2670	0.2656	0.3209	0.1146
LSTM-SInter	0.2062^{-}	0.1609	0.1125	0.2936	0.2567	0.2616	0.3095	0.1163
Transf-SInter	0.2213	0.1608	0.1189	0.2932	0.2627	0.2658	0.3099	0.1141
ID-GInter	0.2407^{+}	0.1828^{+}	0.1294	0.3228^{+}	0.2871^{+}	0.2969^{+}	0.3296	0.1323
CNN-GInter	0.2080^{-}	0.1536	0.1138^{-}	0.2761^{-}	0.2551^{-}	0.2609	0.2836^{-}	0.1132^{-}
LSTM-GInter	0.2026^{-}	0.1496^{-}	0.1164	0.2872^{-}	0.2526^{-}	0.2604^{-}	0.2973^{-}	0.1096^{-}
Transf-GInter	0.2007^{-}	0.1432^{-}	0.1077^{-}	0.2748^{-}	0.2523^{-}	0.2533^{-}	0.2943^{-}	0.1098^{-}

TABLE C.4 – Comparaison des performances des différents modèles sur wikIRS78k. Les amélioration/dégradations statistiquement significatives par rapport à BM25-RM3 sont notées par (+/-) avec une p-value < 0.01.

Modèle	P@5	P@10	P@20	nDCG@5	nDCG@10	nDCG@20	nDCG	MAP
ID-Rep	+ 0.7%	– 0.4%	+ 0.2%	– 0.2%	+ 0.5%	+ 0.3%	+ 0.7%	+ 0.5%
CNN-Rep	+ 0.5%	+ 1.1%	+ 1.5%	– 0.3%	+ 0.2%	– 0.1%	– 0.3%	+ 1.5%
LSTM-Rep	+ 0.1%	+ 0.0%	+ 0.4%	– 0.2%	+ 0.4%	+ 1.4%	– 0.1%	+ 0.9%
Transf-Rep	- 0.4%	+ 0.1%	+ 0.3%	+ 0.1%	– 0.6%	+ 0.0%	+ 0.9%	+ 0.2%
ID-SInter	+ 0.1%	+ 1.0%	+ 1.1%	– 0.1%	– 0.1%	+ 0.0%	– 0.7%	+ 0.7%
CNN-SInter	– 0.2%	+ 0.5%	+ 0.3%	– 0.2%	+ 0.4%	– 0.2%	+ 0.0%	+ 0.3%
LSTM-SInter	– 0.5%	+ 0.8%	+ 0.4%	– 0.1%	+ 0.6%	+ 0.0%	+ 0.3%	+ 0.3%
Transf-SInter	+ 0.6%	+ 0.1%	– 0.3%	+ 0.5%	– 0.1%	+ 1.0%	– 0.1%	– 0.2%
ID-GInter	+ 0.6%	– 0.3%	+ 0.4%	+ 0.0%	+ 0.0%	+ 1.3%	+ 0.4%	+ 1.2%
CNN-GInter	– 0.1%	+ 0.3%	+ 0.7%	+ 0.4%	– 1.1%	+ 0.2%	+ 0.2%	+ 0.6%
LSTM-GInter	+ 0.0%	– 0.1%	– 0.7%	+ 0.5%	+ 0.1%	+ 0.7%	– 0.1%	– 0.3%
Transf-GInter	+ 0.2%	+ 0.9%	– 1.0%	+ 0.0%	+ 0.5%	+ 0.0%	– 0.1%	+ 0.1%

TABLE C.5 – Gains de performance obtenus avec l'incorporation de concepts sur la collection WikIR78k.

Modèle	P@5	P@10	P@20	nDCG@5	nDCG@10	nDCG@20	nDCG	MAP
ID-Rep	– 0.5%	+ 0.0%	+ 0.2%	+ 0.9%	+ 0.4%	+ 0.7%	+ 0.0%	+ 0.0%
CNN-Rep	– 0.7%	+ 0.5%	+ 0.5%	– 0.1%	– 0.6%	+ 0.6%	+ 0.0%	+ 0.3%
LSTM-Rep	+ 0.3%	– 0.7%	– 0.9%	+ 0.7%	+ 0.2%	+ 0.5%	+ 0.0%	+ 1.3%
Transf-Rep	+ 0.8%	+ 0.6%	+ 0.9%	+ 0.1%	+ 0.1%	+ 0.3%	+ 0.7%	+ 0.4%
ID-SInter	– 0.7%	+ 0.2%	– 0.2%	– 0.2%	+ 0.4%	– 0.1%	+ 0.3%	+ 0.0%
CNN-SInter	+ 1.1%	+ 0.4%	+ 0.5%	+ 0.4%	+ 0.2%	+ 0.6%	+ 0.1%	+ 0.7%
LSTM-SInter	+ 0.2%	+ 0.5%	+ 0.5%	– 0.1%	+ 0.3%	+ 0.4%	+ 1.0%	– 0.1%
Transf-SInter	+ 0.3%	+ 0.0%	+ 0.6%	+ 0.5%	+ 1.0%	+ 1.2%	+ 0.3%	+ 0.8%
ID-GInter	– 0.2%	+ 0.1%	+ 0.1%	+ 0.0%	+ 0.3%	+ 1.2%	+ 0.1%	+ 0.5%
CNN-GInter	+ 0.5%	+ 0.2%	+ 0.3%	– 0.3%	+ 0.3%	+ 0.2%	+ 0.6%	+ 0.2%
LSTM-GInter	+ 1.2%	+ 0.3%	+ 0.0%	+ 0.1%	+ 0.6%	+ 0.2%	+ 1.0%	+ 1.2%
Transf-GInter	– 0.2%	+ 0.0%	+ 0.0%	+ 0.0%	+ 0.7%	– 0.4%	– 0.9%	+ 0.1%

TABLE C.6 – Gains de performance obtenus avec l'incorporation de concepts sur la collection WikIRS78k.

modèles	P@5	P@10	ndcg@5	ndcg@10	ndcg	MAP
ID-Rep$_{\text{fT}}$	0.1342	0.0978	0.1546	0.1479	0.1720	0.0721
ID-Rep$_{\text{BERT}}$	0.3070	0.2410	0.3790	0.3518	0.3965	0.1804
CNN-Rep$_{\text{fT}}$	0.2545	0.1966	0.3075	0.2896	0.3434	0.1441
CNN-Rep$_{\text{BERT}}$	0.2274	0.1656	0.2646	0.2519	0.2873	0.1296
LSTM-Rep$_{\text{fT}}$	0.2323	0.1750	0.2761	0.2546	0.3090	0.1261
LSTM-Rep$_{\text{BERT}}$	0.2001	0.1537	0.2360	0.2095	0.2648	0.1035
Transf-Rep$_{\text{fT}}$	0.2716	0.2173	0.3408	0.3173	0.3715	0.1516
Transf-Rep$_{\text{BERT}}$	0.2259	0.1786	0.2696	0.2528	0.3004	0.1234
ID-SInter$_{\text{fT}}$	0.3150	0.2429	0.3962	0.3652	0.4290	0.1769
ID-SInter$_{\text{BERT}}$	0.1820	0.1423	0.2404	0.2394	0.2575	0.1029
CNN-SInter$_{\text{fT}}$	0.2380	0.1719	0.2752	0.2693	0.3160	0.1283
CNN-SInter$_{\text{BERT}}$	0.1641	0.1104	0.1789	0.1787	0.2110	0.0778
LSTM-SInter$_{\text{fT}}$	0.2108	0.1570	0.2601	0.2437	0.2772	0.1178
LSTM-SInter$_{\text{BERT}}$	0.1570	0.1140	0.1846	0.1807	0.2108	0.0899
Transf-SInter$_{\text{fT}}$	0.2440	0.1831	0.3050	0.2891	0.3365	0.1400
Transf-SInter$_{\text{BERT}}$	0.1570	0.1140	0.1846	0.1807	0.2108	0.0899
ID-GInter$_{\text{fT}}$	0.2874	0.2201	0.3601	0.3226	0.3841	0.1581
ID-GInter$_{\text{BERT}}$	0.1891	0.1612	0.2342	0.2113	0.2465	0.1001
CNN-GInter$_{\text{fT}}$	0.2164	0.1658	0.2396	0.2480	0.2916	0.1187
CNN-GInter$_{\text{BERT}}$	0.1539	0.1144	0.1900	0.1783	0.2047	0.0844
LSTM-GInter$_{\text{fT}}$	0.1955	0.1511	0.2444	0.2184	0.2395	0.0995
LSTM-GInter$_{\text{BERT}}$	0.1570	0.1197	0.1853	0.1745	0.2109	0.0874
Transf-GInter$_{\text{fT}}$	0.2238	0.1641	0.2758	0.2622	0.3063	0.1262
Transf-GInter$_{\text{BERT}}$	0.1521	0.1209	0.2012	0.1837	0.2159	0.0852

TABLE C.7 – Comparaison de l'utilisation des vecteurs *fastText* (fT) avec les vecteurs BERT sur WikIR78k.

modèles	P@5	P@10	ndcg@5	ndcg@10	ndcg	MAP
ID-Rep$_{\text{fT}}$	0.1281	0.0761	0.1257	0.1270	0.1513	0.0605
ID-Rep$_{\text{BERT}}$	0.3089	0.2437	0.3655	0.3449	0.3651	0.1635
CNN-Rep$_{\text{fT}}$	0.2621	0.2046	0.3178	0.2973	0.3030	0.1356
CNN-Rep$_{\text{BERT}}$	0.2537	0.1956	0.2967	0.2753	0.2847	0.1191
LSTM-Rep$_{\text{fT}}$	0.2409	0.1830	0.2740	0.2633	0.2799	0.1272
LSTM-Rep$_{\text{BERT}}$	0.2173	0.1651	0.2471	0.2448	0.2457	0.1164
Transf-Rep$_{\text{fT}}$	0.2934	0.2287	0.3495	0.3332	0.3356	0.1548
Transf-Rep$_{\text{BERT}}$	0.2643	0.2037	0.3114	0.2999	0.3151	0.1382
ID-SInter$_{\text{fT}}$	0.2657	0.2093	0.3570	0.3281	0.3492	0.1474
ID-SInter$_{\text{BERT}}$	0.2396	0.1992	0.3255	0.3033	0.3284	0.1427
CNN-SInter$_{\text{fT}}$	0.2281	0.1636	0.3061	0.2670	0.3209	0.1146
CNN-SInter$_{\text{BERT}}$	0.1804	0.1382	0.2506	0.2132	0.2519	0.0878
LSTM-SInter$_{\text{fT}}$	0.2062	0.1609	0.2936	0.2567	0.3095	0.1163
LSTM-SInter$_{\text{BERT}}$	0.1646	0.1262	0.2243	0.2063	0.2394	0.0713
Transf-SInter$_{\text{fT}}$	0.2213	0.1608	0.2932	0.2627	0.3099	0.1141
Transf-SInter$_{\text{BERT}}$	0.1814	0.1290	0.2470	0.2293	0.2583	0.0922
ID-GInter$_{\text{fT}}$	0.2407	0.1828	0.3228	0.2871	0.3296	0.1323
ID-GInter$_{\text{BERT}}$	0.1968	0.1473	0.2586	0.2341	0.2718	0.1017
CNN-GInter$_{\text{fT}}$	0.2080	0.1536	0.2761	0.2551	0.2836	0.1132
CNN-GInter$_{\text{BERT}}$	0.1696	0.1209	0.2121	0.1991	0.2238	0.0871
LSTM-GInter$_{\text{fT}}$	0.2026	0.1496	0.2872	0.2526	0.2973	0.1096
LSTM-GInter$_{\text{BERT}}$	0.1626	0.1169	0.2275	0.1944	0.2368	0.0909
Transf-GInter$_{\text{fT}}$	0.2007	0.1432	0.2748	0.2523	0.2943	0.1098
Transf-GInter$_{\text{BERT}}$	0.1551	0.1080	0.2125	0.1926	0.2249	0.0924

TABLE C.8 – Comparaison de l'utilisation des vecteurs *fastText* (fT) avec les vecteurs BERT sur WikIRS78k.

	AP88-89			**LA**			**FT91-94**		
Modèle	P@5	ndcg@5	MAP	P@5	ndcg@5	MAP	P@5	ndcg@5	MAP
ID-Rep	0.1606	0.1636	0.1095	0.1102	0.1110	0.0663	0.1015	0.1144	0.0101
CNN-Rep	0.1601	0.1686	0.1070	0.1035	0.1051	0.0729	0.1048	0.1203	0.0281
LSTM-Rep	0.1588	0.1601	0.1074	0.1203	0.1046	0.0770	0.1150	0.1138	0.0143
Transf-Rep	0.1715	0.1747	0.1120	0.1138	0.1172	0.0761	0.1090	0.1229	0.0224
ID-Sinter	0.4072	0.3941	0.2412	0.2930	0.3327	0.1981	0.3160	0.3285	0.2266
CNN-Sinter	0.2739	0.2777	0.1660	0.1932	0.2365	0.1435	0.2131	0.2184	0.1492
LSTM-Sinter	0.2495	0.2470	0.1567	0.1808	0.2017	0.1187	0.2036	0.2010	0.1408
Transf-Sinter	0.2884	0.2840	0.1726	0.2157	0.2412	0.1393	0.2303	0.2408	0.1639
ID-Ginter	0.3356	0.3268	0.2007	0.2318	0.2793	0.1652	0.2636	0.2795	0.1923
CNN-Ginter	0.2288	0.2263	0.1386	0.1490	0.1986	0.1106	0.1896	0.1921	0.1359
LSTM-Ginter	0.2051	0.2131	0.1215	0.1457	0.1628	0.0986	0.1627	0.1691	0.1221
Transf-Ginter	0.2351	0.2330	0.1417	0.1557	0.2020	0.1197	0.1800	0.1890	0.1294

TABLE C.9 – Performances sur les collections TREC de nos modèles. Les amélioration/dégradations statistiquement significatives par rapport à BM25-RM3 sont notées par (+/-) avec une p-value < 0.01.

	AP88-89			**LA**			**FT91-94**		
Modèle	P@5	ndcg@5	MAP	P@5	ndcg@5	MAP	P@5	ndcg@5	MAP
ID-Rep$_{\text{WikIR}}$	0.1631	0.1617	0.0961	0.1124	0.1214	0.0812	0.1163	0.1283	0.0532
CNN-Rep$_{\text{WikIR}}$	0.4376	0.4260	0.2637	0.2949	0.3256	0.2219	0.3130	0.3438	0.2230
LSTM-Rep$_{\text{WikIR}}$	0.4120	0.3988	0.2471	0.2741	0.3187	0.2191	0.3074	0.3211	0.2132
Transf-Rep$_{\text{WikIR}}$	0.4649	0.4518	0.2820	0.3135	0.3516	0.2365	0.3390	0.3631	0.2444
ID-Sinter$_{\text{WikIR}}$	0.4863	0.4613	0.3073	0.3410	0.3853	0.2374	0.3575	0.3791	0.2675
CNN-Sinter$_{\text{WikIR}}$	0.3982	0.3978	0.2518	0.2857	0.3260	0.2000	0.2963	0.3134	0.2189
LSTM-Sinter$_{\text{WikIR}}$	0.3874	0.3752	0.2510	0.2609	0.3015	0.1870	0.2819	0.2910	0.2083
Transf-Sinter$_{\text{WikIR}}$	0.4070	0.3908	0.2561	0.2835	0.3164	0.2090	0.2946	0.3278	0.2334
ID-Ginter$_{\text{WikIR}}$	0.4342	0.4114	0.2787	0.3122	0.3420	0.2153	0.3230	0.3414	0.2387
CNN-Ginter$_{\text{WikIR}}$	0.3763	0.3594	0.2414	0.2700	0.2908	0.1858	0.2781	0.2991	0.2193
LSTM-Ginter$_{\text{WikIR}}$	0.3284	0.3131	0.2131	0.2534	0.2678	0.1649	0.2474	0.2561	0.1790
Transf-Ginter$_{\text{WikIR}}$	0.3364	0.3083	0.2132	0.2419	0.2564	0.1614	0.2446	0.2557	0.1793

TABLE C.10 – Performances sur les collections TREC de nos modèles pré-entraînés sur wikIR78k. Les amélioration/-dégradations statistiquement significatives par rapport à BM25-RM3 sont notées par (+/-) avec une p-value < 0.01.

	AP88-89			LA			FT91-94		
Modèle	P@5	ndcg@5	MAP	P@5	ndcg@5	MAP	P@5	ndcg@5	MAP
ID-Rep$_{\text{WikIRS}}$	0.1612	0.1568	0.0942	0.1127	0.1158	0.0784	0.1111	0.1205	0.0452
CNN-Rep$_{\text{WikIRS}}$	0.5372	0.5465	0.3420	0.3944	0.4050	0.2639	0.4221	0.4353	0.2818
LSTM-Rep$_{\text{WikIRS}}$	0.5080	0.5110	0.3056	0.3569	0.3858	0.2493	0.3996	0.3973	0.2588
Transf-Rep$_{\text{WikIRS}}$	0.5619	0.5692	0.3522	0.4121	0.4192	0.2800	0.4333	0.4448	0.2928
ID-Sinter$_{\text{WikIRS}}$	0.5089	0.4768	0.3188	0.3469	0.4073	0.2498	0.3744	0.3840	0.2789
CNN-Sinter$_{\text{WikIRS}}$	0.4714	0.4478	0.2973	0.3213	0.3738	0.2303	0.3476	0.3610	0.2628
LSTM-Sinter$_{\text{WikIRS}}$	0.4329	0.3923	0.2660	0.2959	0.3430	0.1983	0.3076	0.3201	0.2305
Transf-Sinter$_{\text{WikIRS}}$	0.4831	0.4576	0.3000	0.3335	0.3855	0.2430	0.3608	0.3649	0.2593
ID-Ginter$_{\text{WikIRS}}$	0.4855	0.4644	0.3101	0.3397	0.3850	0.2497	0.3650	0.3897	0.2624
CNN-Ginter$_{\text{WikIRS}}$	0.4160	0.4019	0.2701	0.2904	0.3182	0.2035	0.3037	0.3357	0.2421
LSTM-Ginter$_{\text{WikIRS}}$	0.3576	0.3439	0.2335	0.2813	0.2938	0.1897	0.2709	0.2861	0.1993
Transf-Ginter$_{\text{WikIRS}}$	0.3613	0.3433	0.2391	0.2710	0.2895	0.1818	0.2670	0.2867	0.2036

TABLE C.11 – Performances sur les collections TREC de nos modèles pré-entraînés sur wikIRS78k. Les amélioration/dégradations statistiquement significatives par rapport à BM25-RM3 sont notées par (+/-) avec une p-value $<$ 0.01.

Annexe D

Bibliographie

Bibliographie

Amr Ahmed, Nino Shervashidze, Shravan M. Narayanamurthy, Vanja Josifovski, et Alexander J. Smola. Distributed large-scale natural graph factorization. In Daniel Schwabe, Virgílio A. F. Almeida, Hartmut Glaser, Ricardo Baeza-Yates, et Sue B. Moon, editors, *22nd International World Wide Web Conference, WWW '13, Rio de Janeiro, Brazil, May 13-17, 2013*, pages 37–48. International World Wide Web Conferences Steering Committee / ACM, 2013. ISBN 978-1-4503-2035-1. doi : 10.1145/2488388.2488393. URL https://doi.org/10.1145/2488388.2488393. 33

Mohannad Almasri, Catherine Berrut, et Jean-Pierre Chevallet. A comparison of deep learning based query expansion with pseudo-relevance feedback and mutual information. In Nicola Ferro, Fabio Crestani, Marie-Francine Moens, Josiane Mothe, Fabrizio Silvestri, Giorgio Maria Di Nunzio, Claudia Hauff, et Gianmaria Silvello, editors, *Advances in Information Retrieval - 38th European Conference on IR Research, ECIR 2016, Padua, Italy, March 20-23, 2016. Proceedings*, volume 9626 of *Lecture Notes in Computer Science*, pages 709–715. Springer, 2016. ISBN 978-3-319-30670-4. doi : 10.1007/978-3-319-30671-1_57. URL https://doi.org/10.1007/978-3-319-30671-1_57. 25

Massih-Reza Amini et Eric Gaussier. *Recherche d'information : Applications, modèles et algorithmes-Fouille de données, décisionnel et big data*. Editions Eyrolles, 2013. 15, 21

Ioannis Arapakis, Xiao Bai, et Berkant Barla Cambazoglu. Impact of response latency on user behavior in web search. In Geva et al. (2014), pages 103–112. ISBN 978-1-4503-2257-7. doi : 10.1145/2600428.2609627. URL https://doi.org/10.1145/2600428.2609627. 6, 78

Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, et Zachary G. Ives. Dbpedia : A nucleus for a web of open data. In Karl Aberer,

Key-Sun Choi, Natasha Fridman Noy, Dean Allemang, Kyung-Il Lee, Lyndon J. B. Nixon, Jennifer Golbeck, Peter Mika, Diana Maynard, Riichiro Mizoguchi, Guus Schreiber, et Philippe Cudré-Mauroux, editors, *The Semantic Web, 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11-15, 2007*, volume 4825 of *Lecture Notes in Computer Science*, pages 722–735. Springer, 2007. ISBN 978-3-540-76297-3. doi : 10.1007/978-3-540-76298-0_52. URL https://doi.org/10.1007/978-3-540-76298-0_52. 116

Lei Jimmy Ba, Jamie Ryan Kiros, et Geoffrey E. Hinton. Layer normalization. *CoRR*, abs/1607.06450, 2016. URL http://arxiv.org/abs/1607.06450. 54

Ricardo Baeza-Yates et Berthier A. Ribeiro-Neto. *Modern Information Retrieval - the concepts and technology behind search, Second edition*, page 21. Pearson Education Ltd., Harlow, England, 2011. ISBN 978-0-321-41691-9. URL http://www.mir2ed.org/. 91, 155

Thomas C Bagg et Mary Elizabeth Stevens. *Information Selection Systems Retrieval Replica Copies : A-state-of-the-art Report*, volume 13. US Department of Commerce, National Bureau of Standards, 1961. 13

Dzmitry Bahdanau, Kyunghyun Cho, et Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In Bengio et LeCun (2015). URL http://arxiv.org/abs/1409.0473. 40

Yoshua Bengio. Neural net language models. *Scholarpedia*, 3(1) :3881, 2008. doi : 10.4249/scholarpedia.3881. URL https://doi.org/10.4249/scholarpedia.3881. 35

Yoshua Bengio et Yann LeCun, editors. *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL https://iclr.cc/archive/www/doku.php%3Fid=iclr2015:accepted-main.html. 172, 182

Yoshua Bengio, Patrice Y. Simard, et Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Trans. Neural Networks*, 5 (2) :157–166, 1994. doi : 10.1109/72.279181. URL https://doi.org/10.1109/72.279181. 41

Yoshua Bengio, Réjean Ducharme, et Pascal Vincent. A neural probabilistic language model. In Todd K. Leen, Thomas G. Dietterich, et Volker Tresp,

editors, *Advances in Neural Information Processing Systems 13, Papers from Neural Information Processing Systems (NIPS) 2000, Denver, CO, USA*, pages 932–938. MIT Press, 2000. URL http://papers.nips.cc/paper/1839-a-neural-probabilistic-language-model. 37

Steven Bird. NLTK : the natural language toolkit. In Nicoletta Calzolari, Claire Cardie, et Pierre Isabelle, editors, *ACL 2006, 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference, Sydney, Australia, 17-21 July 2006*. The Association for Computer Linguistics, 2006. doi : 10.3115/1225403.1225421. URL https://www.aclweb.org/anthology/P06-4018/. 15

Olivier Bodenreider. The unified medical language system (UMLS) : integrating biomedical terminology. *Nucleic Acids Res.*, 32(Database-Issue) :267–270, 2004. doi : 10.1093/nar/gkh061. URL https://doi.org/10.1093/nar/gkh061. 113

Piotr Bojanowski, Edouard Grave, Armand Joulin, et Tomas Mikolov. Enriching word vectors with subword information. *Trans. Assoc. Comput. Linguistics*, 5 : 135–146, 2017. URL https://transacl.org/ojs/index.php/tacl/article/view/999. 39

Tolga Bolukbasi, Kai-Wei Chang, James Y. Zou, Venkatesh Saligrama, et Adam Tauman Kalai. Man is to computer programmer as woman is to homemaker ? debiasing word embeddings. In Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, et Roman Garnett, editors, *Advances in Neural Information Processing Systems 29 : Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 4349–4357, 2016. URL http://papers.nips.cc/paper/6228-man-is-to-computer-programmer-as-woman-is-to-homemaker-debiasing-word-embeddings. 118, 140

Antoine Bordes, Nicolas Usunier, Alberto García-Durán, Jason Weston, et Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In Burges et al. (2013), pages 2787–2795. URL http://papers.nips.cc/paper/5071-translating-embeddings-for-modeling-multi-relational-data. 33

Vera Boteva, Demian Gholipour, Artem Sokolov, et Stefan Riezler. A full-text learning to rank dataset for medical information retrieval. *Proceedings of the 38th European Conference on Information Retrieval*, 2016. URL http://www.cl.uni-heidelberg.de/~riezler/publications/papers/ECIR2016.pdf. 113

Lila Boualili, Jose G. Moreno, et Mohand Boughanem. Markedbert : Integrating traditional IR cues in pre-trained language models for passage retrieval. In Huang et al. (2020), pages 1977–1980. ISBN 978-1-4503-8016-4. doi : 10.1145/3397271.3401194. URL https://doi.org/10.1145/3397271.3401194. 127

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, et Dario Amodei. Language models are few-shot learners. *CoRR*, abs/2005.14165, 2020. URL https://arxiv.org/abs/2005.14165. 57

Christopher J. C. Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, et Gregory N. Hullender. Learning to rank using gradient descent. In Luc De Raedt et Stefan Wrobel, editors, *Machine Learning, Proceedings of the Twenty-Second International Conference (ICML 2005), Bonn, Germany, August 7-11, 2005*, volume 119 of *ACM International Conference Proceeding Series*, pages 89–96. ACM, 2005. ISBN 1-59593-180-5. doi : 10.1145/1102351.1102363. URL https://doi.org/10.1145/1102351.1102363. 64

Christopher J. C. Burges, Léon Bottou, Zoubin Ghahramani, et Kilian Q. Weinberger, editors. *Advances in Neural Information Processing Systems 26 : 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*, 2013. URL http://papers.nips.cc/book/advances-in-neural-information-processing-systems-26-2013. 173, 184

Jill Burstein, Christy Doran, et Thamar Solorio, editors. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics : Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, 2019. Association for Computational Linguistics. ISBN 978-1-950737-13-0. URL https://www.aclweb.org/anthology/volumes/N19-1/. 176, 188

Vannevar Bush. As we may think. *The atlantic monthly*, 176(1) :101–108, 1945. URL http://www.theatlantic.com/unbound/flashbks/computer/bushf.htm. 13

F. Can et E. A. Ozkarahan. Computation of term/document discrimination values by use of the cover coefficient concept. *JASIS*, 38(3) :171–183, 1987. 132

Iván Cantador, Max Chevalier, Massimo Melucci, et Josiane Mothe, editors. *Proceedings of the Joint Conference of the Information Retrieval Communities in Europe (CIRCLE 2020), Samatan, Gers, France, July 6-9, 2020*, volume 2621 of *CEUR Workshop Proceedings*, 2020. CEUR-WS.org. URL http://ceur-ws.org/Vol-2621. 177

Olivier Chapelle et Yi Chang. Yahoo ! learning to rank challenge overview. In Olivier Chapelle, Yi Chang, et Tie-Yan Liu, editors, *Proceedings of the Yahoo ! Learning to Rank Challenge, held at ICML 2010, Haifa, Israel, June 25, 2010*, volume 14 of *JMLR Proceedings*, pages 1–24. JMLR.org, 2011. URL http://proceedings.mlr.press/v14/chapelle11a.html. 83

Wei Chen, Tie-Yan Liu, Yanyan Lan, Zhiming Ma, et Hang Li. Ranking measures and loss functions in learning to rank. In Yoshua Bengio, Dale Schuurmans, John D. Lafferty, Christopher K. I. Williams, et Aron Culotta, editors, *Advances in Neural Information Processing Systems 22 : 23rd Annual Conference on Neural Information Processing Systems 2009. Proceedings of a meeting held 7-10 December 2009, Vancouver, British Columbia, Canada*, pages 315–323. Curran Associates, Inc., 2009. ISBN 9781615679119. URL http://papers.nips.cc/paper/3708-ranking-measures-and-loss-functions-in-learning-to-rank. 64

Kevyn Collins-Thompson, Qiaozhu Mei, Brian D. Davison, Yiqun Liu, et Emine Yilmaz, editors. *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval, SIGIR 2018, Ann Arbor, MI, USA, July 08-12, 2018*, 2018. ACM. URL http://dl.acm.org/citation.cfm?id=3209978. 177, 179, 187, 192

Alexis Conneau et Guillaume Lample. Cross-lingual language model pretraining. In Wallach et al. (2019), pages 7057–7067. URL http://papers.nips.cc/paper/8928-cross-lingual-language-model-pretraining. 151

Jerome T. Connor, R. Douglas Martin, et Les E. Atlas. Recurrent neural networks and robust time series prediction. *IEEE Trans. Neural Networks*, 5(2) :240–254, 1994. doi : 10.1109/72.279188. URL https://doi.org/10.1109/72.279188. 40

Nick Craswell, Bhaskar Mitra, Emine Yilmaz, Daniel Campos, et Ellen M. Voorhees. Overview of the TREC 2019 deep learning track. *CoRR*, abs/2003.07820, 2020. URL https://arxiv.org/abs/2003.07820. 84

W. Bruce Croft et C. J. van Rijsbergen, editors. *Proceedings of the 17th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval. Dublin, Ireland, 3-6 July 1994 (Special Issue of the SIGIR Forum)*, 1994. ACM/Springer. ISBN 978-3-540-19889-5. doi : 10.1007/978-1-4471-2099-5. URL https://doi.org/10.1007/978-1-4471-2099-5. 180, 186, 187

Zhuyun Dai et Jamie Callan. Deeper text understanding for IR with contextual neural language modeling. In Piwowarski et al. (2019), pages 985–988. ISBN 978-1-4503-6172-9. doi : 10.1145/3331184.3331303. URL https://doi.org/10.1145/3331184.3331303. 126, 127

Zhuyun Dai, Chenyan Xiong, Jamie Callan, et Zhiyuan Liu. Convolutional neural networks for soft-matching n-grams in ad-hoc search. In Yi Chang, Chengxiang Zhai, Yan Liu, et Yoelle Maarek, editors, *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining, WSDM 2018, Marina Del Rey, CA, USA, February 5-9, 2018*, pages 126–134. ACM, 2018. ISBN 978-1-4503-5581-0. doi : 10.1145/3159652.3159659. URL https://doi.org/10.1145/3159652.3159659. 5, 74

Jeffrey Dalton, Laura Dietz, et James Allan. Entity query feature expansion using knowledge base links. In Geva et al. (2014), pages 365–374. ISBN 978-1-4503-2257-7. doi : 10.1145/2600428.2609628. URL https://doi.org/10.1145/2600428.2609628. 25, 26

Mostafa Dehghani, Hamed Zamani, Aliaksei Severyn, Jaap Kamps, et W. Bruce Croft. Neural ranking models with weak supervision. In Kando et al. (2017), pages 65–74. ISBN 978-1-4503-5022-8. doi : 10.1145/3077136.3080832. URL https://doi.org/10.1145/3077136.3080832. 83, 84

Jacob Devlin, Ming-Wei Chang, Kenton Lee, et Kristina Toutanova. BERT : pre-training of deep bidirectional transformers for language understanding. In Burstein et al. (2019), pages 4171–4186. ISBN 978-1-950737-13-0. doi : 10.18653/v1/n19-1423. URL https://doi.org/10.18653/v1/n19-1423. 5, 48, 49, 54, 83, 126, 151

Jeffrey L. Elman. Finding structure in time. *Cogn. Sci.*, 14(2) :179–211, 1990. doi : 10.1207/s15516709cog1402_1. URL https://doi.org/10.1207/s15516709cog1402_1. 40

Yixing Fan, Jiafeng Guo, Yanyan Lan, Jun Xu, Chengxiang Zhai, et Xueqi Cheng. Modeling diverse relevance patterns in ad-hoc retrieval. In Collins-Thompson et al. (2018), pages 375–384. doi : 10.1145/3209978.3209980. URL https://doi.org/10.1145/3209978.3209980. 5, 26, 84

H. Fang, T. Tao, et C. Zhai. A formal study of information retrieval heuristics. In *SIGIR 2004 : Sheffield, UK, July 25-29, 2004*, pages 49–56, 2004. 134

Jibril Frej, Philippe Mulhem, Didier Schwab, et Jean-Pierre Chevallet. Learning term discrimination. In Huang et al. (2020), pages 1993–1996. ISBN 978-1-4503-8016-4. doi : 10.1145/3397271.3401211. URL https://doi.org/10.1145/3397271.3401211. 9, 133

Jibril Frej, Didier Schwab, et Jean-Pierre Chevallet. Knowledge based transformer model for information retrieval. In Cantador et al. (2020). URL http://ceur-ws.org/Vol-2621/CIRCLE20_05.pdf. 9, 106

Jibril Frej, Didier Schwab, et Jean-Pierre Chevallet. MLWIKIR : A python toolkit for building large-scale wikipedia-based information retrieval datasets in chinese, english, french, italian, japanese, spanish and more. In Cantador et al. (2020). URL http://ceur-ws.org/Vol-2621/CIRCLE20_22.pdf. 8, 150

Jibril Frej, Didier Schwab, et Jean-Pierre Chevallet. WIKIR : A python toolkit for building a large-scale wikipedia-based english information retrieval dataset. In Nicoletta Calzolari, Frédéric Béchet, Philippe Blache, Khalid Choukri, Christopher Cieri, Thierry Declerck, Sara Goggi, Hitoshi Isahara, Bente Maegaard, Joseph Mariani, Hélène Mazo, Asunción Moreno, Jan Odijk, et Stelios Piperidis, editors, *Proceedings of The 12th Language Resources and Evaluation Conference, LREC 2020, Marseille, France, May 11-16, 2020*, pages 1926–1933. European Language Resources Association, 2020d. ISBN 979-10-95546-34-4. URL https://www.aclweb.org/anthology/2020.lrec-1.237/. 8, 84

Jibril FREJ, Didier Schwab, et Jean-Pierre Chevallet. Modèle Transformer à base de Connaissances pour la Recherche d'Information dans des Domaines Spécialisés. *Extraction et Gestion des Connaissances (EGC)*, January 2020. URL https://hal.archives-ouvertes.fr/hal-02474706. 9, 106

K. Fukushima. Neural network model for a mechanism of pattern recognition unaffected by shift in position-neocognitron-. *IEICE Technical Report, A*, 62 (10) :658–665, 1979. URL https://ci.nii.ac.jp/naid/10026558204/en/. 43

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, et Yann N. Dauphin. Convolutional sequence to sequence learning. In Doina Precup et Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 1243–1252. PMLR, 2017. URL http://proceedings.mlr.press/v70/gehring17a.html. 52

Shlomo Geva, Andrew Trotman, Peter Bruza, Charles L. A. Clarke, et Kalervo Järvelin, editors. *The 37th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '14, Gold Coast , QLD, Australia - July 06 - 11, 2014*, 2014. ACM. ISBN 978-1-4503-2257-7. doi : 10.1145/2600428. URL https://doi.org/10.1145/2600428. 171, 176

Zoubin Ghahramani, Max Welling, Corinna Cortes, Neil D. Lawrence, et Kilian Q. Weinberger, editors. *Advances in Neural Information Processing Systems 27 : Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, 2014. URL http://papers.nips.cc/book/advances-in-neural-information-processing-systems-27-2014. 180, 188

Ian Goodfellow, Yoshua Bengio, et Aaron Courville. *Deep learning*. The MIT Press, 2016. URL http://cds.cern.ch/record/2244405. 5, 78, 83

Alex Graves, Abdel-rahman Mohamed, et Geoffrey E. Hinton. Speech recognition with deep recurrent neural networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2013, Vancouver, BC, Canada, May 26-31, 2013*, pages 6645–6649. IEEE, 2013. doi : 10.1109/ICASSP.2013.6638947. URL https://doi.org/10.1109/ICASSP.2013.6638947. 40

Aditya Grover et Jure Leskovec. node2vec : Scalable feature learning for networks. In Balaji Krishnapuram, Mohak Shah, Alexander J. Smola, Charu C. Aggarwal, Dou Shen, et Rajeev Rastogi, editors, *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, pages 855–864. ACM, 2016. ISBN 978-1-4503-4232-2. doi : 10.1145/2939672.2939754. URL https://doi.org/10.1145/2939672.2939754. 33

Jiafeng Guo, Yixing Fan, Qingyao Ai, et W. Bruce Croft. A deep relevance matching model for ad-hoc retrieval. In Snehasis Mukhopadhyay, ChengXiang Zhai, Elisa Bertino, Fabio Crestani, Javed Mostafa, Jie Tang, Luo Si, Xiaofang

Zhou, Yi Chang, Yunyao Li, et Parikshit Sondhi, editors, *Proceedings of the 25th ACM International Conference on Information and Knowledge Management, CIKM 2016, Indianapolis, IN, USA, October 24-28, 2016*, pages 55–64. ACM, 2016. ISBN 978-1-4503-4073-1. doi : 10.1145/2983323.2983769. URL https://doi.org/10.1145/2983323.2983769. 61, 69, 70

Jiafeng Guo, Yixing Fan, Xiang Ji, et Xueqi Cheng. Matchzoo : A learning, practicing, and developing system for neural text matching. In Piwowarski et al. (2019), pages 1297–1300. ISBN 978-1-4503-6172-9. doi : 10.1145/3331184.3331403. URL https://doi.org/10.1145/3331184.3331403. 66, 91, 93, 96

Jiafeng Guo, Yixing Fan, Liang Pang, Liu Yang, Qingyao Ai, Hamed Zamani, Chen Wu, W. Bruce Croft, et Xueqi Cheng. A deep look into neural ranking models for information retrieval. *CoRR*, abs/1903.06902, 2019b. URL http://arxiv.org/abs/1903.06902. 5, 59, 63, 65

Christophe Van Gysel et Maarten de Rijke. Pytrec_eval : An extremely fast python interface to trec_eval. In Collins-Thompson et al. (2018), pages 873–876. doi : 10.1145/3209978.3210065. URL https://doi.org/10.1145/3209978.3210065. 96

Shuguang Han, Xuanhui Wang, Mike Bendersky, et Marc Najork. Learning-to-Rank with BERT in TF-Ranking. *arXiv e-prints*, art. arXiv :2004.08476, April 2020. 5

Zellig S. Harris. Distributional structure. *WORD*, 10(2-3) :146–162, 1954. doi : 10.1080/00437956.1954.11659520. URL https://doi.org/10.1080/00437956.1954.11659520. 35

K He, Xiangyu Zhang, Shaoqing Ren, et Jian Sun, editors. *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, 2016a. IEEE Computer Society. ISBN 978-1-4673-8851-1. URL https://ieeexplore.ieee.org/xpl/conhome/7776647/proceeding. 179, 191

Kaiming He, Xiangyu Zhang, Shaoqing Ren, et Jian Sun. Deep residual learning for image recognition. In He et al. (2016a), pages 770–778. ISBN 978-1-4673-8851-1. doi : 10.1109/CVPR.2016.90. URL https://doi.org/10.1109/CVPR.2016.90. 54

William R. Hersh, Chris Buckley, T. J. Leone, et David H. Hickam. OHSUMED : an interactive retrieval evaluation and new large test collection for research. In

Croft et van Rijsbergen (1994), pages 192–201. ISBN 978-3-540-19889-5. doi : 10.1007/978-1-4471-2099-5_20. URL https://doi.org/10.1007/978-1-4471-2099-5_20. 123

Birger Hjørland. Towards a theory of aboutness, subject, topicality, theme, domain, field, content ... and relevance. *J. Assoc. Inf. Sci. Technol.*, 52(9) :774–778, 2001. doi : 10.1002/asi.1131. URL https://doi.org/10.1002/asi.1131. 85

Sepp Hochreiter et Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8) :1735–1780, 1997. doi : 10.1162/neco.1997.9.8.1735. URL https://doi.org/10.1162/neco.1997.9.8.1735. 41, 42

Baotian Hu, Zhengdong Lu, Hang Li, et Qingcai Chen. Convolutional neural network architectures for matching natural language sentences. In Ghahramani et al. (2014), pages 2042–2050. URL http://papers.nips.cc/paper/5550-convolutional-neural-network-architectures-for-matching-natural-language-sentences. 66, 111

Jimmy Huang, Yi Chang, Xueqi Cheng, Jaap Kamps, Vanessa Murdock, Ji-Rong Wen, et Yiqun Liu, editors. *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval, SIGIR 2020, Virtual Event, China, July 25-30, 2020*, 2020. ACM. ISBN 978-1-4503-8016-4. doi : 10.1145/3397271. URL https://doi.org/10.1145/3397271. 174, 177

Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, et Larry P. Heck. Learning deep structured semantic models for web search using clickthrough data. In Qi He, Arun Iyengar, Wolfgang Nejdl, Jian Pei, et Rajeev Rastogi, editors, *22nd ACM International Conference on Information and Knowledge Management, CIKM'13, San Francisco, CA, USA, October 27 - November 1, 2013*, pages 2333–2338. ACM, 2013. ISBN 978-1-4503-2263-8. doi : 10.1145/2505515.2505665. URL https://doi.org/10.1145/2505515.2505665. 60

Alekseui Grigorevich Ivakhnenko et Valentin Grigor'evich Lapa. *Cybernetic predicting devices*. CCM Information Corporation, 1966. 29

Nasreen Abdul Jaleel, James Allan, W. Bruce Croft, Fernando Diaz, Leah S. Larkey, Xiaoyan Li, Mark D. Smucker, et Courtney Wade. Umass at TREC 2004 : Novelty and HARD. In Ellen M. Voorhees et Lori P. Buckland, editors, *Proceedings of the Thirteenth Text REtrieval Conference, TREC 2004, Gaithersburg, Maryland, USA, November 16-19, 2004*, volume 500-261 of *NIST*

Special Publication. National Institute of Standards and Technology (NIST), 2004. URL http://trec.nist.gov/pubs/trec13/papers/umass.novelty.hard.pdf. 21

Kalervo Järvelin et Jaana Kekäläinen. Cumulated gain-based evaluation of IR techniques. *ACM Trans. Inf. Syst.*, 20(4) :422–446, 2002. doi : 10.1145/582415.582418. URL http://doi.acm.org/10.1145/582415.582418. 24

Jimmy, Guido Zuccon, et Bevan Koopman. Payoffs and pitfalls in using knowledge-bases for consumer health search. *Inf. Retr. J.*, 22(3-4) :350–394, 2019. doi : 10.1007/s10791-018-9344-z. URL https://doi.org/10.1007/s10791-018-9344-z. 26, 105

Thorsten Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, July 23-26, 2002, Edmonton, Alberta, Canada*, pages 133–142. ACM, 2002. ISBN 1-58113-567-X. doi : 10.1145/775047.775067. URL https://doi.org/10.1145/775047.775067. 33

Karen Spärck Jones. A statistical interpretation of term specificity and its application in retrieval. *J. Documentation*, 28(1) :11–21, 1972. doi : 10.1108/00220410410560573. URL https://doi.org/10.1108/00220410410560573. 18

Noriko Kando, Tetsuya Sakai, Hideo Joho, Hang Li, Arjen P. de Vries, et Ryen W. White, editors. *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, Shinjuku, Tokyo, Japan, August 7-11, 2017*, 2017. ACM. ISBN 978-1-4503-5022-8. doi : 10.1145/3077136. URL https://doi.org/10.1145/3077136. 176, 190

Liadh Kelly, Lorraine Goeuriot, Hanna Suominen, Tobias Schreck, Gondy Leroy, Danielle L. Mowery, Sumithra Velupillai, Wendy W. Chapman, David Martínez, Guido Zuccon, et João R. M. Palotti. Overview of the share/clef ehealth evaluation lab 2014. In Evangelos Kanoulas, Mihai Lupu, Paul D. Clough, Mark Sanderson, Mark M. Hall, Allan Hanbury, et Elaine G. Toms, editors, *Information Access Evaluation. Multilinguality, Multimodality, and Interaction - 5th International Conference of the CLEF Initiative, CLEF 2014, Sheffield, UK, September 15-18, 2014. Proceedings*, volume 8685 of *Lecture Notes in Computer Science*, pages 172–191. Springer, 2014. ISBN 978-3-319-11381-4. doi : 10.1007/978-3-319-11382-1_17. URL https://doi.org/10.1007/978-3-319-11382-1_17. 123

Diederik P. Kingma et Jimmy Ba. Adam : A method for stochastic optimization. In Bengio et LeCun (2015). URL http://arxiv.org/abs/1412.6980. 32, 96

Thomas N. Kipf et Max Welling. Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=SJU4ayYgl. 35, 151

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, et Radu Soricut. ALBERT : A lite BERT for self-supervised learning of language representations. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=H1eA7AEtvS. 48, 151

Yann LeCun et Yoshua Bengio. *Convolutional Networks for Images, Speech, and Time Series*, page 255–258. MIT Press, Cambridge, MA, USA, 1998. ISBN 0262511029. 66

Yann LeCun, Yoshua Bengio, et Geoffrey E. Hinton. Deep learning. *Nature*, 521 (7553) :436–444, 2015. doi : 10.1038/nature14539. URL https://doi.org/10.1038/nature14539. 30, 32, 47, 155, 156

Hang Li, Tie-Yan Liu, et ChengXiang Zhai. Learning to rank for information retrieval (LR4IR 2008). *SIGIR Forum*, 42(2) :76–79, 2008. doi : 10.1145/1480506.1480519. URL https://doi.org/10.1145/1480506.1480519. 62

Jimmy Lin, Yulu Wang, Miles Efron, et Garrick Sherman. Overview of the TREC-2014 microblog track. In Ellen M. Voorhees et Angela Ellis, editors, *Proceedings of The Twenty-Third Text REtrieval Conference, TREC 2014, Gaithersburg, Maryland, USA, November 19-21, 2014*, volume 500-308 of *NIST Special Publication*. National Institute of Standards and Technology (NIST), 2014. URL https://trec.nist.gov/pubs/trec23/papers/overview-microblog.pdf. 77

Jimmy Lin, Rodrigo Nogueira, et Andrew Yates. Pretrained transformers for text ranking : BERT and beyond. *CoRR*, abs/2010.06467, 2020. URL https://arxiv.org/abs/2010.06467. 77

Tie-Yan Liu. Learning to rank for information retrieval. *Found. Trends Inf. Retr.*, 3 (3) :225–331, 2009. doi : 10.1561/1500000016. URL https://doi.org/10.1561/1500000016. 155

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, et Veselin Stoyanov. Roberta : A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019. URL http://arxiv.org/abs/1907.11692. 151

Thang Luong, Richard Socher, et Christopher D. Manning. Better word representations with recursive neural networks for morphology. In Julia Hockenmaier et Sebastian Riedel, editors, *Proceedings of the Seventeenth Conference on Computational Natural Language Learning, CoNLL 2013, Sofia, Bulgaria, August 8-9, 2013*, pages 104–113. ACL, 2013. ISBN 978-1-937284-70-1. URL https://www.aclweb.org/anthology/W13-3512/. xvii, 40

Thang Luong, Hieu Pham, et Christopher D. Manning. Effective approaches to attention-based neural machine translation. In Lluís Màrquez, Chris Callison-Burch, Jian Su, Daniele Pighin, et Yuval Marton, editors, *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pages 1412–1421. The Association for Computational Linguistics, 2015. ISBN 978-1-941643-32-7. doi : 10.18653/v1/d15-1166. URL https://doi.org/10.18653/v1/d15-1166. 49

Jitendra Malik et Pietro Perona. Preattentive texture discrimination with early vision mechanisms. *J. Opt. Soc. Am. A*, 7(5) :923–932, May 1990. doi : 10.1364/JOSAA.7.000923. URL http://josaa.osa.org/abstract.cfm?URI=josaa-7-5-923. 30

Christopher D. Manning, Prabhakar Raghavan, et Hinrich Schütze. *Introduction to information retrieval*. Cambridge University Press, 2008. ISBN 978-0-521-86571-5. doi : 10.1017/CBO9780511809071. URL https://nlp.stanford.edu/IR-book/pdf/irbookprint.pdf. 22, 23, 85

M. E. Maron et J. L. Kuhns. On relevance, probabilistic indexing and information retrieval. *J. ACM*, 7(3) :216–244, 1960. doi : 10.1145/321033.321035. URL https://doi.org/10.1145/321033.321035. 18

Tomas Mikolov, Kai Chen, Greg Corrado, et Jeffrey Dean. Efficient estimation of word representations in vector space. In Yoshua Bengio et Yann LeCun, editors, *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, 2013a. URL http://arxiv.org/abs/1301.3781. 36, 37, 96

Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, et Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In Burges et al. (2013), pages 3111–3119. URL http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality. 37

Bhaskar Mitra et Nick Craswell. An introduction to neural information retrieval. *Found. Trends Inf. Retr.*, 13(1) :1–126, 2018. doi : 10.1561/1500000061. URL https://doi.org/10.1561/1500000061. 37

Bhaskar Mitra, Fernando Diaz, et Nick Craswell. Learning to match using local and distributed representations of text for web search. In Rick Barrett, Rick Cummings, Eugene Agichtein, et Evgeniy Gabrilovich, editors, *Proceedings of the 26th International Conference on World Wide Web, WWW 2017, Perth, Australia, April 3-7, 2017*, pages 1291–1299. ACM, 2017. ISBN 978-1-4503-4913-0. doi : 10.1145/3038912.3052579. URL https://doi.org/10.1145/3038912.3052579. 5, 6, 13, 26, 71, 84

Bhaskar Mitra, Corby Rosset, David Hawking, Nick Craswell, Fernando Diaz, et Emine Yilmaz. Incorporating query term independence assumption for efficient retrieval and ranking using deep neural networks. *CoRR*, abs/1907.03693, 2019. URL http://arxiv.org/abs/1907.03693. 152

Calvin N Mooers. *The theory of digital handling of non-numerical information and its implications to machine economics*. Zator Co., 1950. 13

Fiona Fui-Hoon Nah. A study on tolerable waiting time : How long are web users willing to wait ? In *9th Americas Conference on Information Systems, AMCIS 2003, Tampa, FL, USA, August 4-6, 2003*, page 285. Association for Information Systems, 2003. URL http://aisel.aisnet.org/amcis2003/285. 6

Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, et Li Deng. MS MARCO : A human generated machine reading comprehension dataset. In *Proceedings of the Workshop on Cognitive Computation : Integrating neural and symbolic approaches 2016 co-located with the 30th Annual Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain, December 9, 2016*, 2016. URL http://ceur-ws.org/Vol-1773/CoCoNIPS_2016_paper9.pdf. 84

Liang Pang, Yanyan Lan, Jiafeng Guo, Jun Xu, et Xueqi Cheng. A study of

matchpyramid models on ad-hoc retrieval. *CoRR*, abs/1606.04648, 2016a. URL http://arxiv.org/abs/1606.04648. xiv, 108, 109

Liang Pang, Yanyan Lan, Jiafeng Guo, Jun Xu, Shengxian Wan, et Xueqi Cheng. Text matching as image recognition. In Schuurmans et Wellman (2016), pages 2793–2799. ISBN 978-1-57735-760-5. URL http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/11895. 68, 111

Liang Pang, Yanyan Lan, Jiafeng Guo, Jun Xu, Jingfang Xu, et Xueqi Cheng. Deeprank : A new deep architecture for relevance ranking in information retrieval. In Ee-Peng Lim, Marianne Winslett, Mark Sanderson, Ada Wai-Chee Fu, Jimeng Sun, J. Shane Culpepper, Eric Lo, Joyce C. Ho, Debora Donato, Rakesh Agrawal, Yu Zheng, Carlos Castillo, Aixin Sun, Vincent S. Tseng, et Chenliang Li, editors, *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, CIKM 2017, Singapore, November 06 - 10, 2017*, pages 257–266. ACM, 2017. ISBN 978-1-4503-4918-5. doi : 10.1145/3132847.3132914. URL https://doi.org/10.1145/3132847.3132914. 5, 26, 84

Bryan Perozzi, Rami Al-Rfou, et Steven Skiena. Deepwalk : online learning of social representations. In Sofus A. Macskassy, Claudia Perlich, Jure Leskovec, Wei Wang, et Rayid Ghani, editors, *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*, pages 701–710. ACM, 2014. ISBN 978-1-4503-2956-9. doi : 10.1145/2623330.2623732. URL https://doi.org/10.1145/2623330.2623732. 33

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, et Luke Zettlemoyer. Deep contextualized word representations. In Marilyn A. Walker, Heng Ji, et Amanda Stent, editors, *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics : Human Language Technologies, NAACL-HLT 2018, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 1 (Long Papers)*, pages 2227–2237. Association for Computational Linguistics, 2018. ISBN 978-1-948087-27-8. doi : 10.18653/v1/n18-1202. URL https://doi.org/10.18653/v1/n18-1202. 40

Benjamin Piwowarski, Max Chevalier, Éric Gaussier, Yoelle Maarek, Jian-Yun Nie, et Falk Scholer, editors. *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2019, Paris, France, July 21-25, 2019*, 2019. ACM. ISBN 978-1-4503-6172-9. doi : 10.1145/3331184. URL https://doi.org/10.1145/3331184. 176, 179, 190

Jay M. Ponte et W. Bruce Croft. A language modeling approach to information retrieval. In W. Bruce Croft, Alistair Moffat, C. J. van Rijsbergen, Ross Wilkinson, et Justin Zobel, editors, *SIGIR '98 : Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, August 24-28 1998, Melbourne, Australia*, pages 275–281. ACM, 1998. ISBN 1-58113-015-5. doi : 10.1145/290941.291008. URL https://doi.org/10.1145/290941.291008. 13, 19

Yifan Qiao, Chenyan Xiong, Zhenghao Liu, et Zhiyuan Liu. Understanding the behaviors of BERT in ranking. *CoRR*, abs/1904.07531, 2019. URL http://arxiv.org/abs/1904.07531. 77

Tao Qin, Tie-Yan Liu, Jun Xu, et Hang Li. LETOR : A benchmark collection for research on learning to rank for information retrieval. *Inf. Retr.*, 13(4) :346–374, 2010. doi : 10.1007/s10791-009-9123-y. URL https://doi.org/10.1007/s10791-009-9123-y. 66, 84

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, et Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8) :9, 2019. 48

C. Ramirez, V. Kreinovich, et M. Argaez. Why l1 is a good approximation to l0 : A geometric explanation. *Journal of Uncertain Systems*, 7(3) :203–207, 2013. 134

S. E. Robertson et H. Zaragoza. The probabilistic relevance framework : BM25 and beyond. *Foundations and Trends in Information Retrieval*, 3(4) :333–389, 2009. 18, 132, 134

Stephen E. Robertson et Steve Walker. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In Croft et van Rijsbergen (1994), pages 232–241. ISBN 978-3-540-19889-5. doi : 10.1007/978-1-4471-2099-5_24. URL https://doi.org/10.1007/978-1-4471-2099-5_24. 13, 18, 92

J. Rocchio. Relevance feedback in information retrieval. *The Smart Retrieval System-Experiments in Automatic Document Processing*, pages 313–323, 1971. URL https://ci.nii.ac.jp/naid/10000074359/en/. 21, 25

D. Roy, S. Bhatia, et M. Mitra. Selecting discriminative terms for relevance model. In *SIGIR 2019, Paris, France, July 21-25, 2019*, pages 1253–1256, 2019. 132

G. Salton. *A theory of indexing*, volume 18 of *Regional conference series in applied mathematics*. SIAM, 1975. 131

G. Salton et C. Yang. On the specification of term values in automatic indexing. *Journal of Documentation*, 29 :351–372, 1973. 18

G. Salton, A. Wong, et C.-S. Yang. A vector space model for automatic indexing. *Commun. ACM*, 18(11) :613–620, 1975. 17, 132

Gerard. Salton. *Automatic Information Organization and Retrieval.* McGraw Hill Text, 1968. ISBN 0070544859. 13

Shota Sasaki, Shuo Sun, Shigehiko Schamoni, Kevin Duh, et Kentaro Inui. Cross-lingual learning-to-rank with shared representations. In Marilyn A. Walker, Heng Ji, et Amanda Stent, editors, *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics : Human Language Technologies, NAACL-HLT, New Orleans, Louisiana, USA, June 1-6, 2018, Volume 2 (Short Papers)*, pages 458–463. Association for Computational Linguistics, 2018. ISBN 978-1-948087-29-2. doi : 10.18653/v1/n18-2073. URL https://doi.org/10.18653/v1/n18-2073. 88

Dale Schuurmans et Michael P. Wellman, editors. *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, 2016. AAAI Press. ISBN 978-1-57735-760-5. URL http://www.aaai.org/Library/AAAI/aaai16contents.php. 185, 189

Fabrizio Sebastiani. A probabilistic terminological logic for modelling information retrieval. In Croft et van Rijsbergen (1994), pages 122–130. ISBN 978-3-540-19889-5. doi : 10.1007/978-1-4471-2099-5_13. URL https://doi.org/10.1007/978-1-4471-2099-5_13. 26

Ralph R Shaw. The rapid selector. *Journal of documentation*, 5(3) :164–171, 1949. 13

Ying Shen, Yang Deng, Min Yang, Yaliang Li, Nan Du, Wei Fan, et Kai Lei. Knowledge-aware attentive neural network for ranking question answer pairs. In Collins-Thompson et al. (2018), pages 901–904. doi : 10.1145/3209978.3210081. URL https://doi.org/10.1145/3209978.3210081. 107, 115

Luca Soldaini et Nazli Goharian. Quickumls : a fast, unsupervised approach for medical concept extraction. In *MedIR workshop, sigir*, pages 1–4, 2016. 115

Luca Soldaini et Nazli Goharian. Learning to rank for consumer health search : A semantic approach. In Joemon M. Jose, Claudia Hauff, Ismail Sengör Altingövde, Dawei Song, Dyaa Albakour, Stuart N. K. Watt, et John Tait, editors, *Advances in Information Retrieval - 39th European Conference on IR Research, ECIR 2017, Aberdeen, UK, April 8-13, 2017, Proceedings*, volume 10193 of *Lecture Notes in Computer Science*, pages 640–646, 2017. ISBN 978-3-319-56607-8. doi : 10.1007/978-3-319-56608-5_60. URL https://doi.org/10.1007/978-3-319-56608-5_60. 26

Charles Spearman. The proof and measurement of association between two things. *American Journal of Psychology*, 15(1) :72–101, 1904. xvii, 40

Chi Sun, Luyao Huang, et Xipeng Qiu. Utilizing BERT for aspect-based sentiment analysis via constructing auxiliary sentence. In Burstein et al. (2019), pages 380–385. ISBN 978-1-950737-13-0. doi : 10.18653/v1/n19-1035. URL https://doi.org/10.18653/v1/n19-1035. 48

Ilya Sutskever, Oriol Vinyals, et Quoc V. Le. Sequence to sequence learning with neural networks. In Ghahramani et al. (2014), pages 3104–3112. URL http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks. 48

Paul Switzer. Vector images in document retrieval. *Statistical association methods for mechanized documentation*, pages 163–171, 1964. 13

Lynda Tamine, Laure Soulier, Gia-Hung Nguyen, et Nathalie Souf. Offline versus online representation learning of documents using external knowledge. *ACM Trans. Inf. Syst.*, 37(4) :42 :1–42 :34, 2019. doi : 10.1145/3349527. URL https://doi.org/10.1145/3349527. 26

Mortimer Taube, CD Gull, et Irma S Wachtel. Unit terms in coordinate indexing. *American Documentation (pre-1986)*, 3(4) :213, 1952. 13

Julián Urbano, Mónica Marrero, et Diego Martín. A comparison of the optimality of statistical significance tests for information retrieval evaluation. In Gareth J. F. Jones, Paraic Sheridan, Diane Kelly, Maarten de Rijke, et Tetsuya Sakai, editors, *The 36th International ACM SIGIR conference on research and development in Information Retrieval, SIGIR '13, Dublin, Ireland - July 28 - August 01, 2013*, pages 925–928. ACM, 2013. ISBN 978-1-4503-2034-4. doi : 10.1145/2484028.2484163. URL https://doi.org/10.1145/2484028.2484163. 96

C. J. van Rijsbergen. A non-classical logic for information retrieval. *Comput. J.*, 29(6) :481–485, 1986. doi : 10.1093/comjnl/29.6.481. URL https://doi.org/10.1093/comjnl/29.6.481. 26

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, et Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, et Roman Garnett, editors, *Advances in Neural Information Processing Systems 30 : Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 5998–6008, 2017. URL http://papers.nips.cc/paper/7181-attention-is-all-you-need. 48, 49, 51, 52, 53

Loïc Vial, Benjamin Lecouteux, et Didier Schwab. Sense Vocabulary Compression through Semantic Knowledge for Word Sense Disambiguation. In *TALN 2019 (Conférence sur le Traitement Automatique des Langues Naturelles)*, Toulouse, France, July 2019. URL https://hal.archives-ouvertes.fr/hal-02176195. 48

Stephen Walker, Richard M Jones, et Rachel De Vere. *Improving subject retrieval in online catalogues : Stemming, automatic spelling correction and cross-reference tables*, volume 2. Boston Spa, Wetherby, UK : British Library Research and Development Department, 1987. 15

Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, et Roman Garnett, editors. *Advances in Neural Information Processing Systems 32 : Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, 2019. URL http://papers.nips.cc/book/advances-in-neural-information-processing-systems-32-2019. 175, 190

Shengxian Wan, Yanyan Lan, Jiafeng Guo, Jun Xu, Liang Pang, et Xueqi Cheng. A deep architecture for semantic matching with multiple positional sentence representations. In Schuurmans et Wellman (2016), pages 2835–2841. ISBN 978-1-57735-760-5. URL http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/11897. 40

Changhan Wang, Kyunghyun Cho, et Jiatao Gu. Neural machine translation with byte-level subwords. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational*

Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020, pages 9154–9160. AAAI Press, 2020. ISBN 978-1-57735-823-7. URL https://aaai.org/ojs/index.php/AAAI/article/view/6451. 15

Yining Wang, Liwei Wang, Yuanzhi Li, Di He, et Tie-Yan Liu. A theoretical analysis of ndcg type ranking measures. In Shai Shalev-Shwartz et Ingo Steinwart, editors, *Journal of Machine Learning Research*, volume 30 of *Proceedings of Machine Learning Research*, pages 25–54, Princeton, NJ, USA, 12–14 Jun 2013. PMLR. URL http://proceedings.mlr.press/v30/Wang13.html. 24

Qiang Wu, Christopher J. C. Burges, Krysta M. Svore, et Jianfeng Gao. Adapting boosting for information retrieval measures. *Inf. Retr.*, 13(3) :254–270, 2010. doi : 10.1007/s10791-009-9112-1. URL https://doi.org/10.1007/s10791-009-9112-1. 13

Fen Xia, Tie-Yan Liu, Jue Wang, Wensheng Zhang, et Hang Li. Listwise approach to learning to rank : theory and algorithm. In William W. Cohen, Andrew McCallum, et Sam T. Roweis, editors, *Machine Learning, Proceedings of the Twenty-Fifth International Conference (ICML 2008), Helsinki, Finland, June 5-9, 2008*, volume 307 of *ACM International Conference Proceeding Series*, pages 1192–1199. ACM, 2008. ISBN 978-1-60558-205-4. doi : 10.1145/1390156.1390306. URL https://doi.org/10.1145/1390156.1390306. 64, 65

Chenyan Xiong, Zhuyun Dai, Jamie Callan, Zhiyuan Liu, et Russell Power. End-to-end neural ad-hoc ranking with kernel pooling. In Kando et al. (2017), pages 55–64. ISBN 978-1-4503-5022-8. doi : 10.1145/3077136.3080809. URL https://doi.org/10.1145/3077136.3080809. 72, 74

Wei Yang, Kuang Lu, Peilin Yang, et Jimmy Lin. Critically examining the "neural hype" : Weak baselines and the additivity of effectiveness gains from neural ranking models. In Piwowarski et al. (2019), pages 1129–1132. ISBN 978-1-4503-6172-9. doi : 10.1145/3331184.3331340. URL https://doi.org/10.1145/3331184.3331340. 4, 5, 100

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, et Quoc V. Le. Xlnet : Generalized autoregressive pretraining for language understanding. In Wallach et al. (2019), pages 5754–5764. URL http://papers.nips.cc/paper/

8812-xlnet-generalized-autoregressive-pretraining-for-language-understanding. 151

Rui Ye, Xin Li, Yujie Fang, Hongyu Zang, et Mingzhong Wang. A vectorized relational graph convolutional network for multi-relational network alignment. In Sarit Kraus, editor, *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pages 4135–4141. ijcai.org, 2019. doi : 10.24963/ijcai.2019/574. URL https://doi.org/10.24963/ijcai.2019/574. 151

Zeynep Akkalyoncu Yilmaz, Wei Yang, Haotian Zhang, et Jimmy Lin. Cross-domain modeling of sentence-level evidence for document retrieval. In Kentaro Inui, Jing Jiang, Vincent Ng, et Xiaojun Wan, editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pages 3488–3494. Association for Computational Linguistics, 2019. doi : 10.18653/v1/D19-1352. URL https://doi.org/10.18653/v1/D19-1352. 77

Wenpeng Yin, Katharina Kann, Mo Yu, et Hinrich Schütze. Comparative study of CNN and RNN for natural language processing. *CoRR*, abs/1702.01923, 2017. URL http://arxiv.org/abs/1702.01923. 44

Quanzeng You, Hailin Jin, Zhaowen Wang, Chen Fang, et Jiebo Luo. Image captioning with semantic attention. In He et al. (2016a), pages 4651–4659. ISBN 978-1-4673-8851-1. doi : 10.1109/CVPR.2016.503. URL https://doi.org/10.1109/CVPR.2016.503. 40

Hamed Zamani, Mostafa Dehghani, W. Bruce Croft, Erik G. Learned-Miller, et Jaap Kamps. From neural re-ranking to neural ranking : Learning a sparse representation for inverted indexing. In Alfredo Cuzzocrea, James Allan, Norman W. Paton, Divesh Srivastava, Rakesh Agrawal, Andrei Z. Broder, Mohammed J. Zaki, K. Selçuk Candan, Alexandros Labrinidis, Assaf Schuster, et Haixun Wang, editors, *Proceedings of the 27th ACM International Conference on Information and Knowledge Management, CIKM 2018, Torino, Italy, October 22-26, 2018*, pages 497–506. ACM, 2018. ISBN 978-1-4503-6014-2. doi : 10.1145/3269206.3271800. URL https://doi.org/10.1145/3269206.3271800. 92, 140

Haïfa Zargayouna, Catherine Roussey, et Jean-Pierre Chevallet. Recherche

d'information sémantique : état des lieux. *Trait. Autom. des Langues*, 56(3), 2015. URL http://atala.org/Recherche-d-information-semantique. 156

C. Zhai et J. D. Lafferty. A study of smoothing methods for language models applied to ad hoc information retrieval. In *SIGIR 2001 : New Orleans, Louisiana, USA, September 9-13, 2001*, pages 334–342, 2001. 21, 132

Yukun Zheng, Zhen Fan, Yiqun Liu, Cheng Luo, Min Zhang, et Shaoping Ma. Sogou-qcl : A new dataset with click relevance label. In Collins-Thompson et al. (2018), pages 1117–1120. doi : 10.1145/3209978.3210092. URL https://doi.org/10.1145/3209978.3210092. 5, 84

Yukun Zhu, Ryan Kiros, Richard S. Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, et Sanja Fidler. Aligning books and movies : Towards story-like visual explanations by watching movies and reading books. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pages 19–27. IEEE Computer Society, 2015. ISBN 978-1-4673-8391-2. doi : 10.1109/ICCV.2015.11. URL https://doi.org/10.1109/ICCV.2015.11. 117