REPUBLIQUE ALGERRIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE MOULOUD MAMMERI DE TIZI-OUZOU
FACULTE DE GENIE ELECTRIQUE ET D'INFORMATIQUE
DEPARTEMENT D'INFORMATIQUE

Mémoire de Fin d'Etudes de Master Professionnel

Domaine : Mathématique et Informatique
Filière : Informatique
Spécialité : Ingénierie des Systèmes d'Information

Présenté par :
Melle BOUCHAAL Lydia.
Melle LEULMI Fazia.

Thème

Extension d'un modèle de recherche d'information pour la prise en compte de la représentation de type wordmbedding

Mémoire soutenu publiquement le 30/09/2017 Devant le jury composé de :

Présidente : Yesli Yasmine
Encadreur : Mr A.Hammache
Examinatrice: Bougchiche L.

Promotion 2016/2017

Remerciement

Nous remercions DIEU de nous avoir donné santé, courage, force et patience pour réaliser ce modeste travail

Nous tenons à exprimer notre profonde gratitude à notre promoteur

Mr ***A. HAMMACHE*** *qui nous a beaucoup aidé afin de réaliser ce travail, avoir était toujours a nos cotés durant la réalisation de ce mémoire.*

Un grand merci à tous ceux qui, de prés ou de loin, ont contribué à la

réalisation de ce modeste travail

Enfin, nos remerciements s'adressent aux membres du jury qui vont nous faire l'honneur de juger notre travail

Dédicace

Je dédie ce travail

A mes chers parents qui ont toujours été là pour moi, qui m'ont donné un magnifique modèle de volonté et de persévérance, pour leur attention, sacrifice et soutien tout au long de mes études.

A mes frères Karim ;Nourdinne,Hamid et Belkacem

A mes sœurs Siham, Nadia et Karima

A mes belles-sœur Nabila ; Naçira et Fatiha

A mes adorables neveu Mouhamed, Kanza, Ikram, Abd nour,Walid et Rayan et Ilyana .

A ma chère enseignante et mon exemple mademoiselle yesli yasmine

A ma très cher binôme Lydia pour les bons moments qu'on a passé ensemble durant notre réalisation de mémoire

A mes camarades de la section mathématiques et informatique.

A tous mes amis (es) et à toute personne qui me connais de prés ou de loin

fazia

Dédicace

Je dédie ce travail

A mes chers parents qui ont toujours été là pour moi, qui m'ontdonné un magnifique modèle de volonté et de persévérance, pour leurattention, sacrifice et soutien tout au long de mes études.

A mon frère mouhand et ma seoursamira,

A mes grands-parents,

A mes très chères tantes Naima, Safia, Fazia, Hakima , Fadhila

A mes ancles Amar, Hakim, Samir et Mouhand

A mes très chers amisCylia, Sihem, Farida, Samia, Dyhia, Nawel, Hamama et la liste est longue.

A ma très cher binôme Fazia pour les bons moments qu'on a passé ensemble durant notre réalisation de mémoire.

A tous mes camarades de notre promotion 2016 2017

Lydia

Liste des figures

Liste des figures

Liste des tableaux

Sommaire

SOMMAIRE

Chapitre III : Approche proposée, implémentions et résultats

Introduction générale

Avec l'augmentation rapide du volume documentaire stocké sous format numérique ; et l'avènement du web, la quantité d'information disponible ne cesse de croitre au cours de ces dernières années, il est devenue alors très difficile de trouver une information ou un document qui répond a un besoin utilisateur.il a fallu donc envisager le développement des outils automatique qui permettent l'accès ciblé et efficace à cette masse donnée. Ces difficultés ont donné naissance à une nouvelle discipline appelée **Recherche d'Information**.

La Recherche d'Information (RI) peut être définie comme une activité dont la finalité est de localiser et de délivrer un ensemble de documents à un utilisateur en fonction de son besoin en informations. Le défi est de pouvoir, parmi le volume important de documents disponibles, trouver ceux qui correspondent au mieux à l'attente de l'utilisateur.

L'opération de la RI est réalisée par des outils informatiques appelés Systèmes de Recherche d'Information (SRI), ces systèmes ont pour but de mettre en correspondance une représentation du besoin de l'utilisateur (requête) avec une représentation du contenu des documents au moyen d'une fonction de comparaison (ou de correspondance). L'essor du web a remis la RI face à de nouveaux défis d'accès à l'information, il s'agit cette fois de retrouver une information pertinente dans un espace diversifié et de taille considérable.

Nous travail se focalise sur le domaine de la recherche d'information (RI), précisément dans l'extension du modèle de recherche d'information qui consiste à reclasser les documents obtenus lors de la recherche simple, ceci à fin d'obtenir des résultats plus pertinents.

Pour réaliser notre objectif nous intégrons la technique du word embedding (intégration des mots) qui permet de représenter les termes de la collection par des vecteurs dans un espace à dimension réduites.

Le word embedding à la capacité de viser la sémantique des mots et pour cela les mots qui sont sémantiquement similaires sont représentés les uns proches des autres dans l'espace vectoriel. Différents modèles existent permettent de construire les word embedding à savoir le modèle Glove, Word2vec qui se basent sur des réseaux de neurones.

Nous exploitons ainsi ces vecteurs des word embedding à fin de recalculer le score des documents restitués lors de la recherche simple.

Le problème qu'on pose dans notre mémoire et qui est notre but à atteindre est comment exploiter la technique du word embedding pour l'extension du modèle de recherche d'information.

Notre mémoire est décomposé en trois (03) principaux chapitres :

Le premier chapitre intitulé **« La Recherche d'Information »,** nous décrivons les points essentiels suivants : Tout d'abord nous donnons les concepts du base du recherche d'informations .on y trouve les notions de besoin en information de requête ; de document et de pertinence et le processus d'indexation. Nous décrivons aussi les différents modèles de recherche d'informations en particulier le model booléen ; le model vectoriel et le modèle probabiliste et le troisième point traité dans ce chapitre c'est évaluation des systèmes de recherche d'information.

Le deuxième chapitre **« le Word Embedding »,** est consacré à la présentation de la technique du word embedding qui est le concept que nous avons intégré dans notre travail à fin d'améliorer les résultats de la recherche d'information, ainsi nous avons présenté ses modèles qui se basent sur les réseaux de neurone.

Le troisième chapitre **« L'implémentation et résultats »,** est dédié à décrire l'approche de notre travail, une implémentation de cette approche et les différents outils utilisés pour la réaliser et nous terminons avec les résultats que nous avons tiré.

Enfin, nous concluons notre mémoire avec une conclusion générale et quelques perspectives.

Chapitre I
La Recherche d'Information

I.1 Introduction

Vu les grandes masses d'information qui se trouve aujourd'hui, l'utilisateur a de plus en plus des difficultés pour accéder à son besoin informationnel. La recherche d'information (RI) traite de la représentation, du stockage, de l'organisation et de l'accès à l'information. Le but d'un système de recherche d'information SRI est de retrouver, parmi une collection de documents préalablement stockés, les documents qui répondent au besoin utilisateur exprimé sous forme de requête. Dans ce chapitre nous présentons les concepts de la RI en particulier en décrivant les notions du document ; de requête et pertinence ainsi que l'architecture du système de recherche d'information (SRI), les modèles de RI et enfin l'évaluation des SRI

I.2 Définition de la Recherche d'Information

Plusieurs définitions ont été attribuées à la recherche d'information nous citons dans ce contexte les deux définitions suivantes

Definition1 : La recherche d'information (RI) est une branche de l'informatique qui s'intéresse à l'acquisition, l'organisation, le stockage, la recherche dans une masse documentaire existante, les documents contenant l'information qui répond au besoin informationnel exprimé par l'utilisateur [6].

Définition 2 : La recherche d'information est une discipline de recherche qui intègre des modèles et des techniques dont le but est de faciliter l'accès à l'information pertinente pour un utilisateur ayant un besoin en information [7].

I.3 Concepts de base de la recherche d'information

La RI fournit les techniques et outils pour permettre de représenter, stocker, organiser, rechercher et retrouver, dans une masse documentaire existante, les documents contenant l'information qui répond au besoin informationnel exprimé par l'utilisateur sous forme de requête. De ce contexte plusieurs concepts clés peuvent être définis, nous allons les clarifier dans ce qui suit :

I.3.1 Document et collection de documents

Un document constitue l'information élémentaire d'une collection de documents. L'information élémentaire, appelée aussi granule de document, peut représenter tout ou une partie d'un document [3], ce dernier peut être un texte, une page WEB, une image, une bande

vidéo, etc. Il s'agit de toute unité qui peut constituer une réponse à un besoin en information exprimé par un utilisateur [1].

L'ensemble des documents manipulés par un système de recherche d'information (SRI) se nomme collection de documents (ou base documentaire ou encore corpus)

I.3.2 Requête

Une requête constitue l'expression, dans un langage de requête, du besoin en information de l'utilisateur [1], une requête est un ensemble de mots clés, et elle peut être exprimée en langage naturel, booléen ou graphique.

I.3.3 La pertinence

La pertinence est un élément crucial et fondamental dans le domaine RI, elle peut être définie comme une mesure d'informativité du document à la requête ou encore la correspondance entre un document et une requête [2]. Il existe deux types de pertinence :

- **La pertinence système :** c'est l'évaluation par le système de recherche d'information, de l'adéquation entre les documents et une requête [1].
- **La pertinence utilisateur:** c'est l'évaluation par l'utilisateur, de la pertinence, vis-à-vis de son besoin en information, des documents retrouvés par le système de recherche d'information [1].

I.4 Système de Recherche d'Information (SRI)

Un Système de Recherche d'Informations (SRI) est un ensemble de programmes informatiques qui permet de retourner à partir d'une collection de documents, ceux dont le contenu correspond le mieux à un besoin en informations d'un utilisateur, exprimé à l'aide d'une requête [4].

I.5 Le processus de la recherche d'information

La figure suivante représente les différentes étapes de la RI dans un système de recherche d'informations (SRI) :

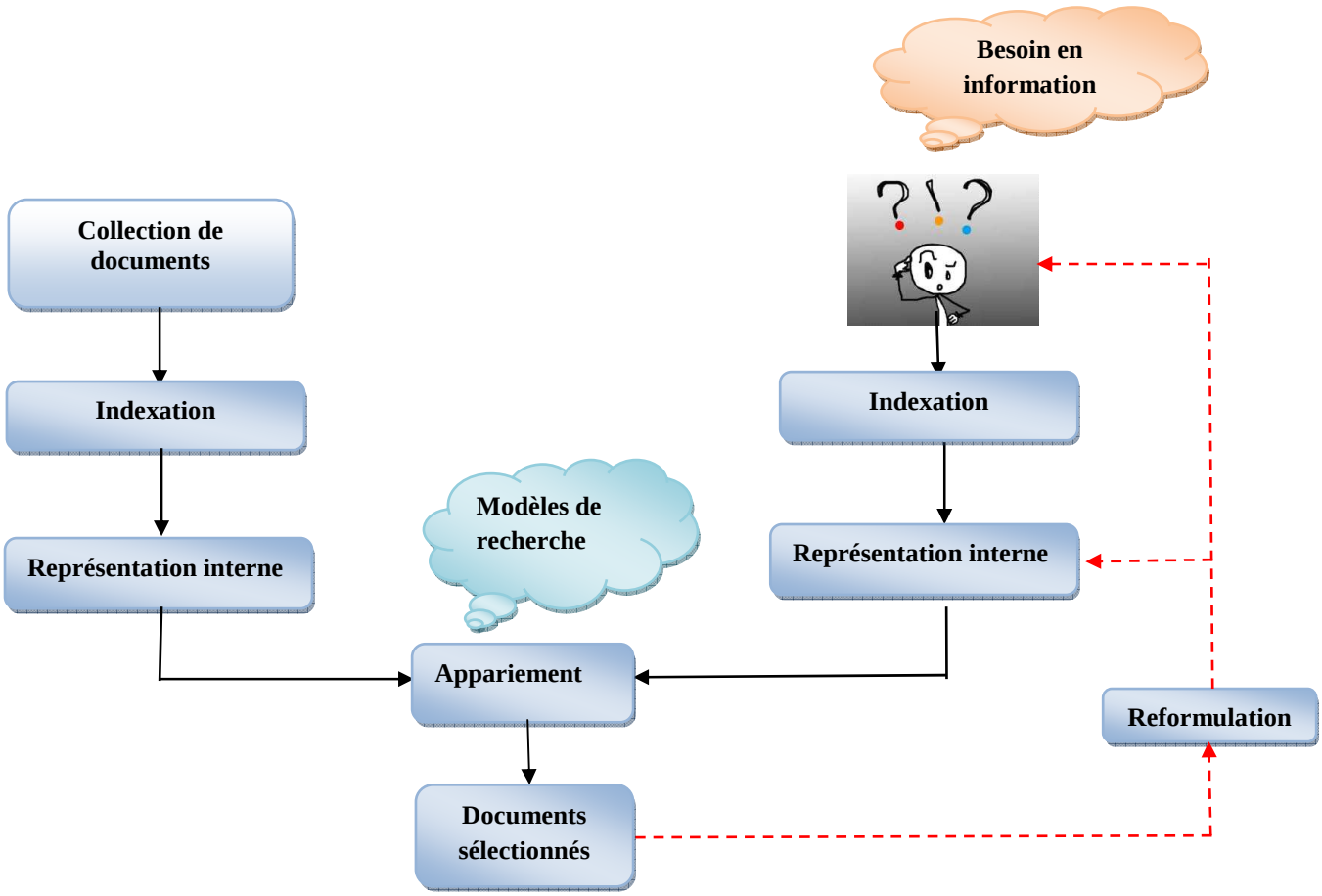


Figure I-1 : Processus de la recherche d'information [1]

A travers le processus on a identifié les processus suivant : l'indexation, l'appariement, et la reformulation de la requête que nous allons détailler ci-dessous :

I.5.1 L'Indexation

L'indexation est un processus qui transforme les documents en substituts appelées aussi descripteurs capables de représenter leurs contenus sémantiques [1], elle a pour objectif de créer une représentation interne (l'index) des documents en vue de faciliter la recherche.

On trouve trois(3) types d'indexation : manuelle, semi-automatique et automatique.

- **Indexation manuelle :**

  C'est un indexeur humain qui se charge de définir les descripteurs (mots clés) représentatifs du contenu du document, l'indexation manuelle assure une meilleure précision dans les documents restitué par le SRI en réponse aux requêtes des utilisateurs, néanmoins cette indexation représente un certain nombre d'inconvénients liés notamment à l'effort et le prix qu'elle exige (en temps et en nombre de

personnes). De plus cette indexation est subjective, qui est liés aux facteurs humains, différents spécialistes peuvent indexer un document avec des termes différents. Pratiquement inapplicable aux corpus de textes volumineux [1].

- **Indexation semi-automatique :**
  L'indexation est réalisée par un programme informatique et un indexeur humain.
  Le choix final des termes d'indexation à partir du vocabulaire fourni, est laissé à l'indexeur humain (généralement spécialiste du domaine) [1].
- **Indexation automatique :**
  C'est un processus complètement automatisé, elle est réalisée par un programme informatique, elle se charge d'extraire les termes caractéristiques du document [1], elle est particulièrement adaptée aux corpus volumineux, elle se passe par un ensemble d'étapes pour créer l'index d'une façon automatique :

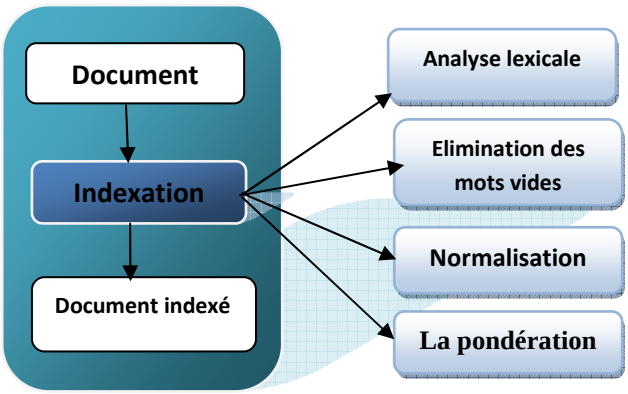


Figure I-2 : Le processus de l'indexation automatique

Les étapes de l'indexation automatique définies dans la **figure I-2** sont détaillées ci-dessous

I.5.1.1 L'analyse lexicale (tokenization en anglais)

Elle permet de convertir un texte de document en liste de token, ce dernier est un groupe de caractères constituant un mot significatif dans un document. L'analyse lexicale inclut les étapes suivantes :

• La conversion de la casse (majuscules en minuscules)
• L'élimination des accents

• La tokénisation : Elle consiste à découper le document en un ensemble d'unités lexicale (mot ou token). Chaque unité lexicale ou un radicale est une séquence de caractères entourées par des séparateurs d'unités. Elle permet alors de reconnaitre les espaces, les chiffres, les ponctuations, etc. [5].

I.5.1.2 L'élimination des mots vides

Cette étape consiste à identifier les mots vides, ou mots qui n'apportent pas de sens au texte (prépositions, déterminants, adverbes, conjonctions, …) et les éliminer, on distingue deux techniques pour l'élimination des mots vides :

- ✓ L'utilisation d'une liste des mots vides (aussi appelée anti-dictionnaire ou stoplist).
- ✓ L'utilisation des mesures statistiques : éliminer des mots dépassant un certain nombre d'occurrences dans la collection ou les mots rares de la collection [5].

I.5.1.3 La normalisation des termes

L'objectif est de ramener les mots de la même famille à leur forme normale, la normalisation s'applique dans les documents et dans la requête [1], plusieurs traitements de normalisation existent dont principalement :

a) **La lemmatisation (Stemming en anglais) :** a pour objectif à construire un lemme d'un mot. Un lemme est tout ce qui reste du mot après avoir éliminé les affixes (préfixes, suffixes). Elle permet de substituer chaque mot par sa racine (lemme), qui est soit sous la forme infinitive si le mot est un verbe, soit sous la forme de singulier masculin si le mot est un nom, adjectif ou article [5].

b) **La racinisation (ou troncature) :** son principe consiste à construire une racine commune à un ensemble de mots de la même famille en tronquant les mots en questions à partir d'un certain rang avec la suppression des flexions et des suffixes [5]. (**Ex :** cheval, chevaux, chevalier, chevalerie, chevaucher →"cheva"(mais pas "cavalier ")).

I.5.1.4 La pondération

La pondération est une fonction fondamentale en RI. Tous les modèles de recherche, excepté le modèle booléen, se basent sur la pondération des termes.
L'idée de la pondération est d'affecter à chaque terme t d'un document d ou d'une requête q, un poids numérique sensé le caractériser dans le document ou la requête, les poids des termes

de la requête et du document peuvent avoir des sémantiques différentes, le poids est donc une mesure statistique de l'importance du terme dans le document (plus un terme est important dans un document, plus son poids dans ce document doit être élevé) [1].

Parmi les mesures de pondération utilisées, nous avons la mesure $\boldsymbol{tf_{t,d}}$*$\boldsymbol{idf_t}$

Notant: $\boldsymbol{tf_{t,d}}$ (term frequency)**:** La fréquence d'occurrence du terme **t** dans le document **d**. cette mesure est proportionnelle à la fréquence du terme dans le document. Plus un terme est fréquent dans un document, plus il est important dans la description de ce document.

$\boldsymbol{idf_t}$*(Inverse of Document Frequency*)**:** La fréquence documentaire **inverse** du terme **t,** c'est une mesure l'importance d'un terme dans toute la collection. L'idée sous-jacente est que les termes qui apparaissent dans peu de documents de la collection sont plus représentatifs du contenu de ces documents que ceux qui apparaissent dans tous les documents de la collection.

Donc le poids $\boldsymbol{w_{t,d}}$ du terme **t** dans un document **d** est définit comme suit :

$$\boldsymbol{w_{t,d}}=\boldsymbol{tf_{t,d}}*\boldsymbol{idf_t} \quad \text{(I.1)}$$

I.5.2 L'appariement document-requête

La fonction l'appariement document-requête permet de mesurer la valeur de la pertinence d'un document vis-à-vis d'une requête. Afin de réaliser cela le SRI représente le document et la requête avec un même formalisme, puis le SRI compare les deux représentations, le résultat de cette comparaison se traduit par un score qui détermine la probabilité de pertinence (degré de la similarité ou degré de correspondance) de document vis-à-vis de la requête. Cette fonction d'appariement est noté RSV (d,q) (RSV : Retrieval Staus Value) où **d** représente le document de la collection et **q** la requête .cette valeur permet en suite au SRI d'ordonner les document renvoyés au utilisateurs [2].

I.5.3 La reformulation de la requête

La reformulation de requête consiste, à partir d'une requête initiale formulée par l'utilisateur, à construire une nouvelle requête qui répond mieux à son besoin informationnel. Pratiquement, la reformulation de la requête consiste à modifier la requête de l'utilisateur par ajout de termes significatifs et/ou réestimation de leurs poids [1]. Nous présentons dans ce qui suit les principales techniques de la reformulation :

I.5.3.1 La réinjection de pertinence(ou relevance feedback)

Lorsque l'information sur la pertinence des documents retournés en réponse à la requête initiale est utilisée pour l'améliorer [1], deux types de réinjection de pertinence existant :

a.1 Réinjection de pertinence explicite :

L'idée est de faire participer l'utilisateur dans le processus de recherche de sorte à améliorer l'ensemble final de résultats. Le procédé de base est le suivant :

- ✓ L'utilisateur formule sa requête,
- ✓ Le système lui renvoie un premier ensemble de résultats de recherche.
- ✓ L'utilisateur marque quelques documents retournés comme pertinents ou non pertinents (en pratique, seuls les 10 (ou 20) premiers documents classés sont examinés).
- ✓ L'idée principale consiste à sélectionner les termes importants des documents considérés comme pertinents, et améliorer l'importance de ces termes dans la nouvelle formulation de requête.
- ✓ Une nouvelle requête sera formulée par le système à partir de la requête initiale en utilisant l'information sur la pertinence (nouveaux termes sélectionnés)
- ✓ La nouvelle requête est obtenue à partir de la requête initiale en appliquant un algorithme spécifique par exemple l'algorithme de Rocchio qui a été proposé pour la réinjection de pertinence dans le modèle vectoriel.
  La formule est comme suit de Rocchio pour la réinjection de pertinence est donnée par [1] :

$$\boldsymbol{Q_m}=\alpha\boldsymbol{Q_0}+\beta\frac{1}{|\boldsymbol{R}|}\sum_{\boldsymbol{d_p}\in\boldsymbol{R}}\boldsymbol{d_p}\ -\boldsymbol{\gamma}\frac{1}{|\boldsymbol{NR}|}\sum_{\boldsymbol{dnp}\in\boldsymbol{NR}}\boldsymbol{d_{np}} \qquad \text{(I.2)}$$

- $\boldsymbol{Q_m}$ **:** le vecteur de la nouvelle requête reformulée.
- $\boldsymbol{Q_0}$ **:** le vecteur de la requête initiale.
- $\boldsymbol{d_p}$ (respectivement $\boldsymbol{d_{np}}$**):**le vecteur associé à un document pertinent retrouvé (respectivement non pertinent).

- **R** est l'ensemble des documents pertinents de la collection.
- **NR** est l'ensemble des documents non pertinents de la collection.
- **α, β, γ** des constantes telles que **α + β + γ =1**

a.2 Réinjection de pertinence implicite (pseudo-réinjection de pertinence) :

L'idée est d'utiliser les résultats de la recherche en vue d'améliorer les résultats du SRI sans l'intervention de l'utilisateur. On suppose uniquement que les **tops m** documents retournés sont considérés comme pertinents, et on les utilise pour reformuler la requête initiale.

La variante de la formule de Rocchio pour la réinjection de pertinence explicite est exprimée par la formule suivante :

$$\boldsymbol{Q_m = \alpha.Q_0 \ + \beta \frac{1}{|R|} \sum_{d_p \in R} d_p} \quad \text{(I.3)}$$

Où

- $\boldsymbol{Q_m}$ **:** le vecteur de la nouvelle requête reformulée.
- $\boldsymbol{Q_0}$ **:** le vecteur de la requête initiale.
- $\boldsymbol{d_p}$ **:** le vecteur associé à un document pertinent retrouvé.
- **R** est l'ensemble des documents pertinents de la collection.
- $\boldsymbol{\alpha}$ et $\boldsymbol{\beta}$ sont des constantes.

I.6 Les modèles de recherche d'information

Un modèle de RI a pour rôle de fournir une formalisation du processus de recherche, il permet de créer une représentation interne pour un document ou pour une requête basée sur des termes pondérés issus de l'indexation, ainsi il permet de définir une méthode de comparaison entre une représentation du document et celle de la requête afin de détecter leur degré de correspondance(ou de similarité). Les modèles de recherche sont classés en trois classes principales : les modèles booléens, les modèles vectoriels et les modèles probabilistes

I.6.1 Les modèles booléens

I.6.1.1 Le modèle booléen de base

Ce modèle est le premier modèle utilisé dans le domaine de la RI. Dans ce modèle, la requête est représentée sous forme d'une expression logique formée de termes d'indexation reliés par des opérateurs booléens AND, OR et NOT. Par contre le document est représenté sous forme de conjonction de termes d'indexation non pondérés. L'appariement requête-document est strict et se base sur des opérations ensemblistes selon les règles suivantes :

RSV (d,t_i) = 1 si t ∈ d, 0 sinon (I.6)

RSV (d, t_iANDt_j) = 1 si (t_i ∈d)et(t_j ∈ d), 0 sinon (I.7)

RSV(d,t_i OR t_j) = 1 si (t_i ∈ d) ou(t_j ∈ d), 0 sinon (I.8)

RSV (d, NOT t_i) = 1 si $t_i \notin$ d, 0 sinon (I.9)

I.6.1.1 Le modèle booléen étendu

Il a été introduit par [Salton et al. 1983]. c'est une extension du modèle précédent qui vise à tenir compte d'une pondération des termes dans le corpus, dans le but d'ordonner les documents retournés par le SRI. l'appariement requête-document est le plus souvent déterminé par les relations introduites dans le modèle p-norm basées sur les p-distances, avec 1 ≤p≤ ∞. La valeur de **p** est indiquée au moment de la requête.

Un document "d" est ainsi représenté par l'ensemble des termes auxquels sont associés des poids : d= {...., (ti pi),....}. La requête est une expression booléenne classique. La fonction d'appariement peut être définie par:

- RSV (d, t_i) = p_i (I.10)
- RSV(d , q_i et q_j) = min(RSV(d,q_i),RSV(d,q_j)) (I.11)
- RSV (d , q_i ou q_j)= max(RSV(d,q_i),RSV(d,q_j)) (I.12)
- RSV(d , non qi)= 1-RSV(d,q_i) (I.13)

I.6.2 Les modèles vectoriels

I.6.2.1 Le modèle vectoriel de base

C'est le modèle le plus populaire en RI. Un document $\boldsymbol{d_i}$ est représenté par un vecteur de poids $\boldsymbol{w_{ij}}$ de dimension **n**, dans l'espace vectoriel composé de tous les termes d'indexation $\boldsymbol{d_i = (w_{i1}, w_{i2}, \dots, w_{in})}$

Une requête **Q** est aussi représentée par un vecteur de poids $\boldsymbol{w_{Q,i}}$ défini dans le même espace vectoriel que le document. $\boldsymbol{Q = (w_{Q1}, w_{Q2}, \dots, w_{Qn})}$

Où $\boldsymbol{w_{Qj}}$ est le poids de terme $\boldsymbol{t_j}$ dans la requête **Q**, et $\boldsymbol{w_{ij}}$ son poids dans le document $\boldsymbol{d_i}$. Ce poids peut être soit une forme de $\boldsymbol{tf_{t,d}}$ * $\boldsymbol{idf_t}$, soit un poids attribué manuellement par l'utilisateur.

La pertinence du document $\boldsymbol{d_i}$ pour la requête **Q** est mesurée comme le degré de corrélation des vecteurs correspondants. Cette corrélation peut être exprimée par l'une des mesures suivantes:

–Le produit scalaire : RSV ($\boldsymbol{d_i}$,Q)=$\sum_{j=1}^{n}(w_{Qj} * w_{ij})$ (I.14)

–La mesure du cosinus :RSV(d,Q)=$\frac{\sum_{j=1}^{n} w_{Qj} * w_{ij}}{(\sum_{j=1}^{n} {w_{Qj}}^2)^{\frac{1}{2}} * (\sum_{j=1}^{n} {w_{ij}}^2)^{\frac{1}{2}}}$ (I.15)

–La mesure de Dice : RSV (d,Q) = $\frac{2 * \sum_{i=1}^{n} w_{ij} * w_{Qj}}{\sum_{j=1}^{n} {w_{Qj}}^2 + \sum_{j=1}^{n} {w_{ij}}^2}$ (I.16)

–La mesure de Jacard : RSV (d,Q) = $\frac{\sum_{i=1}^{n} w_{ij} * w_{Qj}}{\sum_{1}^{n} {w_{Qj}}^2 + \sum_{1}^{n} {w_{ij}}^2] - (\sum_{i=1}^{n} w_{ij} * w_{Qj})}$ (I.17)

–Coefficient de superposition : Sim($d_{i,Q}$)=$\frac{\sum_{i=1}^{n} w_{ij} * w_{Qj}}{\min(\sum_{j=1}^{n} {w_{Qj}}^2, \sum_{j=1}^{n} {w_{ij}}^2)}$ (I.18)

I.6.2.2 Le modèle d'indexation sémantique latente (LSI)

Ce modèle peut être vu comme une amélioration du modèle vectoriel .L'objectif du modèle LSI est de construire des index conceptuels portant sur la sémantique des mots dans les documents. Ces index sont tirés à partir de la structure sémantique latente des textes des documents.

Pour ce faire, partant de l'espace vectoriel de tous les termes d'index, LSI construit un espace d'indexation de taille réduite k, par application de la décomposition en valeur singulière SVD

(Singular Value Decomposition) de la matrice termes-documents [Deerwesteret al., 90], cette décomposition en valeurs singulières permet de « regrouper des mots sémantiquement proches», chaque groupe de mots forme un concept. Les **k** dimensions (ou concepts) obtenus capturent une partie importante de la structure sémantique des documents [Berry et al., 94].

Chaque vecteur document est au final représenté dans l'espace *k*-dimensionnel réduit des termes non bruités. Les documents qui partagent des termes co-occurrents ont des représentations proches. La requête utilisateur est aussi représentée par un vecteur dans l'espace *k*-dimensionnel. Une mesure de similarité est ensuite calculée entre le *k*-vecteur requête et chacun des *k*-vecteurs documents de la collection. A l'issue de la recherche, le système sélectionne les documents pertinents même s'ils ne contiennent aucun mot de la requête [6].

I.6.3 Les modèles probabilistes

Le principe du modèle probabiliste consiste à présenter les résultats d'un SRI dans un ordre basé sur la probabilité de pertinence d'un document vis-à-vis d'une requête.

I.6.3.1 Les modèles probabilistes de base

Soit **Q** est une requête utilisateur et **d** un document, le modèle probabiliste tente d'estimer la probabilité que le document **d** appartienne à la classe des documents pertinents (ou non pertinents). Un document est sélectionnés si la probabilité qu'il soit pertinent pour **Q** notée **P(R|d)** est supérieure à la probabilité qu'il soit non pertinent pour **Q**, notée **P(NR|d).**

RSV (d,Q) est donné par :

$$\textbf{RSV (d,Q)} = \frac{P(R|d)}{P(NR|/d)} \qquad \text{(I.19)}$$

En utilisant la formule de Bayes et en simplifiant on obtient:

$$\textbf{RSV (d,Q)} = \frac{P(d|R)}{P(d|NR)} \qquad \text{(I.20)}$$

P(d|R) (respectivement **P(d|NR**)) est la probabilité que le document appartienne à l'ensemble l'ensemble **R** des documents pertinents (respectivement à l'ensemble **NR** des documents non pertinents).

Il existe d'autres méthodes pour estimer ces probabilités comme dans le modèle BIR(Binary Independance Retrieval). Les probabilités **P (d|R)** et **P(d|NR**) sont données par:

P(d |R)=P($t_1 = x_1, t_2 = x_2, t_3 = x_3$….|R)=$\prod_i p(t_i = x_i|R)$=$\prod_i P(t_i = 1|R)^{x_i}$* $P(t_i = 0|R)^{1-x_i}$ (I.21)

P(d |NR)=P($t_1 = x_1, t_2 = x_2, t_3 = x_3$ ….|NR)= $\prod_i p(t_i = x_i|NR)$ = $\prod_i P(t_i = 1|NR)^{x_i}$ * $P(t_i = 0|NR)^{1-x_i}$ (I.22)

t_i est le *ième* terme utilisé pour décrire le document **d**, et **xi** est sa valeur **0** si le terme est absent, **1** si le terme est présent dans le document.

I.6.3.2 Le modèle de langue

L'utilisation des modèles de langue en RI remonte à 1998 []. Le principe de ce modèle consiste à construire un modèle de langue pour chaque document, soit$\boldsymbol{M_d}$, puis de calculer la probabilité qu'une requete **q** puisse être générée par le modèle de langue du document, soit$\boldsymbol{P(q|M_d,)}$. Cette probabilité est alors exprimée comme suit :

$$\textbf{RSV (d,q) = P}(q = (t_1, t_2, \dots, t_n)|M_d) = \prod_i P(t_i|M_d) \quad \text{(I.23)}$$

P($t|d$) est donnée par :

$$\textbf{P}(t_i|M_d) = \frac{tf(t_i,d)}{\sum_t tf(t,d)} \quad \text{(I.24)}$$

Où **tf(t|d)** est la fréquence d'occurrence du n-gramme (du mot) t_i dans le document **d** et $\sum_t tf(t,d)$ correspond à la taille du document (c'est-à-dire le nombre totale d'occurrence de n-grammes).

Pour remédier au problème posé par des mots de la requête absents dans le document, qui ont pour effet d'avoir la probabilité **P(q|M_d)** nulle ; des techniques de lissage (smoothing) sont utilisés. Le lissage consiste à assigner des probabilités non nulles aux termes, qui n'apparaissent pas dans les documents. Différentes techniques de lissage existent dont le lissage de Laplace, le lissage de Goog-Turing, le lissage Bakoff, le lissage par interpolation, etc [9]

I-7 L'évaluation des SRI

L'évaluation constitue une étape importante dans la mise en œuvre d'un SRI, elle permet de mesurer les caractéristiques du système en termes de qualité de service et de facilité d'utilisation, et d'évaluer la performance du système.

Le but d'un SRI est de retrouver l'information recherchée par l'utilisateur (ou information pertinente) et de la lui retourner dans un délai acceptable, en la lui présentant sous une forme

aisément exploitable. Ceci implique la capacité du système à retrouver les documents pertinents d'une part, et la manière de présenter les résultats d'autre part.

I-7-1 Les mesures d'évaluation de SRI

Pour évaluer la performance d'un SRI, certaines mesures statistiques sont utilisées : **Rappel et précision**, la **MAP, Bruit** et **Silence**,

- **Rappel et précision :** Le rappel et la précision sont deux mesures de base pour évaluer les performances des systèmes de recherche d'information. Le principe de ces deux mesures est basé sur la connaissance à-priori des documents pertinents de la collection d'une part, et d'autre part la partition de l'ensemble des documents restitués par le SRI en deux catégories : documents pertinents et documents non pertinents. La figure I-4, illustre la partition de la collection de tests pour une requête :
-

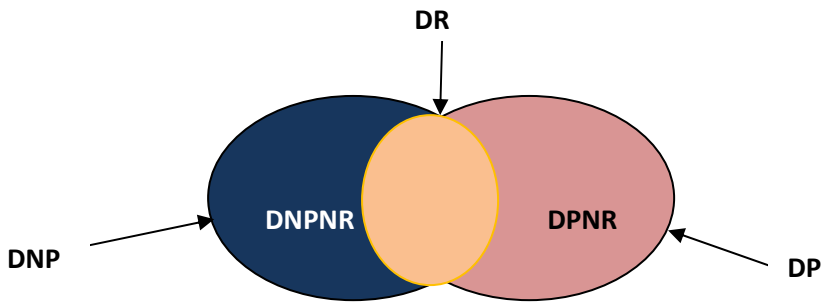


Figure I-4 : Répartition des documents d'une collection suite à une requête

DP : Documents pertinents

DNP : Documents non pertinents

DR : Documents retrouvés

DPNR : Documents pertinents non retrouvés

DNPNR : Documents non pertinents non retrouvés

Le rappel: C'est la capacité du SRI à retrouver les documents pertinents de la collection. Le rappel indique le pourcentage de documents pertinents qui ont été retrouvés par le SRI par rapport à l'ensemble des documents pertinents de la collection. Le rappel est donné par la proportion suivante :

$$\textbf{Rappel} = \frac{\textit{Nombre des documents pertinents retrouvés}}{\textit{Nombre des documents pertinents}} = \frac{|DPR|}{|DP|} \qquad \text{(II-25)}$$

Un rappel égal à **1** signifie que tous les documents pertinents ont été retrouvés.

La précision : Elle calcule la capacité du SRI à retrouver uniquement les documents pertinents. La précision permet de mesurer la fraction des documents pertinents parmi ceux qui ont été retrouvés par le système.

$$\textbf{Précision} = \frac{\textbf{Nombre des documents pertients retrouvés}}{\textbf{Nombre des documents retrouvés}} = \frac{|DPR|}{|DPR \cup DNPR|} \qquad \text{(II-26)}$$

Une précision égale à **1** signifie que le système n'a retrouvé que des documents pertinents.

Exemple :

Un SRI retourne en réponse à la requête sur le mot « pomme », une liste de 60 documents. Sur un total de 100 documents traitant du fruit « pomme », il en fournit 50, mais les 10 documents restants portent plutôt sur la compagnie « pomme et fils » qui vend des tournevis. Le rappel et la précision et de ce système est calculée alors comme suit :

$$\text{Précision} = \frac{50}{60} = 83\%$$

$$\text{Rappel} = \frac{50}{100} = 50\%$$

Bruit et silence: ce sont d'autres mesures complémentaires au Rappel et précision.

$$\textbf{Bruit: B} = \frac{\textbf{Nombre des documents non pertinents retrouvés}}{\textbf{Nombre des documents retrouvés}} = \frac{|DNPR|}{|DPR \cup DNPR|} \qquad \text{(II-27)}$$

$$\textbf{Silence: S} = \frac{\textbf{Nombre des documents non pertinents retrouvés}}{\textbf{Nombre des documents pertients}} = \frac{|DPNR|}{|DP|} \qquad \text{(II-28)}$$

Courbe Rappel/Précision : Les différentes valeurs de précision aux différents points de rappel servent à tracer la courbe Rappel/précision qui a en général l'aspect suivant comme le représente la figure ci-dessous.

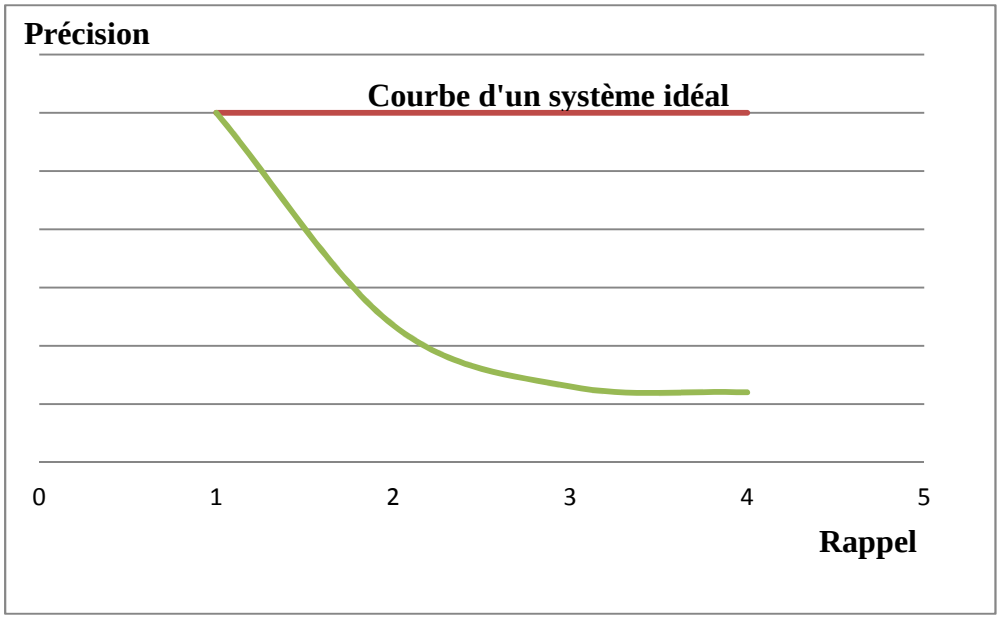


Figure I-5: Courbe rappel-précision

Exemple : le tableau suivant représente les valeurs du rappel et de la précision pour les 10 premiers documents renvoyés par le SRI pour une requête ayant 7 documents pertinents :

Rang du document renvoyé	Pertinence	Rappel	Précision
Doc1	(*)	0.14	1
Doc2		0.14	0.5
Doc3	(*)	0.28	0.66
Doc4	(*)	0.42	0.75
Doc5		0.42	0.6
Doc6	(*)	0.57	0.66
Doc7		0.57	0.57
Doc8		0.57	0.5
Doc9	(*)	0.71	0.55
Doc10	(*)	0.85	0.6

Tableau I-1 : valeurs du Rappel et précision

On représente généralement la courbe que par les valeurs de précision calculées à chaque document pertinent restitué, la courbe rappel-précision est tracée sur la figure suivante:

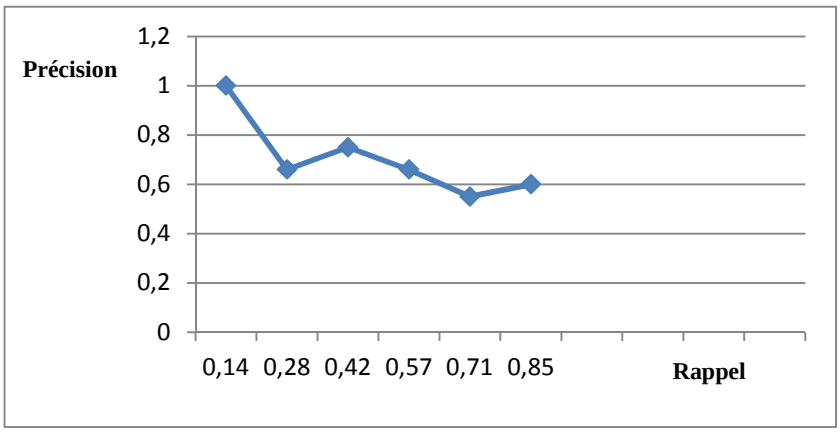


Figure I-6 : Exemple courbe Rappel-précision

- **Précision moyenne (AVG-P), la MAP et la R-précision :**
  - **a) Précision moyenne (AVG-P)** : c'est une valeur moyenne des valeurs de précision à chaque observation d'un document pertinent

Dans l'exemple précédent, les précisions aux documents pertinents observés sont respectivement : 1, 0.66, 0.75, 0.66, 0.55, 0.6 alors:

$$\text{AVG-P}= \frac{1+0.66+0.75+0.66+0.55+0.6}{6} \approx 0.7$$

b) MAP(Mean Average Precision): La MAP est la moyenne des précisions moyennes (**AVG-P**) obtenues sur l'ensemble des requêtes à chaque fois qu'un document pertinent est retrouvé**:**

$$\textbf{MAP}= \frac{\sum_{i=1}^{nq} AvgP(q_i)}{n_q} \qquad \text{(III-29)}$$

n_q **:** est le nombre des requêtes

c) La R-précision: la R-précision pour une requête **q** est la précision calculée au rang **R** (**Ranking**).

$$\textbf{R-prec=P@R=}\frac{|DPR|}{R} \qquad \text{(III-30)}$$

Où **R** est le nombre de documents pertinents pour la requête **q.**

On considère l'exemple précédent: on a 7 documents pertinents, la **R-précision** pour cette requête au rang 7 est :

$$\text{R-Précision} = \frac{4}{7} \approx 0{,}57$$

I-7-2 Les collections de test

Les collections de test sont utilisées afin d'évaluer la performance du SRI, une collection de test est composée d'un corpus de documents, une liste de requêtes prédéfinies et des jugements de pertinences qui sont établis par des experts pour définir les documents pertinents. Plusieurs collections de test ont été construites ; parmi elles

ADI (82 requêtes, 25 documents), **CACM**(3204 requêtes ,64 documents), **CISI(**1460 requêtes,112 documents**)**, **MED(**1033 requêtes, 30 documents**)**, **TIME**(425 requêtes,23)

Les compagnes d'évaluation: les compagnes d'évaluation en RI travaillent pour évaluer le degré de pertinence des réponses du SRI à une requête en utilisant des collections de test.

Chaque compagnie est constituée d'un certain nombre de tâches fournissant des résultats, et un protocole d'évaluation pour chaque tâche. Les compagnies de TREC (Text Retrieval Conference) sont devenues la référence en ce qui concerne l'évaluation des SRI mais on peut également citer les compagnies CLEF (Cross-Language Evaluation Forum), NTCIR (NII Tesbeds and Community for Information access Research), Amarllis et INEX [3].

La compagnie TREC: TREC est un projet international, initié au début des années 90 par le NIST (National Institute of Standards and Technology) aux Etats-Unis, dans le but de proposer des moyens homogènes d'évaluation de systèmes documentaires sur des bases de documents conséquentes. Il est aujourd'hui co-sponsorisé par le NIST et l'ARPA (exDARPA/ITO, pour Defense Advanced Research Projects Agency - Information Technology Office, qui mène plusieurs actions dans le domaine des technologies de l'informatique et de la communication, et qui dépend du ministère de la défense). Le projet TREC consiste en une série d'évaluations annuelles des technologies pour la RI, dont l'objectif est d'une part, d'offrir aux chercheurs le moyen de mesurer sur des

procédures d'évaluation uniformes, l'efficacité de leurs systèmes, d'autre part, de leur permettre de comparer les résultats de leurs systèmes [6].

Les tâches de TREC: Une tâche (ou piste) TREC vise à explorer de nouveaux domaines de recherche : la piste crée l'infrastructure nécessaire (des collections de test, méthodologie d'évaluation, etc.) pour soutenir la recherche sur sa tâche Chaque tâche a ses propres collections de test, et son propre protocole d'évaluation [1]. Voici quelques tâches TREC:

- **Ad-Hoc track:** c'est la tâche principale dans TREC, elle vise à évaluer les performances d'un SRI sur des ensembles statiques de documents, seules les requêtes changent.
- **Question AnsweringTrack:** elle travaille sur des parties de documents, elle vise à rechercher la pertinence d'un document dans sa totalité, on va rechercher la pertinence dans des extraits de documents
- **Web Track:** Une nouvelle tâche ad-hoc dans laquelle les documents sont un ensemble représentatif de documents issus du Web

I-8 Conclusion

Nous avons présenté dans ce premier chapitre les concepts fondamentaux de la recherche d'information. Le rôle d'un système de recherche d'information et de retrouver les informations pertinentes à partir d'une grande masse de données en passant par plusieurs étapes telle que l'indexation, la recherche et la reformulation pour arriver à répondre au besoin de l'utilisateur. Dans le chapitre suivant nous allons présenter un des modèles les plus récents utilisé dans la recherche d'information qui est le "Word embedding".

Chapitre II
Le Word Embedding

II.1 Introduction

Nous présentons dans ce chapitre le word embedding qui est une nouvelle méthode d'apprentissage qui représente les mots des requêtes et les documents dans un espace vectoriel, dans ce propos nous allons présenter comment ces word embeddings peuvent être utilisés dans un modèle de recherche d'information et ce qui apportent comme améliorations dans l'efficacité de la recherche d'information. Dans ce but, nous utilisons des word embeddings neuronaux dont on va étudier ses différents modèles et ses utilisations dans la recherche d'information ad hoc.

II.2 Définition du Word embedding

Le word embedding (en français : incorporation des mots = plongement des mots) appelés aussi modèles sémantiques distribués ou représentation distribuée est un moyen pour représenter les mots des documents et des requêtes dans un espace vectoriel dont chaque mot sera représenté par un vecteur de réels et les mots apparaissant dans des contextes similaires auront des vecteurs plus proches que d'autres apparaissant dans des contextes différents. Le word embedding utilise des modèles de représentation des mots comme Glove et word2vec qui se base sur les réseaux de neurones [1, 2,3]. La figure II-1 représente un exemple des word embeddings obtenus à partir du modèle Glove

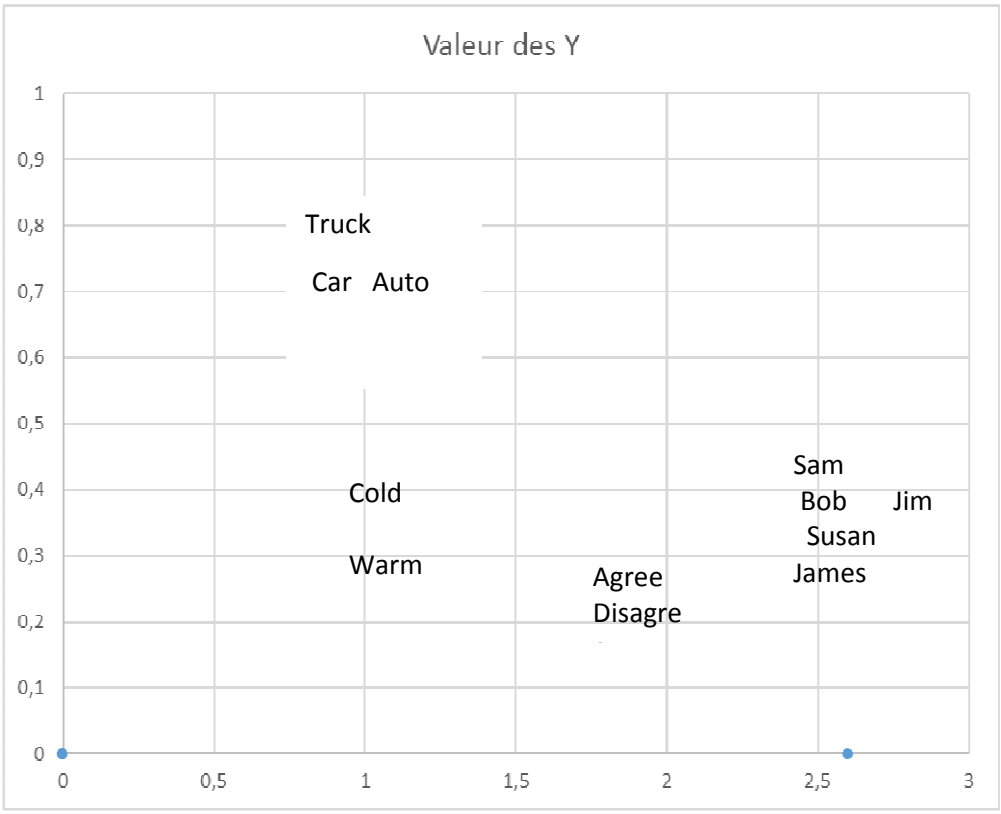


Figure II-1 : Word embeddings obtenus avec le modèle Glove [2]

A partir de la figure précédente on peut noter que par exemple les mots **Car** et **Auto** sont liée par une relation de **synonymie,** ils sont représentés par deux vecteurs qui sont proches ; et les mots **Agree** et **Disagree** sont liés par la relation d'**antonymie** c'est pour cela aussi ils sont plus proches**…** [2].

II.3 Historique

L'apprentissage non supervisé du word embedding a exceptionnellement connu de succès dans nombreuses taches de PNL (Processing Natural Language = processus de langage naturelle) qui consiste à remplacer les anciennes méthodes utilisées dans représentation distribuée.

Depuis les années 1990, les modèles vectoriels ont été utilisés pour représenter des mots. Pendant ce temps, de nombreux modèles ont été développés, y compris l'analyse sémantique latente (LSA) (ou LSI : Indexation Sémantique Latente) [18] qui est la première approche utilisée pour calculer les relations sémantiques entre les termes, et plus tard la méthode LDA (Latent Dirichlet Allocation = Allocation latente de Drichlet) [19].

Bengio et al. [10] ont associé le terme word embeddings (intégration des mots) en 2003 et ils l'ont utilisé dans un modèle de langue neuronal pour la première fois.

En 2008 Collobert et Weston [13] ont montré l'utilité des word embeddings, ils ont conçu une architecture unifiée pour le traitement du langage naturel qui établit non seulement des word embeddings comme un outil utile pour les tâches en aval, mais introduit également une architecture de réseau neuronal qui constitue la base de nombreuses approches actuelles.

Cependant, la notion de word embedding est plus connue grâce à Mikolov et al. en 2013 [15] qui ont créé word2vec, un outil qui permet de construire et utiliser des word embeddings avec ses deux architectures CBOW et Skip-gram. En 2014, Pennington et al. [17] ont lancé GloVe, qui est un autre modèle de word embedding.

II.4 Les réseaux de neurone

L'architecture des modèles du word embedding se base sur des réseaux de neurones, ces derniers travaillent sur la notion d'apprentissage. Nous présentons dans ce qui suit la définition d'un réseau de neurone, la notion d'apprentissage et quelques types de réseaux de neurones les plus utilisés dans les modèles des word embeddings.

II.4.1 Définition des réseaux de neurones artificiels

Les réseaux neuronaux artificiels (réseaux neuromimétiques) sont des réseaux fortement connectés de processeurs élémentaires (les neurones formels= neurones artificiels) fonctionnant en parallèle. Chaque processeur élémentaire calcule une sortie unique sur la base des informations qu'il reçoit. Toute structure hiérarchique de réseaux est évidemment un réseau [21]. Ils sont inspirés par des réseaux de neurones biologiques.

II.4.1.1 Analogie entre neurone biologique / neurone formel (artificiel)

Pour simuler le fonctionnement d'un neurone biologique, on introduit le modèle du neurone formel**.** La **figure II-2** montre la structure d'un neurone artificiel. Chaque neurone est un processeur élémentaire, il reçoit un nombre variable d'entrées en provenance de neurones amonts. A chacune de ces entrées est associé un poids **w** représentatif de la force de la connexion. Chaque processeur élémentaire est doté d'une sortie unique, qui se ramifie ensuite pour alimenter un nombre variable de neurones avals.

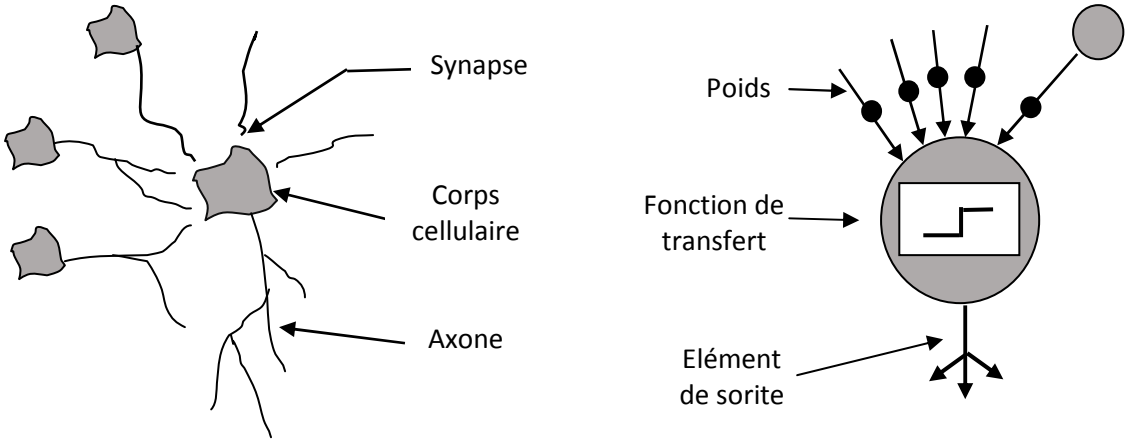


Figure II-2 : Mise en correspondance neurone biologique / neurone formel

- **Fonctionnement d'un neurone formel**

  Un neurone formel (artificiel) est une unité de traitement qui reçoit des données en entrée, sous la forme d'un vecteur, et produit une sortie réelle. Cette sortie est une fonction des entrées et des poids des connexions. La **figure II-3** représente un exemple d'un neurone formel avec trois (3) entrées :

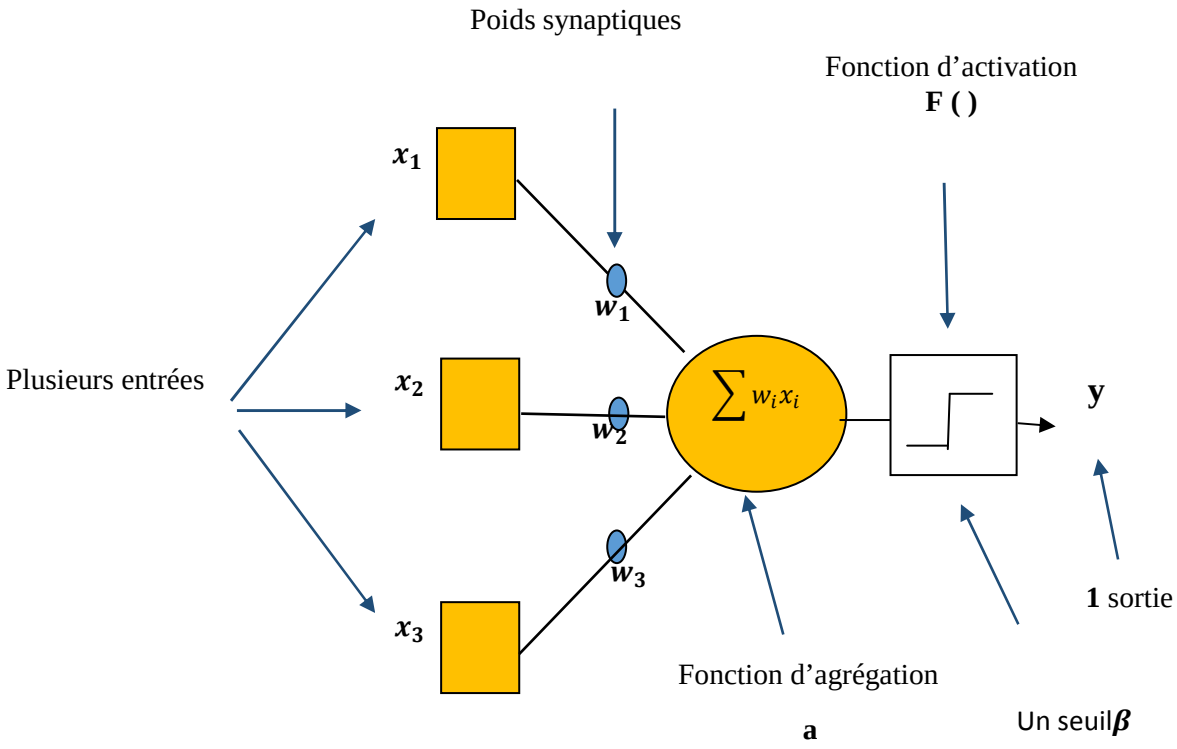


Figure II-3 : Le neurone formel , exemple avec 3 entrées

Les composant d'un neurone formel peuvent donc être définies de cette façon :

$x_1, x_2, \ldots x_n$ **:** constituent les valeurs d'entrées (valeurs scalaires).

$w_1, w_2, w_3 \ldots w_n$: constituent la pondération de chaque flèches ou synapses.

f : constitue une fonction d'activation : Elle s'applique à la sommation des entrées pour être comparée au seuil d'activité β et d'une sortie qui est l'équivalent de l'axone

a : constitue une valeur scalaire d'activation (la **Σ** des entrées pondérées = la fonction d'agrégation).

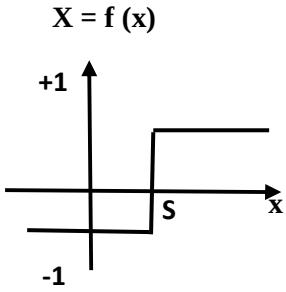
β **:** constitue le seuil (ou biais).

y : est la sortie

On distingue deux phases. La première est habituellement le calcul de la somme pondérée des entrées (a) selon l'expression suivante :

$$a = \sum (w_i \, . \, x_i)$$

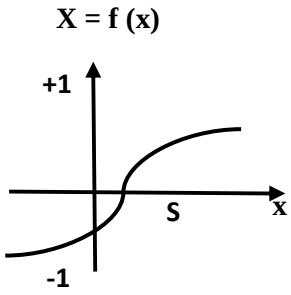
A partir de cette valeur de la somme, la fonction de transfert (ou fonction d'activation) **f( )** calcule la valeur de l'état du neurone. C'est cette valeur qui sera transmise aux neurones avals. Il existe de nombreuses formes possibles pour la fonction de transfert. Les plus courantes sont présentées sur la **figure II-4**. On remarquera que la plupart des fonctions de transfert sont continues, offrant une infinité de valeurs possibles comprisent dans l'intervalle [0, +1] (ou [-1, +1]).



a: fonction à seuil

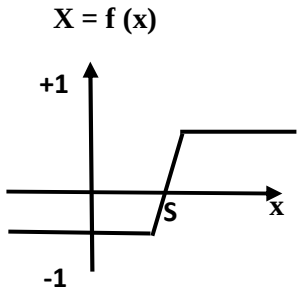
(**S** , la valeur du seuil)

$$f(x)=\begin{cases}\mathbf{0}\ if\ \mathbf{0} > x \\ \mathbf{1}\ if\ x\ \geq \mathbf{0}\end{cases}$$



b: fonction sigmoïde

$$f(x) = \frac{1}{1+e^{-\beta x}}$$



C : fonction linéaire par morceaux.

$$f(x) = \begin{cases}\mathbf{0}\ if\ x \leq\ x_{min} \\ mx + b\ if\ x_{max} > x >\ x_{min} \\ \mathbf{1}\ if\ x \geq x_{max}\end{cases}$$

Figure II-4 : Différents types de fonctions de transfert pour le neurone artificiel

II.4.1.2 Notion d'apprentissage dans les réseaux de neurone

L'apprentissage est une phase du développement d'un réseau de neurones durant laquelle le comportement du réseau est modifié jusqu'à l'obtention du comportement désiré [21], il consiste à modifier le poids des connexions entre les neurones appliquant des règles d'apprentissage, les plus connues sont : Règle de Hebb, correction d'erreur, apprentissage de Boltzmann, apprentissage par compétition.

Deux grandes classes principales de l'apprentissage artificiels sont définies : Apprentissage **supervisée** et **non supervisée**.

Dans le cas de **l'apprentissage supervisé (supervised learning)**, les exemples sont des couples (Entrée, Sortie associée (désirée)), c'est-à-dire qu'on doit disposer d'un comportement précis pour pouvoir l'inculquer au réseau neuronal, alors que l'on ne dispose que des valeurs (Entrée) pour **l'apprentissage non supervisé (unsupervised learning)** c'est-à-dire qu'on veut construire un réseau dont on ne connaît pas a priori la sortie correspondant à des entrées données**.** Cependant les modèles à apprentissage non supervisés s'effectuent sous le contrôle d'un expert [21,22].

II.4.2 Les types des réseaux de neurones

Les connexions entre les neurones qui composent le réseau décrivent la topologie du modèle. Elle peut être quelconque, mais le plus souvent il est possible de distinguer une certaine régularité.

II.4.2.1 Perceptron simple

Le perceptron est un type de réseau de neurones considéré comme le type de réseaux neuronal le plus simple, il se compose de deux couches de neurones : La première couche est composée de cellules d'entrée, la deuxième couche fournit la réponse comme il dispose d'un unique neurone de sortie.

Il est utilisé pour la classification des données en deux catégories, il est limité aux seuls problèmes linéairement séparables. L'impossibilité de traiter les problèmes non linéaires avec les réseaux de type Perceptron a été résolue par un réseau multicouche.

II.4.2.2 Réseau de neurone multicouche (ou singulier)

C'est un réseau entièrement connecté, il était le premier et le plus simple type de réseau de neurones artificiels conçu, les neurones sont arrangés par couche. Il n'y a pas de connexion entre les neurones d'une même couche et les connexions ne se font qu'avec les neurones des couches avales **(figure II-5).** Chaque neurone d'une couche est connecté à tous les neurones de la couche suivante et celle-ci seulement. Ceci nous permet d'introduire la notion de sens de parcours de l'information (de l'activation) au sein d'un réseau et donc définir les concepts de neurone d'entrée, neurone de sortie [21].

Les fonctions d'entrée sont extraites ou apprises par des réseaux de neurone utilisant plusieurs couches empilées complètement connectées. Chaque couche applique une transformation linéaire à la sortie vectorielle de la dernière couche (réalisation d'une transformation affine).

Ainsi, chaque couche est associée à une matrice de paramètres, à estimer pendant l'apprentissage. Ceci est suivi d'une application élémentaire d'une fonction d'activation non linéaire. Dans le cas de la recherche d'information (RI), la sortie de l'ensemble du réseau est souvent une représentation vectorielle de l'entrée ou des scores prévus [3].

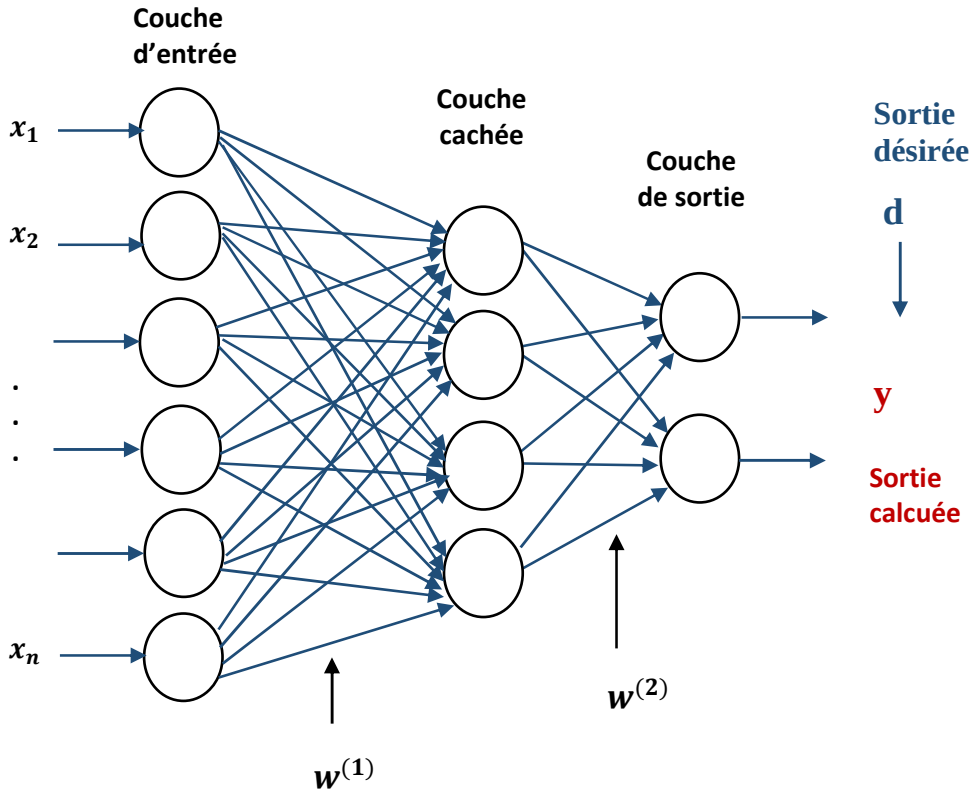


Figure II-5 : Réseau de neurone multicouche

L'apprentissage de ces réseaux s'appuie sur la rétropropagation du gradient de l'erreur (propagation depuis la couche de sortie vers la couche cachée).

L'algorithme de rétro-propagation du gradient

L'algorithme de rétropropagation consiste à mesurer l'erreur entre les sorties désirées et les sorties observées résultant de la propagation vers l'avant des entrées, et à rétropropager cette erreur à travers les couches du réseau en allant des sorties vers les entrées.

L'approche la plus utilisée pour la minimisation de la fonction E (Erreur) est basée sur la méthode du gradient.

On présente le premier vecteur d'entrée, une fois on a la sortie du réseau, l'erreur correspondante et le gradient de l'erreur par rapport à tous les poids sont calculés. Les poids sont alors ajustés.

On refait la même procédure pour tous les exemples d'apprentissage. Ce processus est répété jusqu'à ce que les sorties du réseau soient suffisamment proches des sorties désirées.

II.4.2.3 Le réseau Auto-encodeur

Un réseau neuronal Auto-encodeur est un modèle non supervisé utilisé pour apprendre une représentation des données, généralement il est utilisé dans le but de réduire la dimensionnalité. Un auto-codeur est formé pour reconstruire l'entrée, et la sortie a la même dimension que l'entrée [6,7].

II.4.2.4 Réseau à connexions récurrentes

Les connexions récurrentes ramènent l'information en arrière par rapport au sens de propagation défini dans un réseau multicouche.

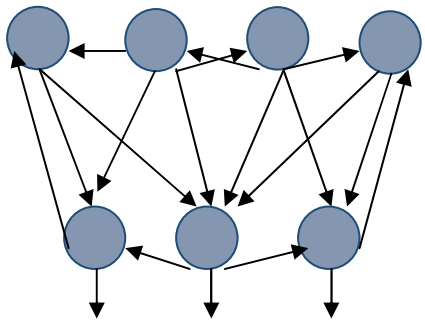


Figure II-6 : Réseau à connexion récurrente

II.4.2.5 Les réseaux de Boltzmann

Les réseaux de Boltzmann sont des réseaux symétriques récurrents. Ils possèdent deux sous-groupes de cellules, le premier étant relié à l'environnement (cellules dites visibles) et le second ne l'étant pas (cellules dites cachées). Cette règle d'apprentissage est de type stochastique (= qui relève partiellement du hasard) et elle consiste à ajuster les poids des connexions, de telle sorte que l'état des cellules visibles satisfasse une distribution probabiliste souhaitée [23].

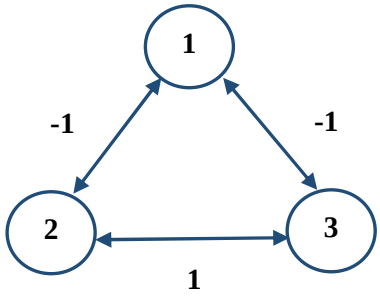


Figure II-7 : Exemple du réseau récurrent symétrique

II.5 Les modèles du word embedding

Ces modèles sont conçus pour produire des word embeddings, chaque modèle a ses propres méthodes et ils sont très étroitement liés avec les modèles de langue. En fait, de nombreux modèles de word embeddings à la pointe de la technologie tentent de prédire le prochain mot dans une séquence dans une certaine mesure. Parmi ces modèles nous distinguons : le modèle C & W, Wor2vec et Glove qu'on va présenter dans ce qui suit mais premièrement nous allons présenter le fonctionnement des modèles de langue.

II.5.1 La modélisation linguistique

Les modèles de langue neuronal consiste généralement à calculer la probabilité d'un mot w_t étant donné ses **n-1** mots précédents, c'est-à-dire **p (wt | wt-1, ⋯wt-n + 1).**

Nous pouvons calculer le produit d'une phrase ou d'un document en fonction des probabilités de chaque mot étant donné ses **n** mots précédents

$$P(w_1, \ldots, w_T) = \prod_i P(w_i | w_{i-1}, \ldots, w_{i-1+n}) \quad \text{(II-1)}$$

Dans les modèles de langue basés sur le **n grammes**, on peut calculer la probabilité d'un mot en fonction des fréquences de ses **n-grammes** constitutifs :

$$P(w_t | w_{t-1}, \ldots, w_{t-1+n}) = \frac{count(w_{t-n+1}, \ldots w_{t-1}, w_t)}{count(w_{t-n+1}, \ldots w_{t-1})} \quad \text{(II-2)}$$

Si **n=1**, on parle du modèle uni-gramme. Dans ce modèle les mots de la séquence sont générés indépendamment les uns des autres.

Dans le cas où n=2, on parle du modèle bi-gramme, chaque mot est dépendant seulement de son prédécesseur

Tandis que n = 5 avec le lissage de kniser-Ney conduit à des modèles lissés de 5 grammes qui se sont révélés être une base de référence forte pour la modélisation linguistique.

Dans les réseaux de neurones, nous atteignons le même objectif en utilisant la fonction de softmax suivante :

$$P(w_t | w_{t-1},\dots,w_{t-1+n}) = \frac{exp(h^T \acute{v}_{w_t})}{\sum_{w_{i\in v}} exp(h^T \acute{v}_{w_t})} \quad \text{(II-3)}$$

h est le vecteur de sortie de l'avant-dernière couche du réseau (la couche cachée dans le réseau multicouches de la figure II-5), tandis que **v'w** est l'embedding de sortie du mot **w**, c'est-à-dire sa représentation dans la matrice pondérale de la couche softmax. Notez que même si **v'w** représente le mot **w**, il est appris séparément du word embedding d'entrée **vw**, car les multiplications dans lesquelles les deux vecteurs sont impliqués diffèrent (**vw** est multiplié par un vecteur d'index, **v'w** avec **h**).

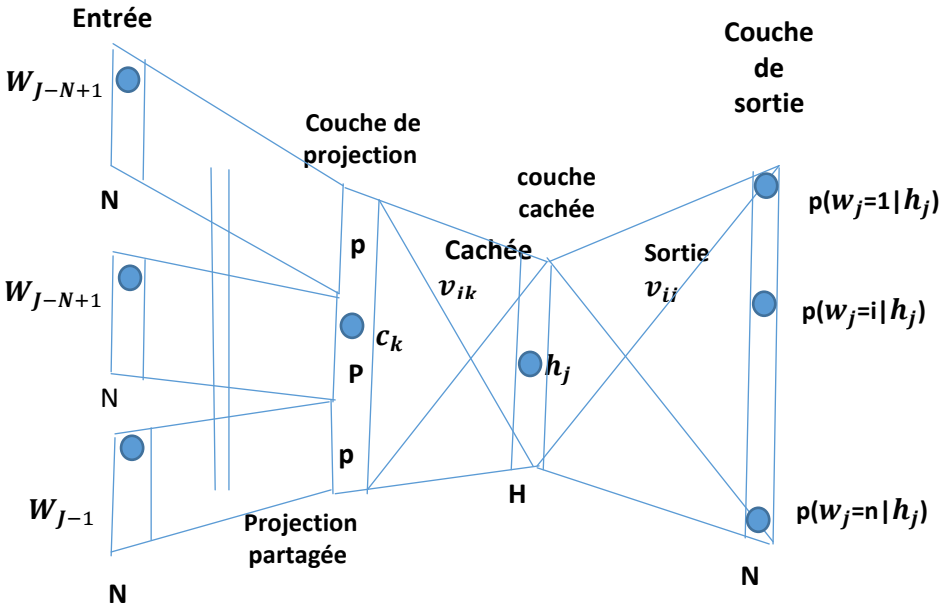


Figure II-8 : Le modèle de langue neuronal (Bengio et al., 2006)) [4]

On calcule la probabilité de chaque mot **w** à la couche de sortie du réseau neuronal. Pour ce faire efficacement, nous effectuons une multiplication matricielle entre **h** et une matrice de

poids constituées par **v'w** de tous les mots **w** dans le vocabulaire **V**. Nous obtenons ensuite le vecteur résultant, c'est-à-dire la sortie d'une couche précédente qui n'est pas une probabilité. En utilisant ensuit la fonction softmax, ce modèle tente de maximiser la probabilité de prédire le mot correct à chaque étape du temps **t**. L'ensemble du modèle tente donc de maximiser la probabilité logarithmique moyenne de l'ensemble du corpus :

$$j_\theta = \frac{1}{T} logp(w_1, \dots, w_T) \quad \text{(II-4)}$$

Par analogie, il est généralement formé pour maximiser la moyenne des probabilités logarithmiques de tous les mots dans le corpus donné leurs **n** mots précédents :

$$j_\theta = \frac{1}{T} \sum_{t=1}^{T} logp(w_t | w_{t-1}, \dots, w_{t-n+1}) \quad \text{(II-5)}$$

II.5.2 Modèle classique de langue neuronal

Le modèle classique de langue neuronal est proposé par Bengio et al. [10] en 2003 est un réseau neuronal multicouche avec une seule couche cachée qui sert à prédire le mot suivant dans une séquence.

Ce modèle maximise ce que nous avons décrit ci-dessus comme l'objectif du modèle de langue neuronal prototypique:

$$J_\theta = \frac{1}{T} \sum_{t=1}^{T} log\, f(w_t, w_{t-1}, \dots, w_{t-n+1}) \quad \text{(II-6)}$$

$f(w_t, w_{t-1}, \dots, w_{t-n+1})$ est la sortie du modèle, c'est-à-dire la probabilité $\mathbf{p(}w_t | w_{t-1}, \dots, w_{t-n+1}\mathbf{)}$ calculé par le softmax, où **n** est le nombre de mots précédents introduit dans le modèle.

L'architecture du modèle de langue classique est composée de :

1. **Couche embedding (couche d'integration):** une couche qui génère des word embeddings en multipliant un vecteur d'index des mots d'entrée par une matrice du word embedding .
2. **Couche (s) intermédiaire (s**): une ou plusieurs couches qui produisent une représentation intermédiaire de l'entrée, une couche entièrement connectée qui applique la non linéarité à la concaténation des word embedding de **n** mots précédents.
3. **Couche softmax:** la couche finale qui produit une distribution de probabilité sur les mots dans V [4].

I.5.3 Le modèle C & W (Collobert et Weston)

Collobert et Weston [5] (ainsi C & W) montrent en 2008 que les word embeddings sont utilisés sur un ensemble de données suffisamment important ont une signification syntaxique et sémantique et améliorent les performances sur les tâches en aval.

Leur solution pour éviter de calculer le softmax coûteux est d'utiliser une fonction objective différente, Collobert et Weston forment un réseau à produire un score supérieur f_θ. À cet effet, ils utilisent un critère de classement par paire, qui est calculé comme suit :

$$J_\theta = \sum_{x \epsilon X} \sum_{w \epsilon V} \max\{0, 1 - f_\theta(x) + f_\theta(x^{(w)})\} \quad \text{(II-7)}$$

Ce modèle consiste à échantillonner les fenêtres **x** contenant **n** mots de l'ensemble de toutes les fenêtres **X** possibles dans le corpus. Pour chaque fenêtre **x**, on produit alors une version incorrecte **x (w)** corrompue en remplaçant le mot central de **x** par un autre mot **w** de **V**. Leur objectif maximise maintenant la distance entre les scores produits par le modèle pour la fenêtre correcte et la fenêtre incorrecte avec une marge de 1. Leur architecture de modèle, représentée sur la figure II-9 :

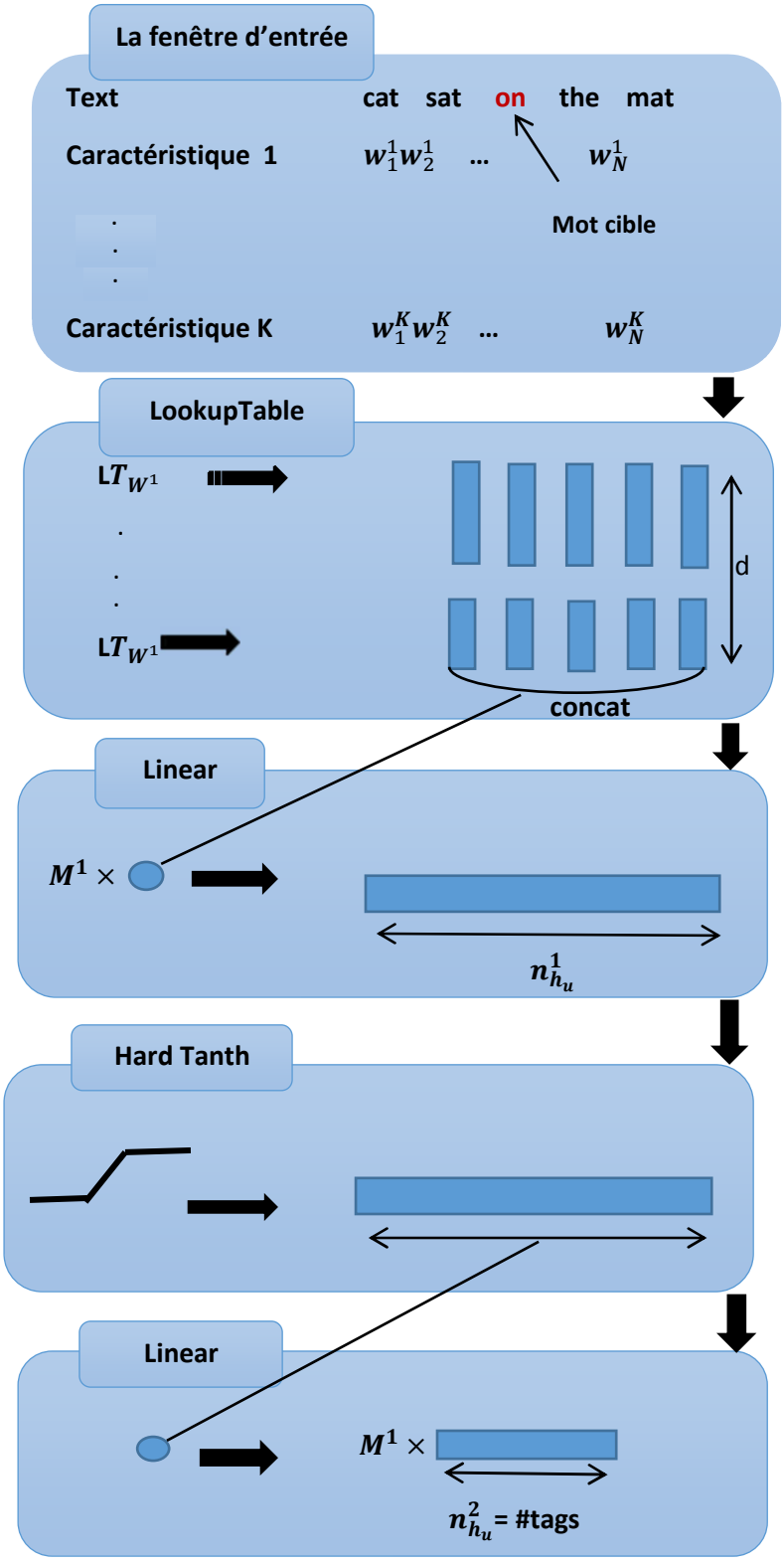


Figure II-9 : Architecture du modèle C & W (Collobert et al., 2011)[20]

Ce modèle produit ainsi les embeddings comme résultat qui possèdent déjà en relation avec plusieurs word embeddings qui sont connues.

Par exemple, les pays sont regroupés ensemble et des mots syntaxiquement similaires occupent des emplacements similaires dans l'espace vectoriel. Alors que leur objectif de classement élimine la complexité du softmax, ils conservent la couche cachée intermédiaire de Bengio et al. [19] (la couche HardTanh de la figure II-9), ce qui constitue une autre source de calcul couteuse.

II.5.4 Le modèle Word2vec

C'est le modèle le plus populaire, word2vec est un modèle qui produit des word embeddings, il utilise un réseau de neurone à trois couches, il a été créé par Mikolov et al. en 2013 [24].
Les word embeddings sont utilisé dans des modèles d'apprentissage profond (deep learning) pour les langages naturels PNL (processing natural language).

Le modèle word2vec se compose de deux (02) architetures : Le modèle Skip-gram et le modèle CBOW que nous décrivons dans ce qui suit.

II.5.4.1 Le modèle Skip-gram

Etant donné un corpus **W=** $\{w_1, \dots, w_t, \dots w_n\}$, le modèle Skip-gram sert à prédire les représentations des contextes **c** de ces mots de **W**.

Le modèle skip-gram construit des représentations vectorielles des mots de contextes.
L'objectif du modèle skip-gram est de maximiser la probabilité logarithmique moyenne suivante :

$$\frac{1}{|w|}\sum_{i=1}^{|w|}\sum_{|w|\leq j\leq |w|, j\neq 0}\log(w_{t+j}|w_t) \qquad \text{[II-8]}$$

La taille de la fenêtre de contexte **|w|** détermine quels mots entourant le mot cible w_t sont considérés pour le calcul de la probabilité logarithmique (où la fenêtre est centrée autour du mot cible). Cependant, notez que dans le modèle skipgram, **|w|** est dans le rayon de la fenêtre maximale : une taille de fenêtre $|\grave{w}| \leq |w|$est échantillonnée uniformément à partir de **[1; |w|]** [14].
La probabilité d'un mot de sortie est calculée selon la fonction softmax:

$$P(w_O|w_I)=\frac{\exp({v_{w_O}}^T v_{w_I})}{\sum_{w=1}^{|V|}\exp({v_w}^T, v_{w_I})} \qquad \text{[II-9]}$$

Où v_{w_I} et v_{w_O} sont les représentations vectorielles des vecteurs d'entrée et de sortie, respectivement, et $\sum_{w=1}^{|V|} \exp(v_w{}^T, v_{w_I})$ est le facteur de normalisation, dont le rôle est de normaliser les résultats internes du produit dans tous les mots de vocabulaire (**|V|** est la taille du vocabulaire). L'échantillonnage négatif est utilisé pour réduire la complexité du calcul [15].

L'architecture du modèle skip-gram est un réseau de neurone construit de trois (03) couches : la couche d'entrée, la couche de projection et la couche de sortie, la figure suivante représente l'architecture du modèle :

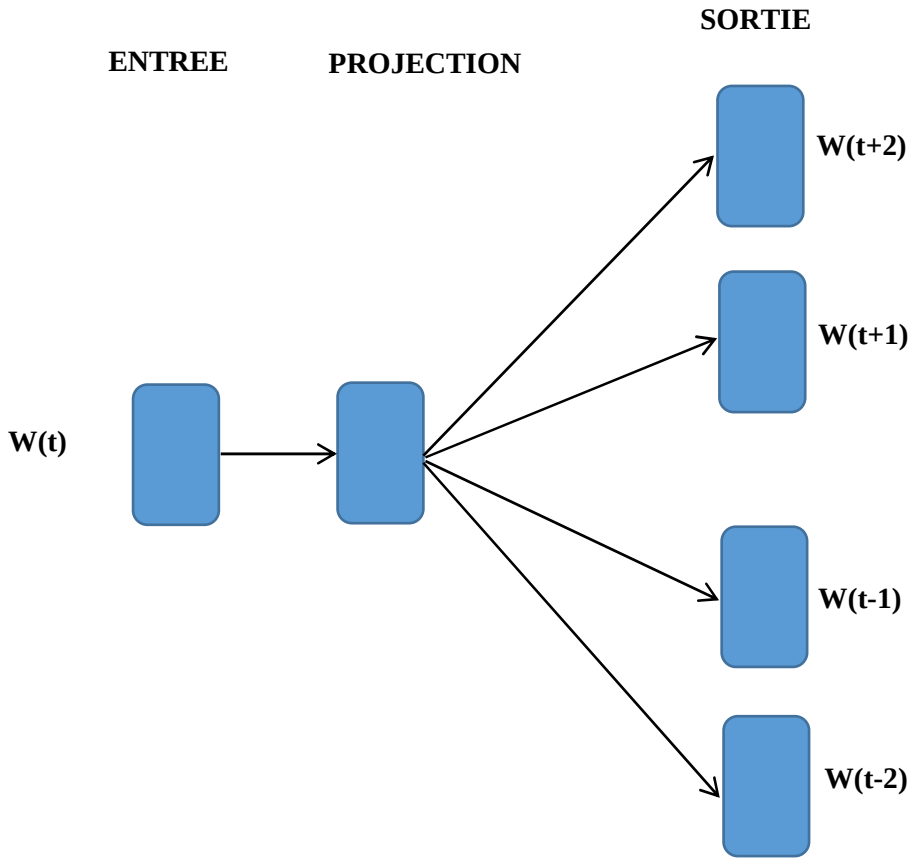


Figure II-10 : Architecture du modèle skip-gram [15]

Exemple :

On considère la phrase suivante : « probability language model for IR »

On sélectionne le mot **model** comme une entrée qui est représentée par **w(t)** dans la figure II-10, et on cherche ses mots voisins : **w(t+1), w(t+2), w(t-1), w(t-1)** en appliquent la fonction II-9.

II.5.4.2 Le modèle CBOW (the Continuous Bag-Of-Words)

CBOW (sac de mots continu) est une architecture du modèle word2vec permet de construire des word embeddings, il travail sur la notion de probabilité conditionnelle c'est à dire prédire un mot étant donné son contexte. Il est basé sur le modèle de langue neuronal [9].

Etant donné un mot cible w_t et une séquence de mots du vocabulaire **W** = $\{w_{t-2}; w_{t-1}; w_{t+1}; w_{t+2}\}$ (où w_{t-k} précède et w_{t+k} suit w_t par **k** positions), l'objectif du modèle CBOW est de maximiser la probabilité de prédire correctement le mot cible w_t. Fortement, le modèle CBOW génère un vecteur v_t, correspondant au mot cible w_t, comme moyenne des vecteurs des mots dans la séquence **W**, c'est-à-dire $v_t = \frac{1}{|w|}\sum_{i=1}^{|w|} v_i$ [8].

La fonction objective de CBOW à son tour n'est que légèrement différente de celle du modèle de langue :

$$J_\theta = \frac{1}{T}\sum_{t=1}^{T} \log p(w_t | w_{t-n}, \dots, w_{t-1}, w_{t+1}, \dots, w_{t+n}) \qquad \text{[II-10]}$$

Au lieu de prendre les **n** mots précédents comme dans le modèle de langue, ce modèle reçoit une fenêtre de **n** mots autour du mot cible w_t .

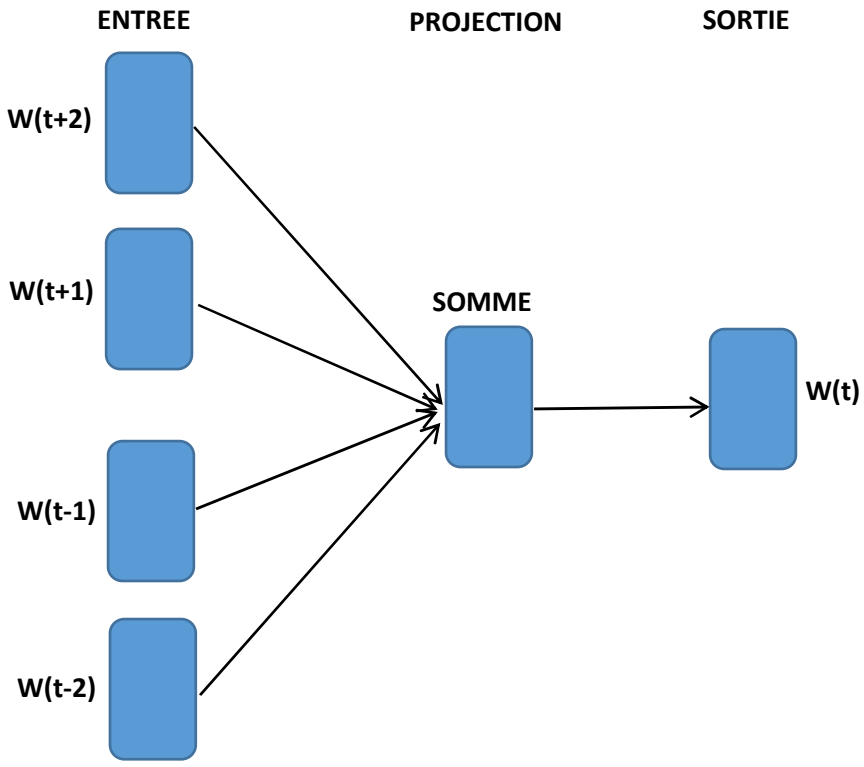


Figure II-11 : Architecture du modèle CBOW [15]

II.5.4.2.1Exemple expliquant le fonctionnement de modèle CBOW

CBOW est très utile pour identifier les mots qui manquent dans une phrase, voici un exemple détaillé qui explique l'architecture du modèle CBOW et son fonctionnement :

Supposant qu'on a le contexte des mots suivant : « Latent Dirichlet Allocation », on va sélectionner le mot «Allocation » qui est le mot qui manque qu'on veut chercher (mot cible),

La taille de la couche d'entrée et la couche de sortie dépond de la taille du corpus, et le nombre de neurones dans la couche cachée représente la dimensionnalité et il varie entre un mot et la taille du vocabulaire. Voici un exemple du modèle CBOW illustré dans la **figure II-12** et les différentes étapes pour arriver à obtenir le mot cible :

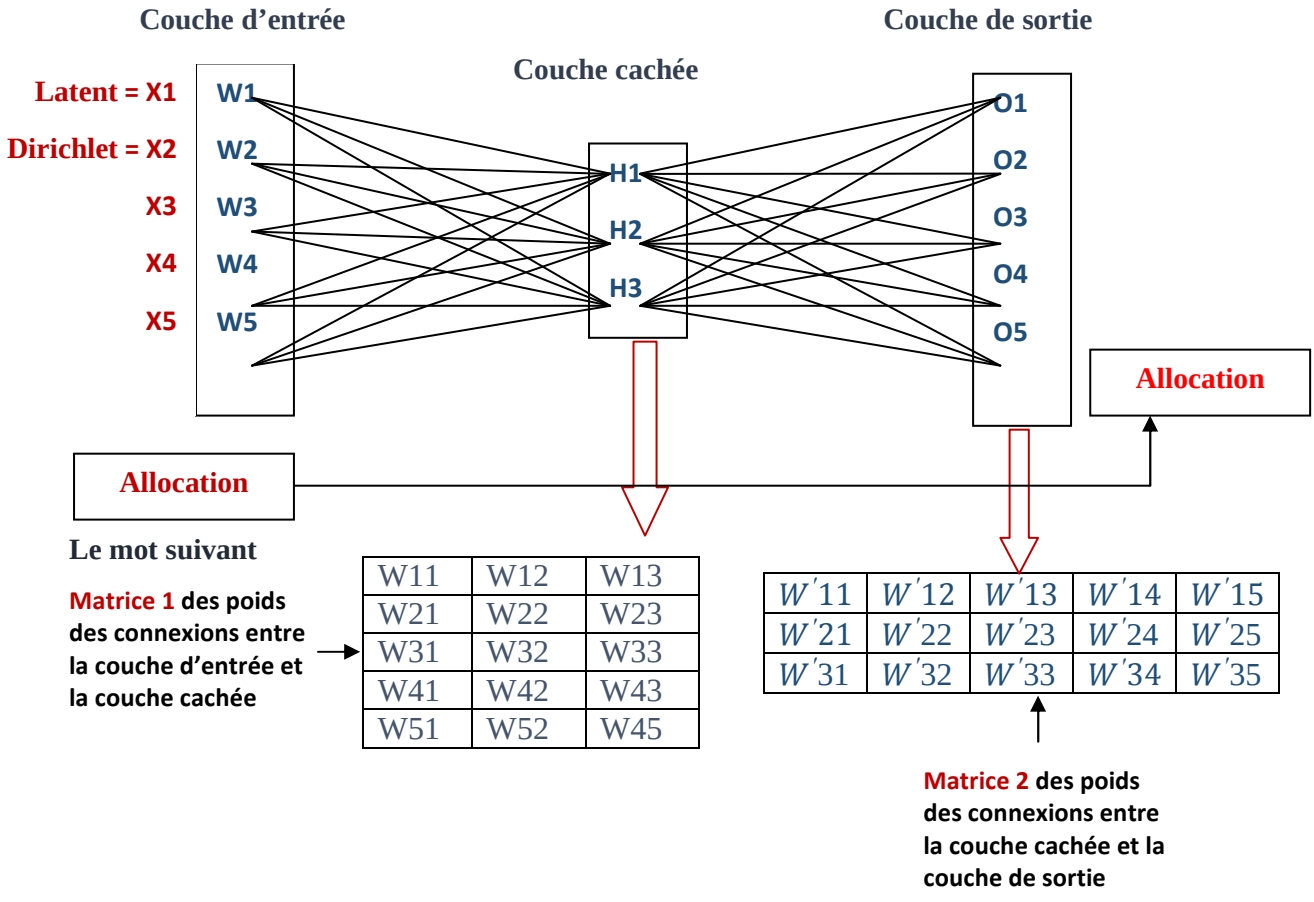


Figure II-12 : Architecture détaillée du modèle CBOW

x1, x2,…, x3 sont des valeurs attribuées aux termes w1, w2,…, w3, et x_1 et x_2 sont égaux à 1, et les autres sont fixés à 0.

Wij = c'est le poids des connexions entre la couche d'entrée et la couche cachée.

$W'ij$: C'est le poids des connexions entre la couche d'entrée et la couche cachée

CBOW passe par deux phases :

a. **Propagation vers l'avant (Forward propagation) :** dans cette phase nous allons calculer les poids de la couche d'entrée vers la couche cachée, ensuite de la couche cachée vers la couche de sortie :

a.1 De la couche d'entrée vers la couche cachée :

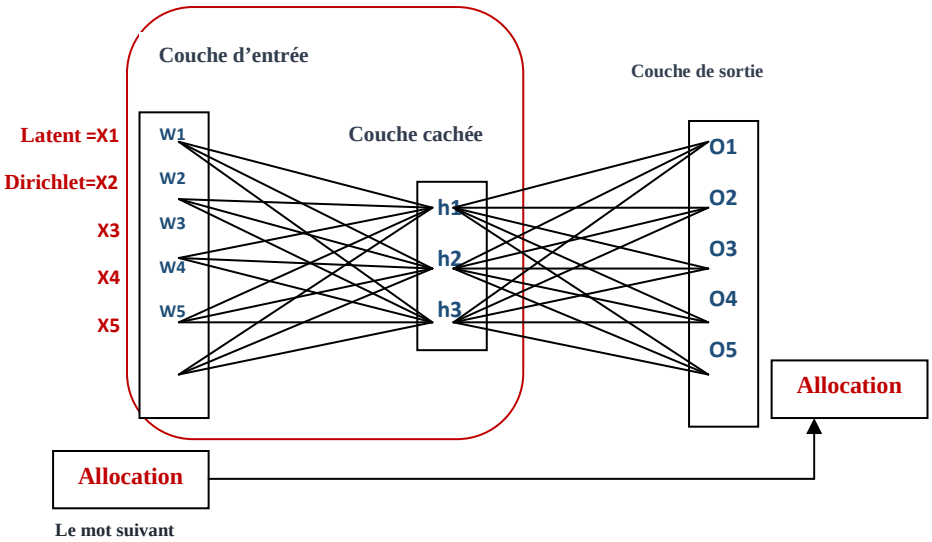


Figure II-13 : propagation vers l'avant : De la couche d'entrée vers la couche cachée

$$h_1=(w_{11}x_1+w_{21}x_2 + w_{31}x_3 + w_{41}x_4 + w_{51}x_5) \quad \text{(III-11)}$$

$$h_2=(w_{12}x_1+w_{22}x_2 + w_{32}x_3 + w_{42}x_4 + w_{52}x_5) \qquad \begin{vmatrix} h_1 \\ h_2 \\ h_2 \end{vmatrix} = w^T x =$$

$$\begin{vmatrix} w_{11}+w_{21}+w_{31}+w_{41}+w_{51} \\ w_{12}+w_{22}+w_{32}+w_{42}+w_{52} \\ w_{13}+w_{23}+w_{33}+w_{43}+w_{53} \end{vmatrix} \begin{vmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{vmatrix} \quad \text{(III-12)}$$

$$h_3=(w_{13}\ x_1+w_{23}x_2 + w_{33}x_3 + w_{43}x_4 + w_{53}x_5) \quad \text{(III-13)}$$

$w^T x$ est la matrice transposée de matrice1

a.2 De la couche cachée vers la couche de sortie:

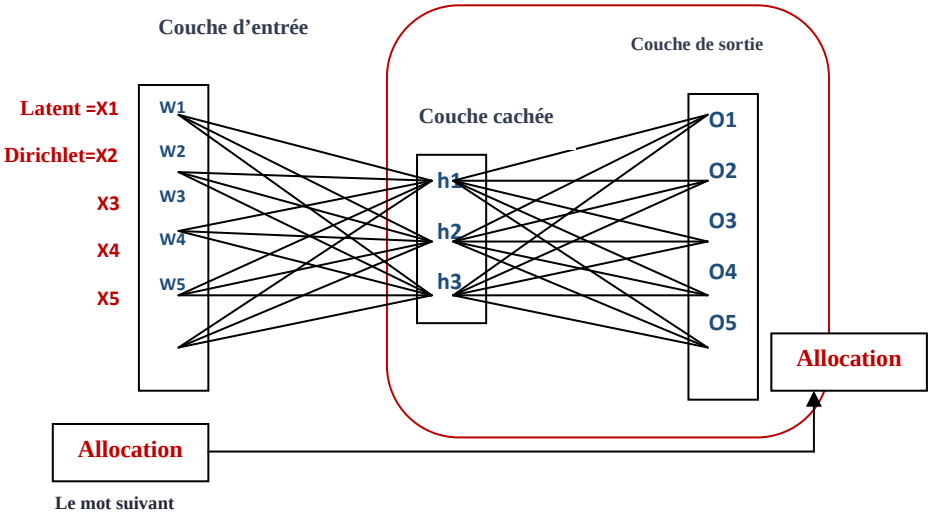


Figure II-14 : Propagation vers l'avant : De la couche cachée vers la couche de sortie

Le calcule de poids pour la couche de sortie se fait en deux parties :

Partie1 : appliquer la fonction de sommation : c'est la somme des poids de la couche cachée multipliée par les poids des connexions entre cette couche et la couche de sortie, soit Net (Oi) ce poids calculé.

Net (O1) $=u_1 = w'_{11}h_1 + w'_{21}h_2 + w'_{31}h_3$ (III-14)

Net (O2) $=u_2 = w'_{12}h_1 + w'_{22}h_2 + w'_{32}h_3$ (III-15)

Net (O4)= $u_3 = w'_{13}h_1 + w'_{23}h_2 + w'_{33}h_3$ = Net (O=)w^Th = $\begin{vmatrix} w'_{11} & w'_{21} & w'_{31} \\ w'_{12} & w'_{22} & w'_{32} \\ w'_{13} & w'_{23} & w'_{33} \\ w'_{14} & w'_{24} & w'_{34} \\ w'_{15} & w'_{25} & w'_{35} \end{vmatrix} * \begin{vmatrix} h_1 \\ h_2 \\ h_2 \end{vmatrix}$ (III-16)

Net (O3) $=u_4 = w'_{14}h_1 + w'_{24}h_2 + w'_{34}h_3$ (III-17)

Net (O5) $=u_5 = w'_{15}h_1 + w'_{25}h_2 + w'_{35}h_3$ (III-18)

w^Th est la matrice transposée de **matrice2.**

Partie2 : le calcule de la softmax de sortie :

$$\textbf{Out}(O_1)= y_1=\frac{e^{Net(O_1)}}{e^{Net(O_1)}+e^{Net(O_2)}+e^{Net(O_3)}+e^{Net(O_4)}+e^{Net(O_5)}}=\frac{e^{u_1}}{e^{u_1}+e^{u_2}+e^{u_3}+e^{u_4}+e^{u_5}} \quad \text{(III-19)}$$

$$\textbf{Out}(O_2)= y_2=\frac{e^{Net(O_2)}}{e^{Net(O_1)}+e^{Net(O_2)}+e^{Net(O_3)}+e^{Net(O_4)}+e^{Net(O_5)}}=\frac{e^{u_2}}{e^{u_1}+e^{u_2}+e^{u_3}+e^{u_4}+e^{u_5}} \quad \text{(III-20)}$$

De la même façon on peut calculer $y_3, y_4\ et\ y_5$

La fonction softmax pour un mot W_j (cible) donnant les mots de contexte W_I est alors calculée comme suit :

$$\textbf{P}(W_j\ |W_I\ )= y_j = \frac{e^{Net(O_j)}}{\sum_{j'=1}^{V} Net(O_{j'})} = \frac{e^{(u_j)}}{\sum_{j'=1}^{V} e^{(u_{j'})}} \quad \text{(III-21)}$$

Partie 2 : calcule d'erreur : nous utilisant le calcul d'erreur pour corriger les poids des connexions entre les trois couches.

Soit w_O le mot cible (Allocation), w_I le contexte des mots donné (Latent Dirichlet), et V la taille de contexte d'entrée. L'objectif est d'optimiser la probabilité conditionnelle du mot w_O sachant w_I, la fonction de perte est définie comme suit :

$$\textbf{Max P}(w_O|w_I)\textbf{= max (log}(y_{j^*})\textbf{)} \quad \text{(III-22)}$$

Par exemple :

$$\textbf{E}(O_4)\textbf{= log P}(w_4|w_I)\textbf{ = log }\left(\frac{e^{u_4}}{e^{u_1}+e^{u_2}+e^{u_3}+e^{u_4}+e^{u_5}}\right) \quad \text{(III-23)}$$

$$= log_e(e^{u_4}) - log_e(e^{u_1}+e^{u_2}+e^{u_3}+e^{u_4}+e^{u_5}) \quad \text{(III-24)}$$

$$= u_4-(e^{u_1}+e^{u_2}+e^{u_3}+e^{u_4}+e^{u_5}) \quad \text{(III-25)}$$

Comme on veut aussi minimiser l'erreur comme suit :

$$\textbf{E = - log P}(w_4|w_I)\textbf{,}\ \text{alors} \quad \text{(III-26)}$$

$$\mathbf{E}(\boldsymbol{O_4}) = -(\boldsymbol{e^{u_1} + e^{u_2} + e^{u_3} + e^{u_4} + e^{u_5}}) - \boldsymbol{u_4} \quad \text{(III-27)}$$

On peut généraliser comme suit :

$$\mathbf{E} = \log \sum_{j'=1}^{V} \boldsymbol{e^{u_{j'}}} - \boldsymbol{u_{j^*}} \quad \text{(III-28)}$$

Où, $\boldsymbol{j^*}$ est l'index du mot cible dans la couche cachée

Maintenant prenant la dérivation de **E** par rapport à $\boldsymbol{u_4}$ nous allons avoir :

$$\frac{\boldsymbol{dE(O_4)}}{\boldsymbol{dE(u_4)}} = \frac{\boldsymbol{u_4}}{\boldsymbol{e^{u_1}+e^{u_2}+e^{u_3}+e^{u_4}+e^{u_5}}} - \frac{\boldsymbol{dE(u_4)}}{\boldsymbol{dE(u_4)}} = \mathbf{Out}(\boldsymbol{O_4}) - \frac{\boldsymbol{dE(u_4)}}{\boldsymbol{dE(u_4)}} = \boldsymbol{y_4 - 1} \quad \text{(III-29)}$$

Nous remarquons que $\boldsymbol{u_4}$ est la valeur du mot cible $\boldsymbol{O_4}$ (Allocation), c'est pour ça sa valeur est égale à**1.**

Nous pouvons généraliser cette dérivation comme suit:

$$\frac{\boldsymbol{dE}}{\boldsymbol{du_j}} = \frac{\boldsymbol{d}\,(\boldsymbol{log \sum_{j'=1}^{V} e^{u_{j'}} - u_{j^*}})}{\boldsymbol{du_j}} = \frac{\boldsymbol{e^{u_j}}}{\boldsymbol{\sum_{j'=1}^{V} e^{u_{j'}}}} - \frac{\boldsymbol{d(u_{j^*})}}{\boldsymbol{du_j}} = \boldsymbol{y_j - t_j := e_j} \quad \text{(III-30)}$$

$\boldsymbol{t_j} = \frac{\boldsymbol{d(u_{j^*})}}{\boldsymbol{du_j}}$

$\boldsymbol{t_j}$ **= 1** si **(**$\boldsymbol{t_j = t_{j^*}}$**)** si non $\boldsymbol{t_j}$**=0**

b. La rétro propagation (back-propagation) :

- ✓ **de la couche de sorite vers la couche cachée :** Cette phase nous permet de mettre à jour les poids de tous les neurones de la couche cachée vers la couche de sortie ($W'_{11}, W'_{12}, W'_{13}, \ldots, W'_{35}$).

Etape1 : prendre le gradient de 'E' par rapport à W'_{11}.

$$\frac{\boldsymbol{dE(O_1)}}{\boldsymbol{dW'_{11}}} = \frac{\boldsymbol{dE(O_1)}}{\boldsymbol{d_{u_1}}} * \frac{\boldsymbol{d_{u_1}}}{\boldsymbol{dW'_{11}}} \quad \text{(III-31)}$$

$$\frac{\boldsymbol{dE(O_1)}}{\boldsymbol{d_{u_1}}} = \boldsymbol{(y_1 - 0) = e_1} \quad \text{(III-32)}$$

$$\frac{\boldsymbol{d_{u_1}}}{\boldsymbol{dW'_{11}}} = \frac{\boldsymbol{d(W'_{11}h_1+W'_{21}h_2+W'_{31}h_3)}}{\boldsymbol{dW'_{11}}} = \boldsymbol{h_1} \quad \text{(III-33)}$$

$$\frac{\boldsymbol{dE(O_1)}}{\boldsymbol{dW'_{11}}} = \frac{\boldsymbol{dE(O_1)}}{\boldsymbol{d_{u_1}}} * \frac{\boldsymbol{d_{u_1}}}{\boldsymbol{dW'_{11}}} = \boldsymbol{e_1 * h_1} \quad \text{(III-34)}$$

Etape 2 : mettre à jour le poids W'_{11} :

$$W'_{11}{}^{new} = W'_{11} - \eta \frac{dE(O_1)}{dW'_{11}} = W'_{11} - \eta (e_1 - h_1) \quad \text{(III-35)}$$

η représente un rang d'apprentissage $\in$ [0,1]

De la même façon nous pouvons calculer le poids de $W'_{12}, W'_{13}, W'_{21}\ldots, W'_{53}$.

- ✓ **De la couche cachée vers le couche d'entrée :** Cette phase nous permet de mettre à jour les poids de tous les neurones de la couche cachée vers la couche de sortie ($W_{11}, W_{12}, W_{13},\ldots, W_{35}$).

Etape1 : prendre le gradient de 'E' par rapport à W_{11}.

$$\frac{dE}{dW_{11}} = \frac{dE}{dh_1} * \frac{dh_1}{dW_{11}} \quad \text{(III-36)}$$

$$\frac{dh_1}{dW_{11}} = \left(\frac{dE}{du_1} * \frac{du_1}{dh_1}\right) + \left(\frac{dE}{du_2} * \frac{du_2}{dh_1}\right) + \left(\frac{dE}{du_3} * \frac{du_3}{dh_1}\right) + \left(\frac{dE}{du_4} * \frac{du_4}{dh_1}\right) + \left(\frac{dE}{du_5} * \frac{du_5}{dh_1}\right) \quad \text{(III-37)}$$

$$= (e_1 W'_{11}) + (e_2 W'_{12}) + (e_3 W'_{13}) + (e_4 W'_{14}) + (e_5 W'_{15}) \quad \text{(III-38)}$$

$\left(\frac{dE}{dh_1} * \frac{du_1}{dh_1}\right)$ est l'erreur propagée à partir du neurone de mot de sortie O_1 de la couche de sortie

$\left(\frac{dE}{dh_2} * \frac{du_2}{dh_1}\right)$ est l'erreur propagée à partir du neurone de mot de sortie O_2 de la couche de sortie

…

$$\frac{dh_1}{dW_{11}} = \frac{d(W_{11}x_1 + W_{21}x_2 + W_{31}x_3 + W_{41}x_4 + W_{51}x_5)}{dW_{11}} = x_1 \quad \text{(III-39)}$$

$$\frac{dE}{dW_{11}} = \left(\frac{dE}{dh_1} * \frac{dh_1}{dW_{11}}\right) = ((e_1 W'_{11}) + (e_2 W'_{12}) + (e_3 W'_{13}) + (e_4 W'_{14}) + (e_5 W'_{15})) * (x_1)$$

(III-40)

Etape 2 : mettre à jour le poids W_{11}

$$W_{11}{}^{new} = W_{11} - \eta \frac{dE}{dW_{11}} \quad \text{(III-41)}$$

$$= W_{11} - \eta((e_1 W'_{11}) + (e_2 W'_{12}) + (e_3 W'_{13}) + (e_4 W'_{14}) + (e_5 W'_{15})) * (x_1) \quad \text{(III-42)}$$

De la même manière nous pouvons calculer $W_{12}, W_{13}, W_{21}\ldots, W_{53}$

II.5.5 Le modèle Glove (Global Vectors for Word Representation)

GloVe est un algorithme d'apprentissage non supervisé qui permet de construire des word embeddings. Il sert à effectuer des calcules de co-occurrence de mots à partir d'un corpus, et les représentations résultantes présentent des sous-structures linéaires intéressantes de l'espace vectoriel des word embeddings.

Les auteurs de Glove (Pennington et al.) [26] montrent que le rapport des probabilités de co - occurrence de deux mots est ce qui contient des informations et vise à présenter cette information en tant que différences vectorielles.

Pour ce faire, ils proposent un objectif pondéré des moindres carrés **J** qui vise directement à minimiser la différence entre le produit des points des vecteurs de deux mots et le logarithme de leur nombre de co-occurrences :

$$\mathbf{J}=\sum_{i,j=1}^{V} \boldsymbol{f}(\boldsymbol{X_{ij}})(\boldsymbol{w_i^T \widetilde{w}_J} + \boldsymbol{b_i} + \boldsymbol{\widetilde{b}_J} - \boldsymbol{log X_{ij}})^2 \qquad \text{[II-43]}$$

Où $\boldsymbol{w_i}$ et $\boldsymbol{b_i}$ sont le vecteur du mot et bias respectivement du mot **i**, $\boldsymbol{\widetilde{w}_j}$ et $\boldsymbol{b_j}$ sont le vecteur de mot de contexte et biais respectivement du mot **j**, $\boldsymbol{x_{ij}}$ est le nombre de fois que le mot **i** se produit dans le contexte du mot **j** et **f** est une fonction de pondération qui attribue un poids relativement inférieur aux co-occurrences rares et fréquentes.

GloVe prend une telle matrice plutôt que l'ensemble du corpus comme entrée [16].

Le modèle glove n'as pas une grande différence entre le modèle word2vec, les deux modèles apprennent des représentations géométriques (vecteurs) des mots à partir de leur co-occurrence.

La différence entre ces deux modèles est que word2vec est un modèle «prédictif» (il utilise les probabilités), alors que GloVe est un modèle basé sur la co-occurrence.

II-6 Le word embedding dans la recherche d'information (RI)

La recherche d'informaion ad hoc a été abordée plus directement en 2013. Nous présentons dans cette section les études récentes qui ont été réalisées pour l'extension du modèle de recherche d'information traditionnel en intégrant le word embeding.

Dans les modèles de recherche traditionnels les termes sont représentés sous forme de symboles discrets qui ne peuvent pas être comparés directement les uns aux autres. De manière conceptuelle, cela équivaut à une représentation unique dans lequel chaque terme est représenté comme un vecteur épars avec une dimension égale à la taille du vocabulaire (chaque dimension correspondante à un mot unique), cette technique est connu sous la représentation one-hot qui permet de représenter un terme donné **t** avec un vecteur de **0** et définissons l'indice correspondant à **t** à **1**. Dans une telle représentation, tous les termes sont orthogonaux et équidistants les uns aux autres et, par conséquent, il est difficile de reconnaître la similarité des termes dans l'espace vectoriel. Cela a conduit à un problème de désordre de vocabulaire dans lequel un système de recherche d'information (SRI) ne peut pas reconnaître lorsque les termes sont distincts mais liés qui se produisent dans la requête et le document.

Pour remédier à ce problème, la technique de word embedding est apparu, elle permet de représenter chaque symbole en tant que vecteur à faible dimension (par exemple, des centaines de dimensions), les word embeddings représentent une similitude sémantique et syntaxique dans la mesure où les word embeddings similaires seront proches l'un de l'autre dans l'espace vectoriel. Cela permet d'effectuer des opérations algébriques simples entre les vecteurs du word embedding qui reflètent la signification du mot, par exemple, wv ("Madrid") - wv ("Espagne") + wv ("France") devrait être proche à wv ("Paris").

Différentes études ont été effectuées dans la recherche d'information en intégrant le word embedding. Le tableau suivant présente les différentes études exploitant la technique de word embedding dans la RI Ad-hoc :

Tâche	**Etudes**
Recherche Ad-hoc	ALMasri et al. (2016), Amer et al. (2016), BWESG (Vulic and Moens (2015)), Clinchant and Perronnin (2013), Diaz et al.(2016), GLM (Ganguly et al. (2015)), Mitra et al. (2016), Nalisnick et al. (2016), NLTM (Zuccon et al. (2015)), Rekabsaz et al.(2016), Roy et al. (2016), Zamani and Croft (2016a), Zheng and Callan (2015)

Tableau II-1: Tâches de la RI ad-hoc résolues par le word embedding.

Ganguly et al. (2015) proposent un modèle de langage généralisé (GLM) pour intégrer des word embedding avec une modélisation de langage de probabilité de requête. La similarité sémantique entre les termes de requête et de document / collection est mesurée par la similarité du cosinus entre les word embedding construites à partir du word2vec CBOW.

Roy et al. (2016) proposent trois approches pour utiliser les word embeddings dans l'extension de requête basée sur K-Nearest Neighbor (KNN).

La première approche calcule les K voisins les plus proches pour chaque terme de requête basée sur la représentation des word embeddings. Pour chaque voisin le plus proche, ils calculent la similarité moyenne du cosinus par rapport à tous les termes de la requête, en sélectionnant les termes de la top-K selon le score de cosinus moyen.

La deuxième approche consiste à réduire l'espace de vocabulaire considéré par KNN en considérant uniquement les termes apparaissant dans les documents M pseudo-pertinents les plus récents récupérés par la requête.

Dans la troisième approche, une stratégie d'élagage itérative (coûteuse) est appliquée pour réduire le nombre de voisins les plus proches, en supposant que les voisins les plus proches sont semblables les uns aux autres.

Les résultats négatifs montrent les word embeddings ne produisent pas d'améliorations dans cette formulation.

Rekabsaz et al. (2016) recommande de choisir des termes similaires en fonction d'un seuil de similarité plutôt que de KNN (K-Nearest Neighbor = k voisins plus proches). Ils choisissent le seuil en le définissant pour n'importe quel terme, le nombre prévu de termes liés dans le seuil est égal au nombre moyen de synonymes sur tous les mots dans la langue. Cette méthode évite de contraindre la technique KNN comme dans (Roy et al., 2016). Ils utilisent le modèle word2vec skip-gram pour produire les word embeddings qui sont utililisés pour calculer la similarité du cosinus. Les expériences sur TREC 6-8 et HARD 2005 intègrent cette méthode dans un modèle de langue de traduction pour la recherche ad hoc et se compare à une version de base de modèle de langue et à un modèle de langue de traduction qui utilise KNN pour sélectionner des mots similaires. Le modèle de langue de traduction basé sur le seuil atteint la plus haute précision moyenne (MAP).

Zuccon et al. (2015) proposent le modèle de traduction de langage neuronale (NLTM) qui intègre les word embeddings. Ce modèle consiste à estimer la probabilité de traduction entre

les termes comme le cosinus similarité des deux termes divisés par la somme de la similarité de cosinus entre le terme de traduction et tous les termes dans le vocabulaire. Le modèle de traduction utilisé avant l'arrivé des word embeddings est connu sous le nom de l'Information Mutuelle (MI) pour estimer la probabilité de traduction. Les expériences d'évaluation de l'approche NLTM sur la recherche ad hoc sont rapportées sur les collections TREC AP87-88, WSJ87-92, DOTGOV et MedTrack a fournit des améliorations modérées par rapport aux systèmes MI. L'analyse des différents paramètres des modèles des word embeddings montrent que la dimensionnalité de l'espace vectoriel, la taille de la fenêtre de contexte et le type du modèle (CBOW ou skip-gram) n'ont pas d'impact sur le modèle de langage de traduction neuronal NLTM. Par contre le choix du corpus pour l'apprentissage des word embeddings montre que l'efficacité du modèle NLTM semble généralement plus élevée lorsque les word embeddings sont estimés à l'aide de la même collection dans laquelle la recherche doit être effectuée.

Zheng et Callan (2015) utilisent les word embeddings obtenus par le modèle word2vec cbow pour construire des vecteurs de « caractéristiques » pour chaque mot de requête. Ce vecteur est obtenu par la soustraction entre la moyenne des vecteurs des word embeddings des termes de requête et les vecteurs des word embeddings des termes donnés. Cette technique consiste à prédire un poids cible pour chaque terme de requête étant donné son vecteur caractéristique en utilisant la régression. Le poids cible de chaque terme est tiré des jugements de pertinence. On suppose que des jugements de pertinence sont disponibles pour la même collection.

Des expériences sur la recherche ad hoc sont effectuées sur quatre (4) collections de test TREC: Robust04, WT10t, GOV2 et ClueWeb09-Cat-B, en utilisant trois (03) modèles de recherche : BM25, la modélisation de langage unigram (LM) et le modèle de dépendance séquentielle. La recherche montre que les word embedding peuvent être efficacement exploités pour améliorer la précision de recherche ad hoc sans nécessiter une modélisation de réseau neuronal de bout en bout. En ce qui concerne la dimensionnalité des word embeddings, Zheng et Callan (2015) trouvent que 100 dimensions fonctionnent mieux pour estimer les poids des termes, mieux que 300 et 500.

II-7 Conclusion

L'utilisation des word embeddings ont fait un progrès dans la recherche d'information, dans la construction des word embeddings les modèles de langue neuronaux ont contribué dans les différents modèles de word embedding. Nous avons présenté dans ce chapitre ces modèles

tels que C&W, Word2vec et Glove ainsi les différents types de réseaux de neurone qui ont contribué à l'architecture de ces modèles. Nous avons vu aussi quelques travaux qui ont exploité le word embedding dans la recherche d'information (RI).

Le troisième chapitre est consacré à l'implémentation de notre approche, nous présentons aussi les différentes plateformes qu'on a utilisé pour la réaliser ainsi que les résultats que nous avons obtenus.

Chapitre III
L'Implémentation et résultats

III.1 Introduction

Il existe plusieurs méthodes pour améliorer les résultats de la recherche d'information, notre travail se focalise sur l'extension du modèle de recherche d'information. Dans ce chapitre nous présentons l'approche proposé qui se base le word embedding, son implémentation ainsi que les outils de recherche qu'on a utilisé dans notre travail, et enfin nous observons les résultats obtenus avant et après l'intégration de notre approche.

III.2 l'environnement de développement

Nous allons présenter dans cette section les différents outils utilisés dans notre environnement à savoir la plateforme TERRIER, le langage java ainsi l'environnement Netbeans.

III.2.1 L'outil de recherche TERRIER

III.2.1.1 Définition

Terrier est un outil de recherche gratuit, très flexible, efficace et effectif, facilement déployable sur de grandes masses de collections de documents. IL implémente plusieurs fonctionnalités de recherche et fourni une plateforme idéal pour le développement rapide et l'évaluation pour les applications de recherche à grande échelle.

C'est une plate forme, complète et transparente pour la recherche et l'expérimentation dans la recherche de texte, cette recherche peut être effectuée facilement sur les collections de tests standard TREC et CLEF [1].

Terrier peut indexer de large collections de documents jusqu'à 50 millions de documents, il est écrit en Java, fonctionne sous différentes plateformes: Windows, Mac OS X, Linux, Unix.

III.2.1.2 L'architecture du terrier

L'architecture de Terrier distingue les deux phases classiques: l'indexation et la recherche.

•Un corpus documentaire est fourni en entrée au module d'indexation

•Les documents de la collection passent par un ensemble de prétraitements tels que la tokenisation

•Les tokens sont ensuite injectés dans une chaine de traitement TermPipeLine, à savoir le StopWords pipeline pour l'élimination des mots vides de sens, ou encore les Stemming pipeline et qui dépendent de la langue en question.

•La phase d'indexation conduit à la construction de l'index (Data Structures)

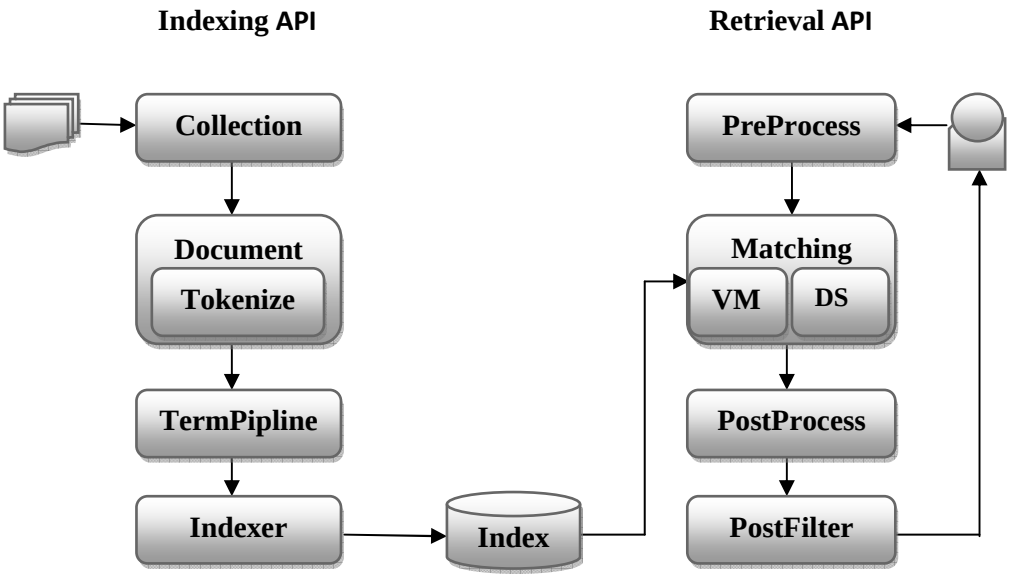


Figure III-1 : L'architecture du Terrier

a. **L'indexation :** l'indexation dans Terrier passe par ces étapes suivantes :
    1. **Collection :** lire les documents
    2. **Document :** extraire le texte brut
    3. **Tokenizer :** fragmenter (tokenizer) le texte extrait
    4. **TermPipline : (chaine de traitement)** traiter les tokens
    5. **Indexer :** indexe les tokens traités
b. **La recherche :** ses étapes sont résumées ci-dessous :
    1. **Manager:** Analyse la requête utilisateur
    2. **Matching:** Recherche les documents
    3. **VMs** et **DSMs:** Marque (Score) les documents
    4. **PostProcess:** Post-Traitement, met à jour les résultats globaux
    5. **PostFilter:** met à jour chaque document individuel retrouvé

III.2.1.3 Structure du dossier Terrier

Ci-dessous les dossiers qu'on trouve dans Terrier et la description de chacun d'eux

Bin	Contient des fichiers script
Doc	Englobe une documentation complète de Terrier (horsligne)
Etc	Contient les fichiers de configuration
Lib	Contient des librairies Java compilées
Share	Contient la stopword list ainsi que les tests
Src	Contient le code source
Var	C'est ici qu'est stocké l'index et les résultats de l'évaluation

Tableau III-1 : La structure du dossier Terrier [4]

III.2.1.4 Installation du Terrier

a) Sous Windows:

Pré-requis:

Installer la JRE de java (version 1.7 ou plus par exemple, pour la version 4.1 de Terrier), et configurer la variable d'environnement.

Installer:

Télécharger une version de terrier, à partir du site officiel http://www.terrier.org/,

Extraire le fichier .zip téléchargé,

Télécharger une collection de test.

Indexation:

Appeler cette collection à partir du fichier «terrier_setup.bat» en spécifiant le chemin de cette collection.

b) Sous linux:

Pré-requis:

Installer la JRE de java (version 1.7 ou plus par exemple, pour la version 4.1 de Terrier), et configure la variable d'environnement.

Installation:

Télécharger une version de terrier sur le site officiel terrier.org

Télécharger une collection de tests.

Remarque: TREC est la collection utilisée par défaut dans terrier mais on peut indexer d'autres collections de tests avec

Indexation: (utilisation de batch (TREC) terrier)

Accéder au dossier terrier en tapant la commande:

Cd «chemin menant au dossier de terrier»

III.2.1.5 Utilisation de Terrier dans l'indexation, la recherche et l'évaluation

a. **L'indexation :** voici les différentes étapes de l'indexation sur la ligne de commande :

- Aller dans le dossier terrier :

  cd terrier-2.1

- On spécifie le chemin de la collection:

  ./bin/trec_setup.sh «chemin vers la collection»

Cela donne comme résultat un fichier **collection.spec** qui est crée au niveau du dossier **etc** et il contient la collection à indexer.

Pour indexer une collection on fait appel au fichier script **trec_terrier.sh** et en ajoutant le **–i** comme suit:

./bin/trec-terrier.sh –i

Ou bien on peu indexer en ajoutant **–i –j (./bin/trec-terrier.sh –i –j)** pour plus de rapidité

Par défaut, les résultats d'indexation sont créés dans le dossier **var/index** de terrier

Une fois l'indexation de la collection est terminée, on peut vérifier l'index en utilisant la commande **--printstats** comme suit:

./bin/trec_terrier --printstats

Cette commande permet d'afficher les statistiques de l'index : Le nombre de documents, le nombre de tokens, le nombre de termes

Figure III-2 : les statistiques de l'index

On vérifier si on peut avoir des résultats d'une recherche en tapant une requête en utilisant le script :

./bin/interactive_terrier

Figure III-3 : résultats pour la requête « «compressed »

A la réponse pour la requête «compressed », terrier a trouvé que le document AP880719-0086 contient 40733 fois le mot «compressed » qui est le plus pertinent avec un score 8.557.

b. La recherche : Pour faire la recherche sur une collection indexée, on suit les étapes suivantes:

1. Effectuer certaines configurations. La plupart des fonctionnalités de terrier sont contrôlées par des propriétés. On peut configurer ceci dans le fichier **Terrier.Propreties.Sample** qui se trouve dans le dossier **etc** de terrier, ou bien spécifier chaque propriété dans la ligne de commande. Dans ce qui suit, nous allons utiliser la ligne de commande pour spécifier les propriétés appropriées.

Pour faire la recherche et évaluer les résultats d'un lot de requêtes, on a besoin de savoir:

- Le chemin d'accès aux requêtes (aussi connu sous le nom de topic files), spécifié en utilisant le fichier **trec.topics** qui se trouve dans le dossier **etc**, au début ce fichier est vide, on a qu'à écrire le chemin des topics dedans.
- Le modèle de poids (eg: TF-IDF) à utiliser, spécifié en utilisant **trec.model**, si on veut choisir un modèle on a qu'à enlever le « # » qui se trouve au début de chaque modèle
- Le fichier des jugements de pertinence correspondant (qrels) aux requêtes, spécifié par **trec.qrels**, de la même manière on va ajouter les qrels en écrivant le chemin vers ces qrels.

Pour commencer la recherche, l'option **–r** demande à Terrier d'exécuter une recherche sur un lot, c'est-à-dire rechercher les documents estimés être les plus pertinents pour chaque requête dans le fichier topics file. On effectue la recherche en exécutant la commande suivante

./bin/trec_terrier –r

Figure III-4 : Résultats de la recherche

Si tout va bien, le résultat de la recherche est sauvegardé dans le fichier **.res** file qui ce trouve dans le dossier **var/results** appelé TF_IDF_2.res (pour le modèle TF-IDF). On appelle chaque **.res** un exécutable.

On peut aussi configurer plus d'options en utilisant la ligne de commandes:

bin/trec_terrier -r -c 10.99

- Le paramètre **–r** demande à terrier de faire la recherche
- Le paramètre **–c** informe terrier sur les paramètres du modèle de poids

c. L'évaluation : L'évaluation des résultats de recherche obtenus s'effectue en utilisant l'option **–e** de trec_terrier:

./bin/trec_terrier –e

Figure III-5 : Résultats de l'évaluation

Terrier va aller dans le dossier **var/results**, évalue chaque fichier **.res** et enregistre les résultats dans un fichier **.eval** nommé comme le fichier **.res** correspondant.

III.2.2 Le langage java

Java est un langage de programmation moderne développé par **Sun Microsystems** (aujourd'hui racheté par **Oracle.** Il ne faut surtout pas le confondre avec JavaScript (langage de scripts utilisé principalement sur les sites web), car Java n'a rien à voir.

Une de ses plus grandes forces est son excellente portabilité : une fois votre programme créé, il fonctionnera automatiquement sous Windows, Mac, Linux, etc.

- **Java est simple :** Java est un langage simple à prendre en main, basé sur le langage C/C++ mais laisse de coté les sources de problèmes (pointeurs, structures, gestion de la mémoire, héritage multiple, macros etc.).

- **Java est orienté objet**

Tout est classe.

Héritage simple.

Une librairie plus de classes est fournie.

- **Java est distribué :** Java propose une API réseau standard. Cette dernière permet de manipuler, par exemple, les protocoles HTTP & FTP avec aisance.

Des API pour la communication entre des objets distribués (Remote Method Invocation).

Java est interprété : Un code source doit être traduit dans le langage machine avant d'être exécuté.

Compilateur: traduction du code source dans le langage binaire de la machine sur laquelle il sera exécuté.

Interpréteur: idem qu'un compilateur, sauf qu'il procède par étapes successives de compilation et exécution. Chaque instruction est compilée puis exécutée, puis le tour à l'instruction qui suit etc.

Compilateur Java traduit le code source Java en bytecode (code portable). Par la suite un interpréteur Java spécifique à une machine donnée (Java Virtual Machine : JVM Machine Virtuelle), traduit et exécute le bytecode.

Java est indépendant de l'architecture : Le bytecode généré n'est pas lié à un système d'exploitation en particulier. De ce fait, il peut être interprété très facilement sur n'importe quel environnement disposant d'une JVM.

- **Java est portable :** Java est portable d'un système à un autre : int 32 bits alors qu'en C/C++ 16 ou 32 bits.

- **Java est robuste :**

Pas de pointeurs.

Gestion de mémoire indépendante.

Mécanisme d'exceptions pour la gestion des erreurs.

Compilateur très contraignant.

Pas d'héritage multiple ni surcharge des opérateurs.

- **Java est sécurisée :** quatre niveaux de sécurité :

Langage et son compilateur contraignant.

Vérifier : vérifier le bytecode.

Class Loader : le chargeur de classe.

Security Manager : protection des fichiers et accès au réseau.

III.2.3 Netbeans

L'EDI NetBeans est un environnement de développement intégré, placé en open source par Sun en juin 2000, se concentrant principalement sur simplifier le développement d'applications Java. Il fournit du support pour tous les types d'applications Java, depuis le client riche jusqu'aux applications d'entreprises multicouches, en passant par les applications pour les mobiles supportant Java.

NetBeans est disponible sous Linux, Solaris , Mac OS X ou sous une version indépendante des systèmes d'exploitation (requérant une machine virtuelle Java). Un environnement Java Development Kit JDK est requis pour les développements en Java.

III.3.3.1Configuration Java Développent Kit (JDK)

Configurer les variables d'environnement PATH et JAVA_HOME pour pointer au répertoire qui contient java et javac

Pour cela allez dans:

Poste de travail→ propriétés →paramètres système avancés→ variables d'environnement

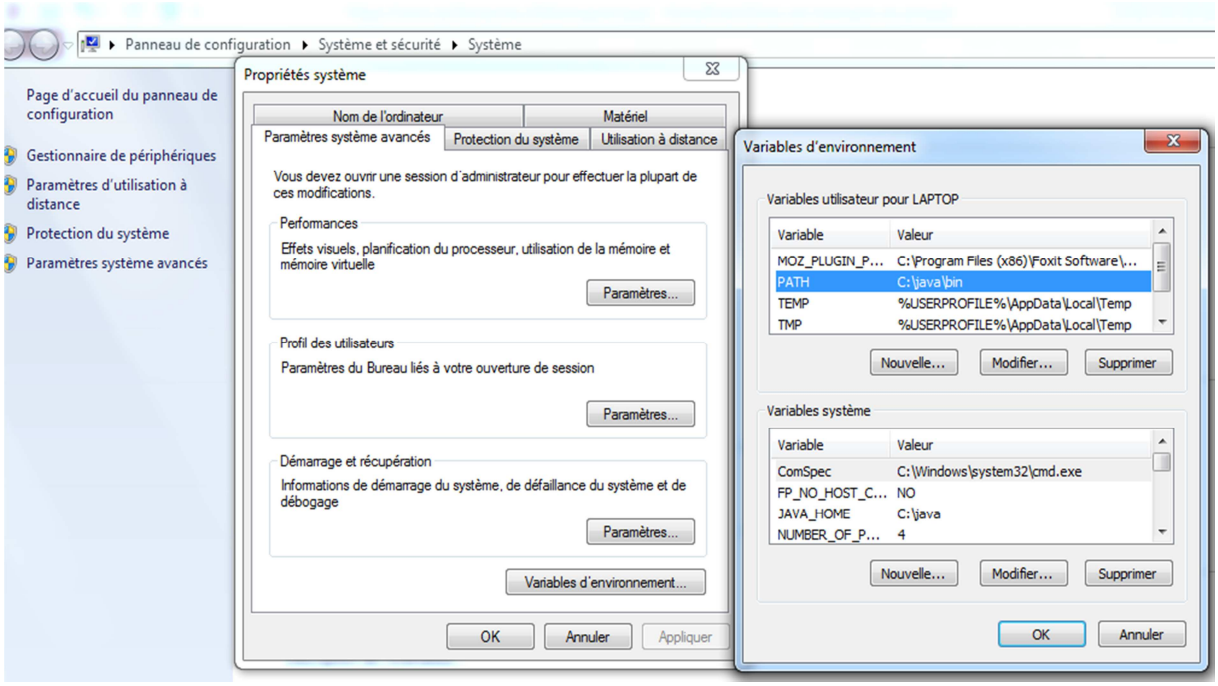


Figure III-6 : Configuration JDK

III.3.3.2 Créer un projet java

- La première étape consiste à créer un projet Java simple à l'aide d'Eclipse IDE.

- Suivez l'option Fichier -> Nouveau -> Projet et enfin sélectionnez JavaAssistant de projet dans la liste de l'assistant. Donner un nom de votre projet Expl:TerrierEssai en utilisant la fenêtre de l'assistant comme suit:

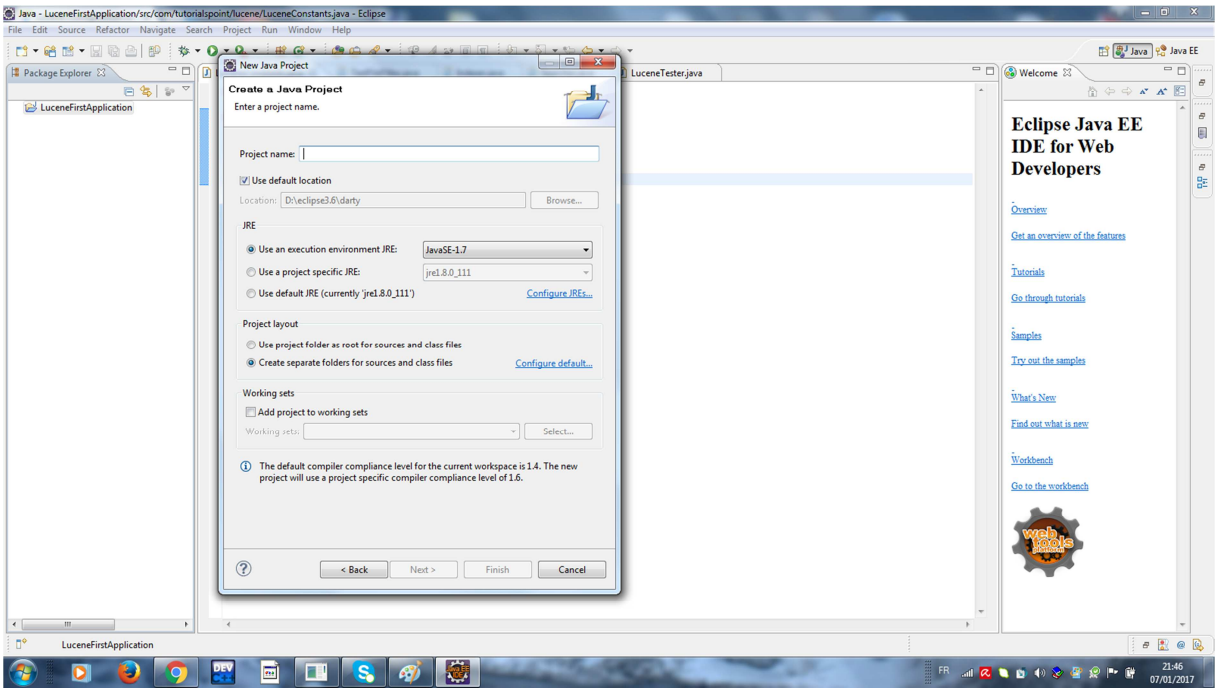


Figure III-7 : créer un projet Java

III.2.3.3 Ajouter les bibliothèques requises

Ajoutons la bibliothèque de TERRIER dans notre projet.

- cliquez droit sur le dossier Librairie, puis cliquez sur Add jar/Folders

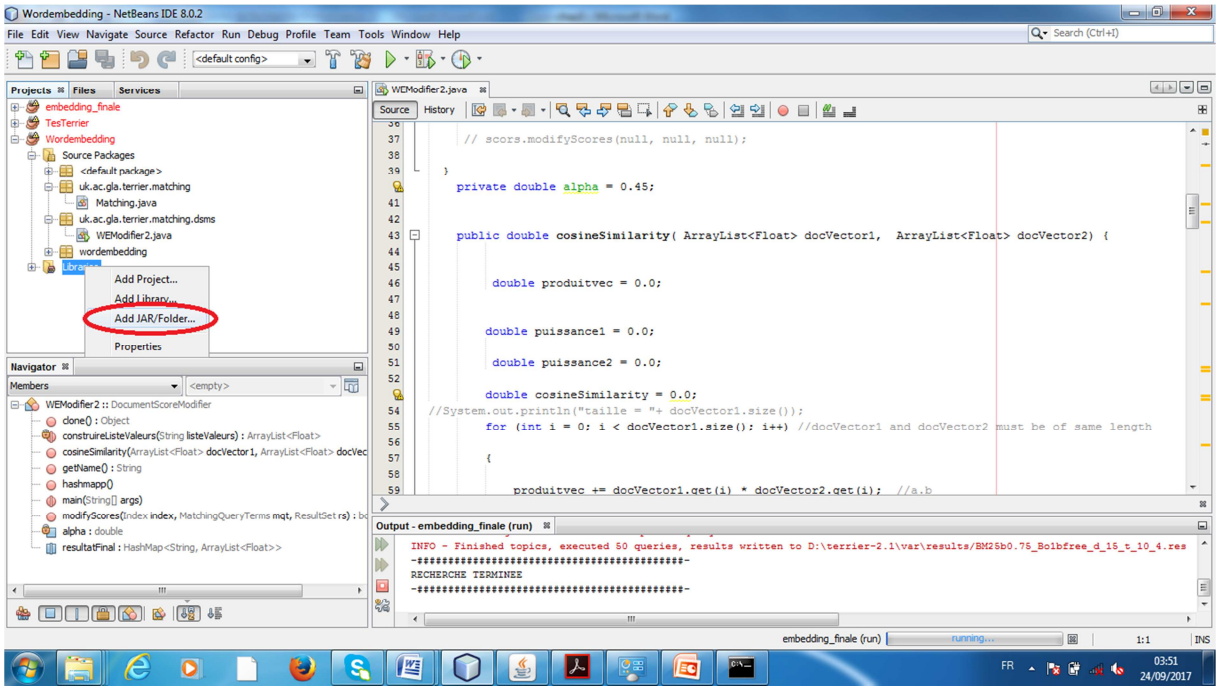


Figure III-8 : Ajouter les bibliothèques de Terrier

III.3 Description de l'approche proposée

Nous décrivons dans ce qui suit le principe de notre approche, un schéma illustrant les différentes étapes suivi pour le réaliser et enfin un algorithme qui l'implémente.

III.3.1 Principe de l'approche

Les modèles de recherche d'information (RI) nous permettent de répondre au besoin de l'information, les documents retournés sont classés par leurs scores vis-à-vis des requêtes avec un ordre décroissant sachant que le document le plus pertinent a un score plus élevé. Néanmoins dans certains cas nous obtenons des résultats qui ne sont pas satisfaisants, c'est-à-dire que le SRI peut retourner des documents qui ne répondent pas à la requête effectuée.

Parmi les raisons invoquées de la non pertinence des résultats l'utilisation de la technique de sac de mots par les modèles de RI. Cette technique ne permet pas de capturer toute la sémantique. La technique de word embedding est utilisée pour remédier aux lacunes de la représentation sac de mot. C'est pour quoi nous avons opté pour l'utilisation de cette technique pour étendre les modèles de RI.

III.3.2 Formalisation de l'approche

Dans notre approche nous utilisons la technique du word embedding pour la sélection des termes des tops documents pertinents retournés. Nous rappelons que le word embedding à pour objectif de représenter les termes sous forme de vecteurs dans un espace d'une dimension réduite dans notre cas (200) dimensions. Nous présentons dans la figure suivante l'emplacement de notre approche dans l'architecture d'un SRI. Nous expliquons ensuite les étapes de notre approche

La figure suivante illustre une architecture que nous avons suivie pour réaliser notre approche :

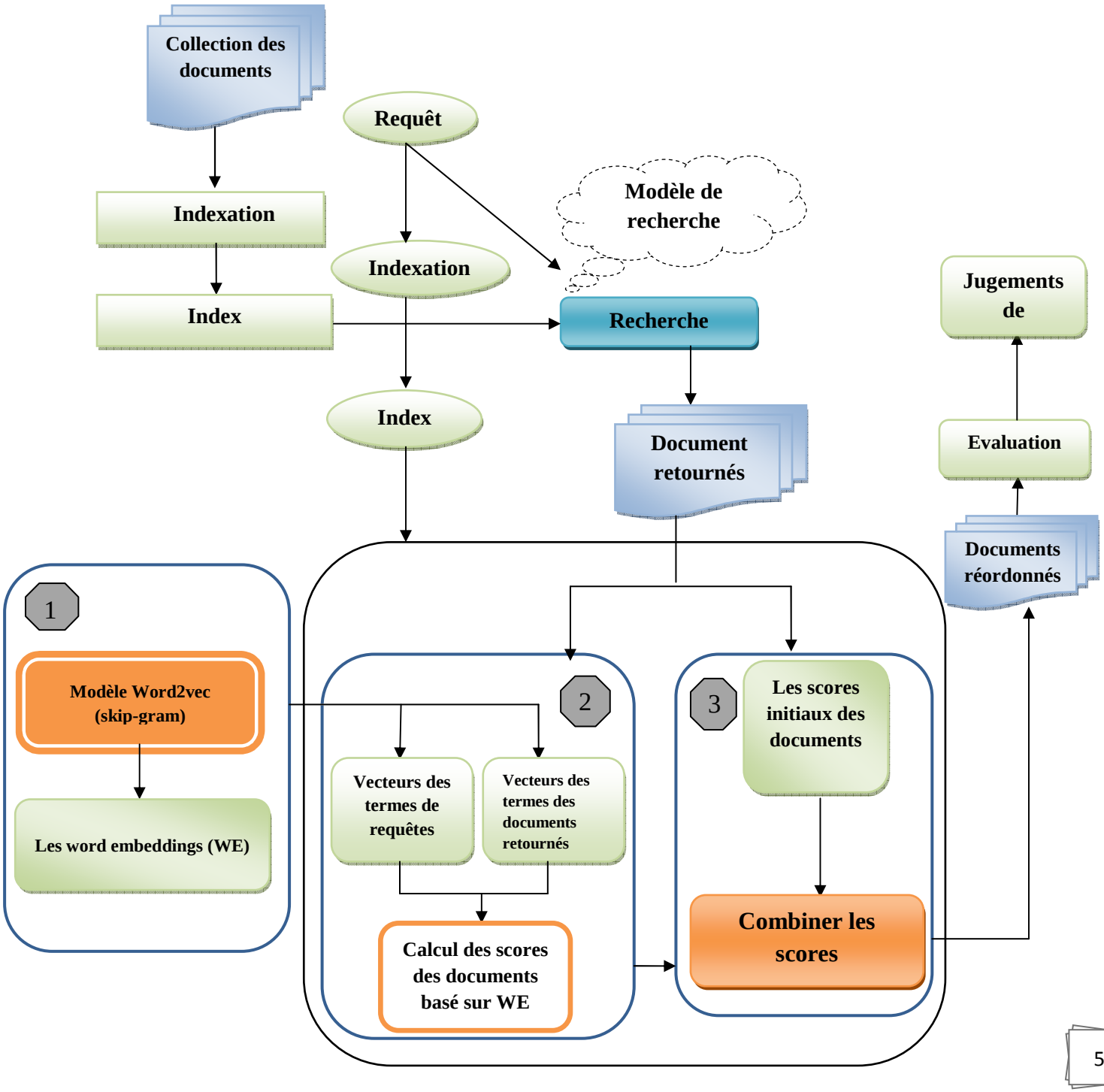


Figure III-9 : Emplacement de notre approche dans un SRI

III.3.3 Les étapes de l'approche

Une fois que la première recherche simple est effectuée nous obtenons les tops documents retournés classés par ordre décroissant par rapport à leurs score, par défaut le SRI retourne les 1000 premiers documents pertinents pour chaque requête. Le modèle de recherche que nous avons utilisé est TF-IDF.

Nous exploitons ainsi ces résultats dans le recalcule du nouveau score qui va nous permettre d'obtenir une nouvelle classification des documents.

Etape 1: Extraction des word embeddings

Nous rappelons qu'il existe plusieurs manières pour construire les word embedding, le modèle le plus connu est word2vec qui constitue de deux architectures skip-gram et cbow.

La figure suivante représente un exemple d'un fichier contenant les word embedding obtenu à partir de l'algorithme de l'architecture skip-gram :

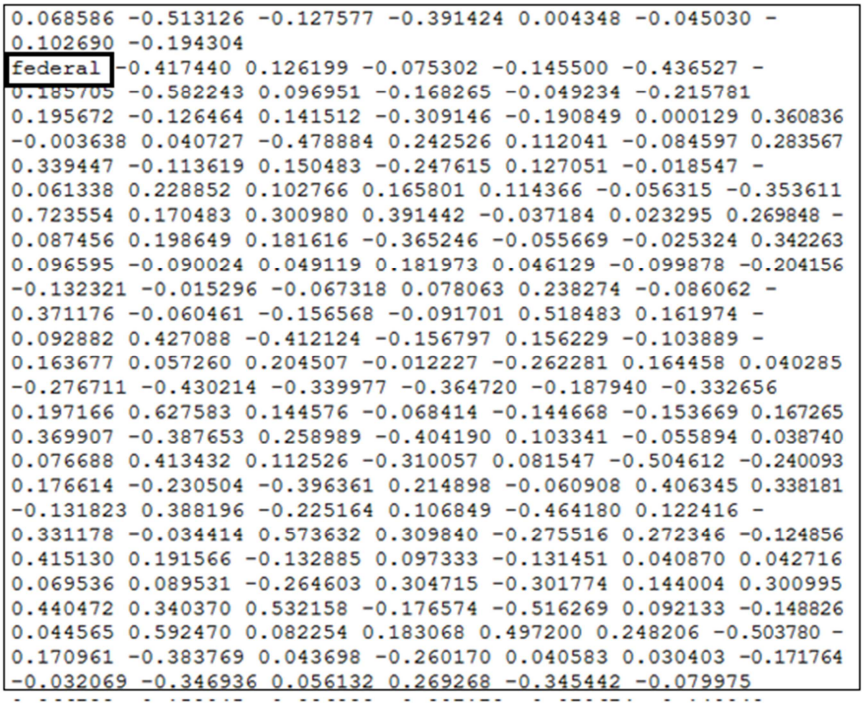


Figure III-10 : exemple d'un fichier contenant les word embedding [5]

Des recherches ont déduit que le modèle qui a donné des meilleurs résultats pour la valeur de la **MAP** est l'architecture « skipgram » en choisissant une taille de la fenêtre petite (window size = w =5) et une dimensionnalité des word embedding = 200 [3], par conséquence nous avons choisis à utiliser ce fichier. Il contient **93951** termes des documents et des requêtes de la collection **APP88**, chaque terme est représenté avec un vecteur d'une telle dimension, dans le cas de ce fichier illustré dans la figure III-2 les vecteurs sont de dimension 200.

Par exemple :

Le mot « federale » est représenté avec ses 200 valeurs (-0.417440, 0.126199,…, -0.019100) dans l'espace vectoriel.

Dans cette étape nous allons extraire les word embeddings des termes des tops documents et les requêtes.

Etape 2 :

Après avoir récupéré ces vecteurs, nous calculons le score de similarité pour chaque terme du document avec tous les termes de la requête.

La formule qui permet de calculer le score entre les word embeddings des documents et les requêtes est donnée comme suit :

$$\boldsymbol{scoreWE(D_i, Q_0)} = \sum_{t_j \in D_i} \sum_{t_i \in Q_0} \boldsymbol{cos}\ (\boldsymbol{V_{t_i}}, \boldsymbol{V_{t_j}}) \qquad \text{(III-1)}$$

Où $\boldsymbol{D_i}$ est le document numéro **i** et $\boldsymbol{Q_0}$ est la requête,

$\boldsymbol{t_j}$ est un terme du document $\boldsymbol{j}$ et $\boldsymbol{t_i}$ le terme de la requête $\boldsymbol{i}$

$\boldsymbol{V_{t_i}}$ est un vecteurs word embedding de terme de la requête $\boldsymbol{t_i}$ et $\boldsymbol{V_{t_j}}$ est un vecteur word embedding des termes du document $\boldsymbol{t_j}$

La similarité cosinus entre deux vecteurs est calculée comme suit :

$$\boldsymbol{cos}\left(\boldsymbol{V_{t_i}}, \boldsymbol{V_{t_j}}\right) = \frac{\boldsymbol{V_{t_i}} * \boldsymbol{V_{t_j}}}{|\boldsymbol{V_{t_i}}| * |\boldsymbol{V_{t_j}}|} \qquad \text{(III-2)}$$

Etape 3 :

Nous allons combiner le score obtenu à l'étape 1 ($\boldsymbol{scoreWE(D_i, Q_0)}$) avec le score initial obtenu lors de la recherche simple effectuée, la formule finale est comme suite :

$$\textbf{scorefinal}(\boldsymbol{D_i, Q_0}) = (\boldsymbol{\alpha}\ \textbf{scoreinitial}(\boldsymbol{D_i, Q_0}) + (\mathbf{1} - \boldsymbol{\alpha})\textbf{score}\boldsymbol{WE}(\boldsymbol{D_i, Q_0})) \quad \text{(III-3)}$$

$\textbf{scoreinitial}(\boldsymbol{D_i, Q_0})$ **:** sont les scores initiaux récupérés de la première recherche simple

$\boldsymbol{scoreWE}(\boldsymbol{D_i, Q_0})$ **:** sont les scores donnés par la formule III-1

Et $\boldsymbol{\alpha}$ est une variable ∈ **[0,1],** elle permet de faire une normalisation entre scoreinitial(D_i, Q_0) et $scoreWE(D_i, Q_0)$ **.**

III.3.4 L'implémentation de l'approche

Cette étapes consiste à décrire un algorithme d'implémentions de l'approche proposé qui est illustré ci-dessous :

Entrées : Top_Document, termes_requêtes, termes_tops_documents, scors_initaux, variable α
Sorties : nouveaux scores

Début

// Déclarer une table et la remplir par les word embeddings

Table <Terme, Vecteur-WE>

Initialiser $\boldsymbol{\alpha}$

Récupérer Top_Document, termes_requêtes, termes_tops_documents, et scoreInitial(D_i, Q_0)

Pour (i=0 jusqu'à taille Top_Document)

Récupérer tous les identifiants des termes et leur fréquence pour chaque document i

Déclaration et initialiser la variable $scoreWE(D_i, Q_0)$)

Pour (j=0 jusqu'au nombre termes_tops_documents pour le document i)

Récupérer les termes des documents

Si le terme n'est pas nul

Récupérer son vecteur dans **Table**

Pour (k=0 jusqu'au nombre termes_requêtes)

Récupérer les termes de la requête k

Récupérer son vecteur dans **Table**

Si (les vecteurs des termes des documents et les requêtes différent de nul)

Calculer le score des word embeddings

$$\boldsymbol{scoreWE(D_i, Q_0)} = \sum_{t_j \in D_i} \sum_{t_i \in Q_0} \boldsymbol{cos}\ (\boldsymbol{V_{t_i}}\ ,\ \boldsymbol{V_{t_j}})$$

Fin si

Fin pour

Fin pour

Calculer le score final

$$\textbf{scoreFinal}(\boldsymbol{D_i, Q_0}) = (\boldsymbol{\alpha}\ \textbf{scoreInitial}(\boldsymbol{D_i, Q_0}) + (\boldsymbol{1-\alpha})\textbf{score}\boldsymbol{WE}(\boldsymbol{D_i, Q_0}))$$

Fin pour

Fin

III.4 Résultats d'évaluation

Après avoir fait l'indexation et la recherche simple, nous effectuerons une nouvelle recherche avec notre nouveau modèle de recherche, ensuit nous observons les valeurs de la MAP après avoir lancé l'évaluation sur l'invite de commende à chaque changement de nombre des tops documents restitués et la valeur **,** nous présentons de ce qui suit la collection de test utilisée, les requêtes ainsi que les résultats d'évaluation obtenues.

III.4.1 La collection de test et les requêtes utilisées

L'outil de recherche TERRIER peut travailler avec des collections TREC ou CLEF. La collection que nous avons utilisée pour nos expérimentations est : **TREC AP88** (**A**ssociated ***P***ress newswire, 19**88**).

Pour la recherche nous avons utilisé 50 requêtes issues des topics numérotées « 51-100 » de la collection TREC.

III.4.2 Présentation des résultats obtenus

Pour évaluer les performances de notre approche, nous devons tout d'abord fixer les Résultats de base : recherche simple et recherche avec notre nouveau modèle.

Ces résultats seront comparés par la suite à ceux obtenus après l'application de notre approche en appliquant les paramètres de recherche :

III.4.2.1 Résultats obtenus avec la recherche simple

Recherche simple	
Le modèle de recherche	TF-IDF
MAP	**0.1245**

Tableau III-2 : Résultats obtenus avec la recherche simple (TF-IDF)

III.4.2.2 Résultats obtenus avec notre approche

Nous avons observé les différentes valeurs de la MAP (précision moyenne) à chaque valeur de α et à chaque changement du nombre des tops documents restitués (**20, 30, 50, 80, 100, 150**).

Les résultats d'évaluation avec notre approche sont résumés dans le tableau suivant :

Nombre des tops documents	**Valeur de α**	**Map (précision moyenne)**
	0.01	**0.1265**
	0.05	**0.1265**
	0.1	0.1260
	0.15	0.1259
	0.2	0.1260
	0.25	0.1263
	0.3	0.1263
	0.35	0.1262

20	0.4	0.1260
	0.45	0.1259
	0.5	0.1258
	0.55	0.1257
	0.6	0.1254
	0.65	0.1253
	0.7	0.1252
	0.75	0.1254
	0.8	0.1252
	0.85	0.1252
	0.9	0.1249
	0.95	0.1247
	1	0.1245
30	0.01	0.1249
	0.05	0.1253
	0.1	0.1249
	0.15	0.1248
	0.2	0.1250
	0.25	0.1253
	0.3	0.1253
	0.35	0.1252
	0.4	0.1251
	0.45	0.1251
	0.5	0.1250
	0.55	0.1249
	0.6	0.1249
	0.65	0.1248
	0.7	0.1248
	0.75	0.1250
	0.8	0.1249
	0.85	0.1250
	0.9	0.1248
	0.95	0.1246
	1	0.1245
50	0.01	0.1239
	0.05	0.1245
	0.1	0.1239
	0.15	0.1239
	0.2	0.1241
	0.25	0.1241
	0.3	0.1247
	0.35	0.1249
	0.4	0.1250
	0.45	0.1252
	0.5	0.1251
	0.55	0.1249
	0.6	0.1252

	0.65	0.1253
	07	0.1247
	0.75	0.1249
	0.8	0.1252
	0.85	0.1250
	0.9	01250
	0.95	0.1251
	0.1	0.1247
80	0.01	0.1238
	0.05	0.1239
	0.1	0.1234
	0.15	0.1234
	0.2	0.1236
	0.25	0.1241
	0.3	0.1243
	0.35	0.1243
	0.4	0.1244
	0.45	0.1246
	0.5	0.1246
	0.55	0.1247
	0.60	0.1248
	0.65	0.1248
	0.7	0.1248
	0.75	0.1251
	0.8	0.1250
	0.85	0.1251
	0.9	0.1248
	0.95	0.1247
	1	0.1245
100	0.01	0.1247
	0.05	0.1247
	0.1	0.1247
	0.15	0.1228
	0.2	0.1231
	0.25	0.1236
	0.3	0.1238
	0.35	0.1240
	0.4	0.1240
	0.45	0.1243
	0.5	0.1244

	0.55	0.1240
	0.6	0.1246
	0.65	0.1247
	0.7	0.1247
	0.75	0.1250
	0.8	0.1240
	0.85	0.1250
	0.9	0.1248
	0.95	0.1248
	1	0.1247
150	0.01	0.1225
	0.05	0.1247
	0.1	0.1223
	0.15	0.1225
	0.2	0.1229
	0.25	0.1234
	0.3	0.1237
	0.35	0.1239
	0.4	0.1240
	0.45	0.1243
	0.5	0.1244
	0.55	0.1245
	0.6	0.1246
	0.65	0.1247
	0.7	0.1247
	0.75	0.1247
	0.8	0.1250
	0.85	0.1251
	0.9	0.1248
	0.95	0.1247
	1	0.1245

Tableau III-3 : Résultats de l'évaluation avec notre approche

Les résultats obtenus dans le tableau montrent qu'il y'a une amélioration avec notre approche de la valeur de la MAP au niveau des requêtes 52, 53, 54, 55, 56, 57, 58, 62, 81, 84, 85, 90, 92, 99 et 100

La meilleur MAP est obtenue en prenant 20 tops documents restitués avec la valeur de $\boldsymbol{\alpha}$ est égale à **0.01** et la même chose pour $\boldsymbol{\alpha}$ **= 0.05.**

Quelques requêtes ont été diminuées avec une faible proportion aux niveaux des requêtes 61, 64, 68, 71, 88 et 97

On remarque aussi que quelques requêtes n'ont pas changé de valeur comme
51 ;59 ;60 :62 :63 ;66 ;69 ;70,72 ;73 ;74 ;75 ;76 ;77 ;78 :80 ;82 ;83,86,87 ;89 ;93 ;94 ;95 ;96 ;, 98

La figure suivante représente un schéma illustrant les résultats d'évaluation par requêtes pour la recherche simple et avec l'utilisation de notre approche :

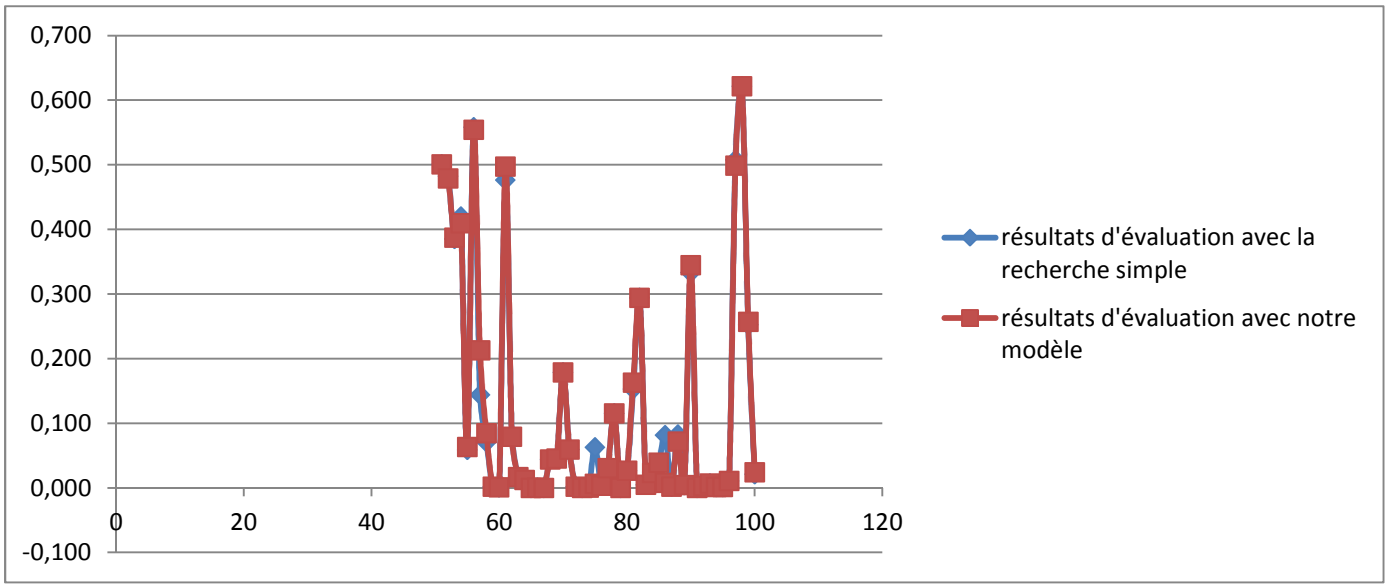


Figure III-10 : Deux graphes illustrant l'évaluation par requêtes pour la recherche simple et avec notre modèle

III.5 Conclusion

Dans ce chapitre, nous avons présenté le principe de notre approche et son fonctionnement, nous avons aussi présenté l'environnement d'implémentation à savoir la plateforme TERRIER, l'environnement Netbeans.

Enfin, nous avons présenté et discuté les résultats de notre approche vis-à-vis le modèle de recherche TF_IDF avec les résultats obtenus avec notre approche.

Une amélioration est constatée avec notre approche qui est basé sur la technique du word embedding sur les requêtes.

Conclusion générale

Conclusion générale

Le travail présenté dans le cadre de ce mémoire se comporte sur la recherche d'information. Notre contribution se porte sur l'extension du modèle de recherche d'information en utilisant la technique du word embedding.

Pour mener à terme notre travail, nous avons donné un aperçu général sur la recherche d'information ainsi le rôle des systèmes de recherche d'information, nous avons présenté par la suit la technique du word embedding et ses quelques modèles de base ainsi que son utilisation dans la recherche d'information De même nous avons défini et suivi, la plate-forme de recherche d'information Terrier, le langage de programmation 'Java' et l'environnement 'Netbeans' qu'on utilisé pour mettre en œuvre notre approche.

L'approche proposée nous a montré qu'il existe toujours d'autres manières pour améliorer les résultats de la recherche d'information.

L'utilisation du word embedding dans notre approche a porté une amélioration par rapport aux résultats de la recherche simple.

Ce travail nous a permis :

D'améliorer nos connaissances sur la recherche d'information, de mettre l'accent sur la manière dont les systèmes de recherche d'information fonctionnent dans la plate-forme Terrier.

Approfondir nos connaissances sur les réseaux de neurones.

Découvrir une nouvelle technique qui est le word embedding.

Perspectives

Malgré les résultats améliorés obtenues avec notre approche, nous souhaiterons toujours aller vers le meilleur, non pas seulement extension du modèle de recherche d'information mais d'en réaliser un plus performant.

Références bibliographiques

[1] : Cours de RI. Master2 ingénierie des systèmes d'informations. Université mouloud mamerie tizi-ouzou.2016.2017

[2] : M Hammache Arezki. Recherche d’information : un modèle de langue combinant mots simples et mots composés. Université Mouloud Mammeri de TIzi-ouzou.

[3] : Mr.Abdelkrim Bouramoul. Recherche d’informationcon textuelle et semantique sur l web. Informatique. Université MENTOURI de Constantine, 2011.

[4] : Abbassi Meftah. Un modèle de reformulation des requêtes pour la recherche d’information sur le Web

[5] : Mr.Ounnaci Iddir. Recherche D'information Dans Les Documents Pédagogiques Sructurés Adaptée Aux Besoins Spécifiques Des Apprenants. Ingénierie Des Systèmes Informatiques. Université Mouloud Mammeri De Tizi-Ouzou

[6] : Fatiha Boubkeur-Amirouche. Contribution à la définition de modèles de recherche d'information flexibles basés sur les CP-Nets. Thèse de doctotorat en informatique. L’Université Toulouse III - Paul Sabatier. 2008

[7] : Jérôme Bondu. “Panorama d’outils de recherche d’informations gratuits et en ligne”. Inter-Ligere Sarl. http://www.inter-ligere.com/article-30587376.html.2009

[8] : Ponte, J.M, Croft, W. B. A language modeling approach to information retrieval. Processing of the 21st annual international ACM SIGIR conference on Research and development in information retrieval,pp. 275-281, 1998

[9] : Zhai, C., Lafferty, J. A study of smoothing methods for language models applied to add hoc information retrieval. In W.B Croft, D.J. Harper, D.H. Kraft, & J. Zobel (Eds), Processing of the 24th annual information ACM-SIGHIR Conference on Researche and Developement in Information Retrieval, pp. 334-342, 2001

[10] : https://fr.wikipedia.org/wiki/Word_embedding

[11] : https://youtu.be/Eku_pbZ3-Mw

[12] : Ye Zhang∗, Md Mustafizur Rahman, Alex Braylan, Brandon Dang, Heng-Lu Chang, Henna Kim, Quinten McNamara, Aaron Angert, Edward Banner, Vivek Khetan, Tyler McDonnell, An Thanh Nguyen, Dan Xu, Byron C. Wallace, Matthew Lease†. Neural

Références bibliographiques

Information Retrieval: A Literature Review .∗Department of Computer Science, University of Texas at Austin, yezhang@utexas.edu †School of Information, University of Texas at Austin. ml@utexas.ed. 27 novembre 2016

[13] : http://ruder.io/word-embeddings-1/

[14] : https://en.wikipedia.org/wiki/Types_of_artificial_neural_networks

[15] : Erhan, D., Bengio, Y., Courville, A., Manzagol, P.-A., Vincent, P., and Bengio, S. (2010). Why does unsupervised pre-training help deep learning? Journal of Machine Learning Research, 11(Feb):625–660.

[16] : Hinton, G. E. and Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. Science, 313(5786):504–507.

[17] : Integrating and Evaluating Neural Word Embeddings in Information Retrieval
Guido Zuccon1 Bevan Koopman; Peter Bruza Leif Azzopardi. Queensland University of Technology, Brisbane, Australia. Australian e-Health Research Centre, CSIRO, Brisbane, Australia. University of Glasgow, Glasgow, United Kingdom
g.zuccon@qut.edu.au,bevan.koopman@csiro.au.p.bruza@qut.edu.au.leif.azzopardi@glasgow.ac.uk

[18] : T. Mikolov, K. Chen, G. Corrado, & J. Dean. Efficient estimation of word representations in vector space. In Workshop at ICLR, 2013

[19] : Bengio, Y., Ducharme, R., Vincent, P., & Janvin, C. (2003). A Neural Probabilistic Language Model. The Journal of Machine Learning Research, 3, 1137–1155.

http://doi.org/10.1162/153244303322533223

[20] :Clinchant, S. and Perronnin, F. (2013). Aggregating continuous word embeddings for information retrieval. In Proceedings of the Workshop on Continuous Vector Space Models and their Compositionality, pages 100–109.

[21] : Bromley, J., Bentz, J. W., Bottou, L., Guyon, I., LeCun, Y., Moore, C., Sackinger, E., and Shah, R. ¨(1993). Signature verification using a siamese time delay neural network. International Journal ofPattern Recognition and Artificial Intelligence, 7(04):669–688.

Références bibliographiques

[22] : Collobert, R., & Weston, J. (2008). A unified architecture for natural language processing. Proceedings of the 25th International Conference on Machine Learning - ICML '08, 20(1), 160–167.

[23] : Y. Goldberg & O. Levy. word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method. arXiv preprint arXiv:1402.3722, 2014.

[24] : T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, & J. Dean. Distributed representations of words & phrases & their compositionality. In NIPS, pg. 3111–3119, 2013.

[25] : http://ruder.io/secret-word2vec/

[26] : Pennington, J., Socher, R., & Manning, C. D. (2014). Glove: Global Vectors for Word Representation. Proceedings of the 2014 Conférence on Empirical Methods in Natural Language Processing, 1532–1543. http://doi.org/10.3115/v1/D14-1162

[27] : S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman. Indexing by latent semantic analysis. Journal of the American Society for Information Science, 41(6):391–407, 1990

[28] : Blei, D. M., Ng, A. Y., and Jordan, M. I. Latent dirichlet allocation. Journal of Machine Learning Research, 3: 993–1022, 2003.

[29] : Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., & Kuksa, P. (2011). Natural Language Processing (almost) from Scratch. Journal of Machine Learning Research, 12 (Aug), 2493–2537. Retrieved from http://arxiv.org/abs/1103.0398

[30] : http://www.touzet.org/Claude/Web-Fac Claude/Les_reseaux_de_neurones_artificiels.pdf

[31]: http://www.trop.uha.fr/pdf/cours-wira.pdf

[32]: Cours réseaux de neurones. Master2 réseaux mobilité et système embarqués. Université mouloud Mammeri t.o. 2016/2017

[33]: LeCun, Y. and Bengio, Y. (1995). Convolutional networks for images, speech, and time series. The handbook of brain theory and neural networks, 3361(10):1995.

[34] : introduction à l'pp artif,power.p

[35]: http://www.terrier.org/

Références bibliographiques

[36]: PDF: OunisI., Amati G., PlachourasV., He B., Macdonald C., Lioma C., « Terrier : A High Per-formance and Scalable Information Retrieval Platform »,Proceedings of ACM SIGIR'06 Workshop on Open Source Information Retrieval(OSIR 2006)

[37]: adcs2015_neural_translation_lm. Integrating and Evaluating Neural Word Embeddings in Information Retrieval. Guido Zuccon1 Bevan Koopman, Peter Bruza1 Leif Azzopardi Queensland University of Technology, Brisbane, Australia. Australian e-Health Research Centre, CSIRO, Brisbane, Australia. University of Glasgow, Glasgow, United Kingdom. https://github.com/ielab/adcs2015-NTLM

[38]: Tutorial - †AM3. Large-scale Information Retrieval Experimentation with Terrier. Rodrygo Santos, Richard McCreadie, Vassilis Plachouras

[39]: https://github.com/ielab/adcs2015-NTLM