REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Mohamed Khider – BISKRA

Faculté des Sciences Exactes, des Sciences de la Nature et de la Vie

Département d'informatique

N° d'ordre : SIOD /M2/2020

Mémoire

Présenté pour obtenir le diplôme de master académique en

Informatique

Parcours : **Système d'information optimisation et de décision (SIOD)**

Accélération du processus d'intégration de termes dans le deep learning

---

Par :

ATIA HADJER

Soutenu le septembre 2020, devant le jury composé de :

		Président
Meadi Mohamed Nadjib	MCB	*Encadreur*
		Examinateur

Dédicace

Je dédie ce modeste travail aux êtres qui me sont les plus chers :

A mes parents, pour l'éducation qui m'ont prodigué et leurs sacrifices qui m'ont permis de tenir gouvernail et qui ont tant souffert pour faire de moi ce que je suis actuellement.

A mes sœurs.

A tous les membres de ma grande famille ATIA et HIMA.

A mes amis (es).

A tous mes collègues de la promotion «2020 ».

Remerciement

Je remercie Dieu le tous puissant de m'avoir donné toute la force nécessaire pour accomplir ce modeste travail. Je tiens particulièrement à remercier mon encadreur Meadi Mohamed Nadjib, pour ces orientations et ces conseils bénéfiques.

Je tiens à exprimer chaleureusement mes sincères remerciements aux personnes qui m'ont fourni un grand soutien pour que ce travail soit achevé.

Nos grand remerciement aussi vont à l'équipe du département d'informatique, ainsi que le Mr le président et les membres de jury qui ont pris la peine d'évaluer notre travail.

Et à tout ceux que je n'avais pas cité, je dit encore :

Merci de mon cœur.

Résumé

Le traitement du langue naturelle, ou TLN connaît depuis récemment un fort regain d'intérêt dans la communauté scientifique. Cet engouement est directement lié aux progrès réalisés depuis quelques années dans le domaine de l'intelligence artificielle (machine learning et deep learning plus particulièrement). Cela nous porte à croire que c'est un problème de la reconnaissance du langage humain par des machines est désormais quasiment résolu.

Ce travail propose des solutions pour éliminer l'un des problèmes du modèle Bert . C'est l'incapacité à gérer une longue séquence de texte. diviser le texte que ce soit en le divisant en parties ou en utilisant son résumé, ou en en prenant des parties, que ce soit au recto, au verso ou au milieu, ou en combinant les parties précédentes.

Conçu et implémenté sur une base de données BBC NEWS.

Mots clés : TNL, plongement de mot, Bert.

ملخص

في الآونة الأخيرة ، شهدت معالجة اللغة الطبيعية ، أو TLN ، تجددًا قويًا للاهتمام في المجتمع العلمي. يرتبط هذا الحماس ارتباطًا مباشرًا بالتقدم المحرز في السنوات القليلة الماضية في مجال الذكاء الاصطناعي (التعلم الآلي والتعلم العميق على وجه الخصوص). هذا يقودنا إلى الاعتقاد بأن مشكلة التعرف على لغة الإنسان من قبل الآلات قد تم حلها الآن تقريبًا.

يقترح هذا العمل حلولاً للتخلص من إحدى مشكلات نموذج BERT. إنه عدم القدرة على التعامل مع سلسلة طويلة من النصوص. يقسم النص إما بتقسيمه إلى أجزاء أو باستخدام ملخصه ، أو بأخذ أجزاء سواء في المقدمة أو الخلف أو في المنتصف ، أو بجمع الأجزاء السابقة.

تم تصميمه وتنفيذه على قاعدة بيانات BBC NEWS.

الكلمات المفتاحية: البرمجة اللغوية العصبية، تضمين الكلمات، بيرت.

ABSTRACT

Recently, Natural Language Processing, or NLP, has seen a strong resurgence of interest in the scientific community. This enthusiasm is directly linked to the progress made in the last few years in the field of artificial intelligence (machine learning and deep learning in particular). This leads us to believe that it is a problem problem of the recognition of human language by machines is now almost solved.

This work proposes solutions to eliminate one of the problems of the Bert model. It is the inability to handle a long sequence of text. divide the text either by dividing it into parts or using its summary, or by taking parts, whether on the front, back or the middle, or by combining the preceding parts.

Designed and implemented on a BBC NEWS database.

Keywords: NLP, Word Embeddings, Bert.

Table des matières

Liste des figures

Liste des Tableaux

ACRONYMES

WE	Word Embedding.
TLN	Traitement du Langue Naturelle.
CBOW	Sac-de-mots continus (Bag of words).
SG	Skip Gram.
R&D	Research and Development.
AYN	Attention is All You Need.
IDF	Inverse Document Frequency.
TF	Term Frequency.
GloVe	Global Vectors for Word Representation.
ELMo	Embeddings from Language Model.
BERT	Bidirectional Encoder Representations from Transformers.
RNN	Recurrent neural network.
LM	Language Model

Introduction générale

L'évolution énorme des réseaux de neurones ont conduit à des développements de plusieurs domaines tels que la vision par ordinateur, la reconnaissance vocale et le traitement du la langue naturelle (TLN) ... etc. L'un des développements récents les plus influents du TLN est l'utilisation de Word Embedding, pour représentés les mots par des vecteurs dans un espace continu, capturant de nombreuses relations syntaxiques et sémantiques entre eux. Au cours des dernières années, les avantages de l'utilisation de représentations de mots distribuées (Embeddings) ont été illustrés et en évidence dans de nombreuses tâches TNL différentes, comme par exemple l'analyse des sentiments, la reconnaissance d'entités nommées et une partie du balisage vocal.

La recherche sur le traitement du langue naturelle a fait d'énormes progrès après avoir été relativement stationnaire pendant quelques années. Les représentations d'encodeur bidirectionnel de Google de Transformer (BERT) devenant le point culminant d'ici la fin de 2018 pour atteindre des performances de pointe dans de nombreuses tâches TLN. BERT est le surnom donné par Google à un changement majeur dans la manière dont fonctionne son moteur de recherche. Avec cette mise à jour, Google souhaite répondre plus efficacement aux requêtes des internautes, en tenant compte des mots-clés utilisés et surtout de la manière dont ils sont agencés les uns aux autres.

Fondamentalement, BERT permettre de comprendre le « sens » d'une requête et, donc, la signification des mots qui sont utilisés dans un contexte précis. En somme, il ne s'agit plus de prendre les termes isolément, mais de les comprendre selon leur voisinage. Or, cela nécessite de tenir compte de certains éléments d'une requête qui étaient jusqu'à présent délaissés, parce que jugés secondaires.

Malgré tout cela, BERT a des limites, et l''une de ces limites est le manque de capacité à gérer une longue séquence de texte. Par défaut, BERT prend en charge jusqu'à 512 tokens. Ce travail a le but à proposer des solutions à ce problème.

Outre une introduction générale et une conclusion, notre mémoire de master est organisé en 3 chapitres:

Dans le premier chapitre, nous présentons quelques notions fondamentales liées aux Word embedding, en donnant sa définition et ses différents types, en nous concentrant sur le modèle BERT, en mentionnant certains de ses avantages et inconvénients, et des propositions pour remédier à certains des inconvénients.

Le second chapitre est réservé a` la conception de notre systéme, son architecture globale et détaillée. Le dernier chapitre est consacré a` l'implémentation et la réalisation de la proposition de notre système, et aussi les expérimentations et la discussion des résultats obtenus. Le mémoire est terminé par une conclusion générale contenant les perspectives envisagées.

Ce mémoire se terminera par une conclusion qui présente un bilan et propose certaines perspectives.

Chapitre

WORD EMBEDDING

1. Introduction

La TNL est l'application de diverses méthodes pour calculer des choses utiles à partir d'un texte. Cela commence par convertir le texte en question en quelque chose sur lequel nous pouvons utiliser un ordinateur pour faire des calculs et prendre des décisions. certaines données s'y prêtent directement, comme par exemple lorsqu'on traite les images nous utilisons des vecteurs de coefficients associées aux intensités de pixel, et les coefficients de densité spectrales lorsqu'on traite spectre audio.

Parmi les techniques récemment apparues, il y a la représentation de mots individuels comme vecteurs de valeur réelle dans un espace vectoriel prédéfini. Ces représentations sont appelées word embeddings, neural embeddings, ou en français représentation vectorielle du mot, représentation continue du mot et aussi plongement de mot.

Avant le word embedding, la représentation en sac de mots (simples) était la plus couramment utilisée dans les applications traitant des textes. Cependant, la représentation en sac de mots ne capture pas les relations entre les mots. word embedding est une solution qui permet de pallier ce problème. word embedding repose sur la théorie linguistique fondée par Zelling Harris et connue sous le nom de Distributional Semantics. Cette théorie considère qu'un mot est caractérisé par son contexte, c'est à dire par les mots qui l'entourent. Ainsi, des mots qui partagent des contextes similaires partagent également des significations similaires. Les algorithmes de word embedding sont le plus souvent

employés pour décrire des mots à travers de vecteurs numériques, mais ils peuvent également être utilisés pour construire des représentations vectorielles de phrases entières, de données biologiques comme les séquence d'ADN, ou encore des réseaux représentés comme des graphes **[1]**.

Dans ce chapitre nous allons présenter les concepts de base ainsi que les différentes approches de word embedding.

2. Définition de Word Embedding

Le word embedding (WE) désigne un ensemble de techniques qui visent à représenter les mots ou les phrases d'un texte par des vecteurs de nombres réels, décrits dans un modèle vectoriel (En anglais: Vector Space Model). Ces nouvelles représentations de données textuelles ont permis d'améliorer les performances des méthodes de traitement automatique des langues naturelles «TNL» (ou Natural Language Processing «NLP»), comme le Topic Modeling ou le Sentiment Analysis **[2]**.

Les WEs constituent une projection des mots du vocabulaire dans un espace de faible dimension de manière à préserver les similarités sémantiques et syntaxiques. Ainsi, si les vecteurs de mots sont proches les uns des autres en termes de distance, les mots doivent être sémantiquement ou syntaxiquement proches. Chaque dimension représente une caractéristique latente du mot, qui peut capter des propriétés syntaxiques et sémantiques **[1]**.

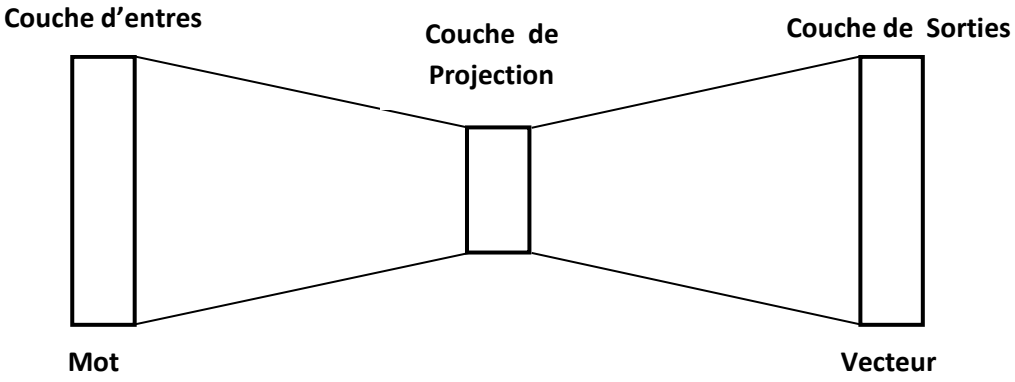

FIGURE 1: Architecture WE.

3. Les Types de WE

Les méthodes de WE peuvent être catégorisées en deux types: basées sur la fréquence et basées sur les réseaux de neurones. Ces deux types diffèrent lors de la construction des vecteurs de mots ainsi que par le contexte qu'elles prennent en compte. Les méthodes basées sur le comptage utilisent les documents comme contexte et capturent la similarité sémantique entre documents alors que celles basées sur les réseaux de neurones utilisent les mots voisins comme cotexte pour détecter la similarité mot à mot. Les approches basées sur le comptage tendent à être utilisées pour la modélisation de sujets car elles capturent très bien la relation sémantique, alors que les approches basées sur les réseaux de neurones sont plus efficaces pour obtenir la similarité entre les mots **[1]**.

3.1 WE basées sur le comptage

3.1.1 Sac de Mots (Bag of Words)

Le principe du sac de mots est plutôt simple. On peut dire qu'il ressemble même à celui de l'encodage one-hot **[3]**. Son principe se résume en 3 phases :

- La décomposition des mots. On appelle cela aussi la tokenisation: est une étape simple mais indispensable. Le principe est simple il faut découper la ou les phrases en phonèmes ou mots.
- Ce dictionnaire (vocabulaire La constitution d'un dictionnaire global qui sera le vocabulaire: Il est évident qu'on ne va pas traiter une seule phrase ! on doit en traiter un grand nombre, et donc on construire une sorte de dictionnaire (ou vocabulaire) qui va consolider tous les mots que l'on a tokenisé.) nous permettra par la suite de réaliser l'encodage nécessaire pour « numériser » nos phrases. pour résumer, cette étape nous permet de créer notre sac de mots. Nous aurons donc à la fin de cette étape tous les mots (uniques) qui constituent les phrases de notre jeu de données.
- L'encodage: c'est l'étape qui va transformer les mots en nombre. l'idée est simple, pour chaque nouvelle phrase, il faut la tokeniser (même méthode que pour la constitution du vocabulaire) . Si le mot de la phrase existe dans le vocabulaire, il suffit alors de mettre un 1 l'emplacement du mot dans le vocabulaire **[3]**.

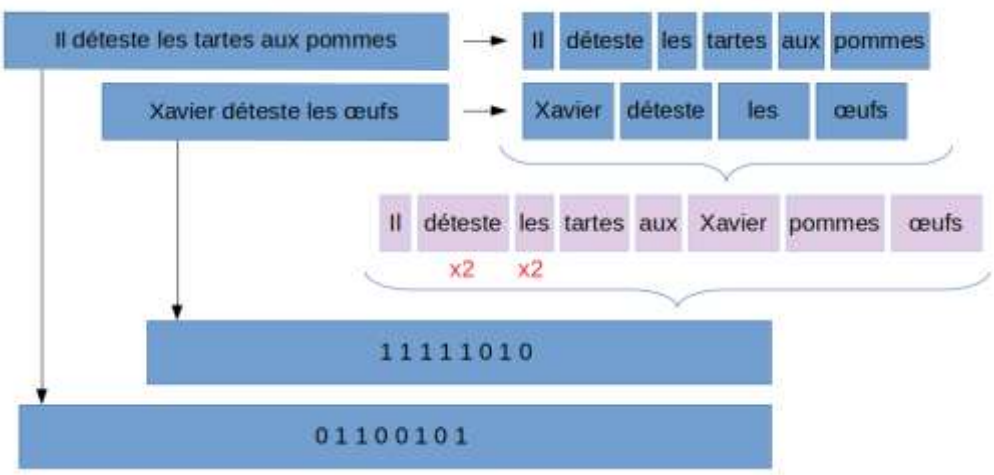

FIGURE 2 : Les étapes de sac de mot.[**3]**

3.1.2 TF-IDF

Une des mesures les plus utilisées pour estimer l'importance des mots est la pondération IDF (Inverse Document Frequency) . Elle opère une transformation des effectifs bruts des mots qui sont calculés dans le cadre du traitement de texte, et permet d'exprimer simultanément les fréquences auxquelles certains termes ou mots spécifiques apparaissent dans un ensemble de documents, ainsi que leurs spécificités sémantiques. Le principe de l'approche est de déterminer les mots les plus pertinents dans un texte et ceux qui sont insignifiants comme les stopwords. Ainsi les mots les moins fréquents sont les plus discriminants.

Cette mesure est généralement utilisée avec le calcul des TFs (Term Frequency) qui consiste à calculer la fréquence d'un mot par rapport à un document. Plusieurs variations de TF existent comme nous pouvons le voir dans tableau 01.

Schéma de pondération	**Formule de TF**
Binaire	**0, 1**
Fréquence brute ($f_{t,d}$)	$\begin{cases} \frac{n_{ij}}{\sum_{k=1}^{k=m} n_{kj}} & si\ n_{ij} > 0 \\ 0 & si\ n_{ij} < 0 \end{cases}$
	$\begin{cases} \frac{n_{ij}}{\max n_{kj}} & si\ nij > 0 \\ 0 & si\ n_{ij} < 0 \end{cases}$
Normalisation logarithmique	$\begin{cases} \frac{n_{ij}}{1 + \log(1 + logn_{ij})} & si\ n_{ij} > 0 \\ 0 & si\ n_{ij} < 0 \end{cases}$

Normatlisation « 0.5 » par le max	$\mathbf{0.5 + 0.5}\dfrac{f_{iq}}{\max\{f_{1q}, f_{2q}, \ldots\ldots, f_{v_q}\}}$
Normalisation par le max	$\boldsymbol{k + (k + 1)}\dfrac{f_{iq}}{\max\{f_{1q}, f_{2q}, \ldots\ldots, f_{v_q}\}}$

Tableau 01 : Variantes de TF.

Où n_{ij} : le nombre d'occurrences de terme$_i$ dans le document$_j$.

IDF consiste à réduire les coordonnées de certains axes, correspondant à des termes qui existent dans beaucoup documents.

$$IDF = log\ (N / n)$$

Où, N est le nombre total de documents et n est le nombre de documents dans lesquels un terme t est apparu.

Le TF-IDF est une mesure largement utilisée, elle permet d'avoir une estimation globale sur l'importance d'un terme contenu dans un document, relativement à une collection ou un corpus, elle est obtenue en multipliant TF par IDF **[4]**:

$$\textbf{TF-IDF (t) = IDF (t) * TF (t)}$$

3.1.3 Matrice de Co-occurrence

La matrice de Co-occurrence des mots décrit comment les mots se produisent ensemble qui, à leur tour, capture les relations entre les mots.La matrice de cooccurrence de mots est calculée simplement en comptant comment deux ou plusieurs mots se produisent ensemble dans un corpus donné. À titre d'exemple de Co-occurrence de mots, considérons un corpus composé des documents suivants:

penny wise and pound foolish

a penny saved is a penny earned

Soit count (w (next) | w (current)) représenter le nombre de fois le mot w (next) suit le mot w (current) , nous pouvons résumer les statistiques de co-occurrence pour les mots "a" et "penny" comme suit:

	a	and	earned	foolish	is	penny	pound	saved	wise
a	0	0	0	0	0	2	0	0	0
penny	0	0	1	0	0	0	0	1	1

Tableau 02 : Statistiques co-occurrence pour les mots "a" et "penny".**[5]**

Le tableau ci-dessus montre que «a» est suivi deux fois par «penny» tandis que les mots «*earned*», «sauved» et «wise» suivent chacun «penny» une fois dans notre corpus. Ainsi, «earned» est une fois sur trois susceptible d'apparaître après «penny». Le nombre indiqué ci-dessus est appelé fréquence bigramme , il ne regarde que le mot suivant d'un mot courant. Étant donné un corpus de N mots, nous avons besoin d'un tableau de taille *N*x*N* pour représenter les fréquences bigrammes de toutes les paires de mots possibles. Une telle table est très clairsemée car la plupart des fréquences sont égales à zéro. En pratique, les nombres de cooccurrences sont convertis en probabilités. Il en résulte des entrées de ligne pour chaque ligne totalisant une dans la matrice de cooccurrence.

Mais n'oubliez pas que cette matrice de cooccurrence n'est pas la représentation vectorielle de mots qui est généralement utilisée. Au lieu de cela, cette matrice de cooccurrence est décomposée en utilisant des techniques comme PCA, SVD, etc. en facteurs et la combinaison de ces facteurs forme la représentation vectorielle de mots **[5]**.

3.2 Les Modèles de WE

Pour entraîner les WEs, il ’existe plusieurs techniques qui sont composées de deux parties, la première, qui est principalement des pointeurs, fait simplement référence aux techniques classiques des WEs, qui peuvent également être considérées comme des WEs statiques, car le même mot aura toujours la même représentation quelque soit le contexte où se trouve. Nous trouvons par exemple:

- *Word2Vec.* **[28]**
- *GloVe.* **[31]**
- *fastText*. **[32]**

La deuxième partie présente des nouvelles techniques de WEs qui prennent en considération le contexte du mot et peuvent être considérées comme des techniques de WEs dynamiques, dont la plupart utilisent un modèle de langage pour aider à modéliser la représentation d'un mot. J'essaie de décrire des techniques d’embeddings contextuelle **[6]**:

- *ELMO.* **[33]**
- *BERT.* **[34]**

3.2.1 WEs Classique

3.2.1.1 Word2vec

Word2vec est une méthode fondée sur des réseaux de neurones artificiels et définie dans **[28]**. Cette méthode propose deux architectures de réseaux de neurones : l’architecture en CBOW (En frençais: sac-de-mots continus, ou En anglais: Continuous Bag-of- Words) et l’architecture Skip-gram. Ces deux architectures se présentent sous la forme de réseaux de neurones artificiels simples. Ils sont constitués de trois couches : une couche d’entrée, une couche cachée et une couche de sortie. La couche d’entrée contient soit un "sac-de-mots" (CBOW), soit un mot seul (Skip-gram). La couche cachée correspond à la projection des mots

d'entrée dans la matrice des poids. Cette matrice est partagée par tous les mots (matrice globale). Enfin, la couche de sortie est composée de neurones "softmax". Pour des raisons de complexité algorithmique due à la couche "softmax", les auteurs dans (Mikolov et al., 2013b) ont introduit deux alternatives appelées "échantillons négatifs" et "softmax hiérarchique". Le couplage de ces fonctions avec la simplicité de ces réseaux leur permettent d'être entraînés sur de très grandes quantités de textes, et ainsi d'obtenir des modélisations de meilleure qualité que les modèles plus complexes à base de récurrence ou de convolution par exemple **[44]**. Un exemple de ces deux architectures est présenté dans la Figure 3.

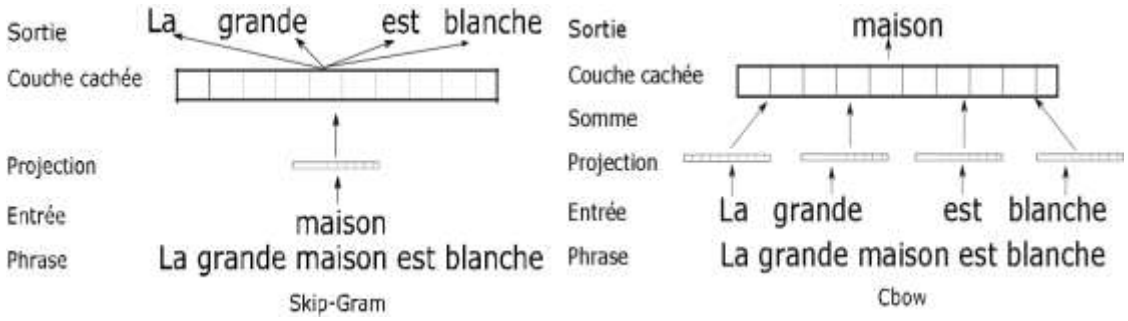

FIGURE 3 : Exemples des architectures CBOW et Skip-gram de Word2vec. **[7]**

Ces modèles sont capables de capturer des régularités sémantiques et syntaxiques **[30]**. En effet, la distance qui sépare la projection de deux mots peut représenter une relation complexe telle que la notion de "singulier-pluriel" ou "masculin-féminin" **[29]** comme le montre la figure 4 **[7]**.

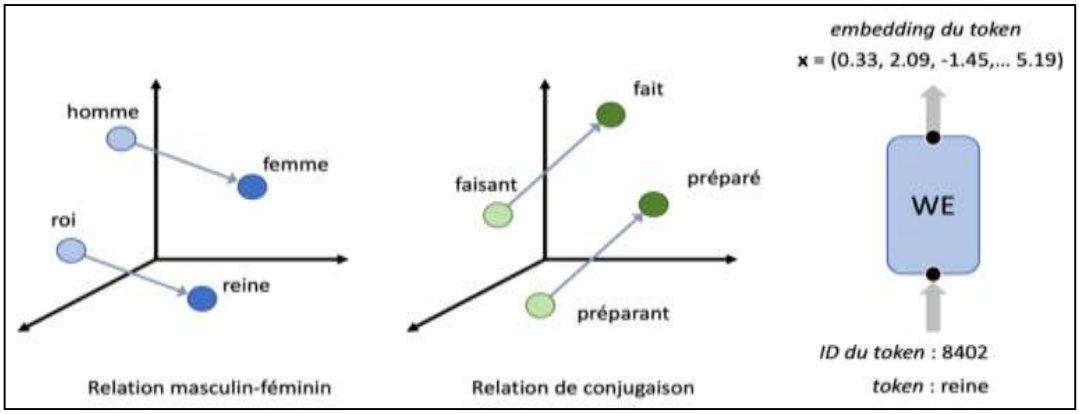

FIGURE 4 : Exemples de relations de mots dans l'espace Word2vec.**[8]**

Word2vec est connu comme un modèle «sans contexte» dans le sens où il produit un vecteur unique pour un mot quel que soit le contexte dans lequel il a été utilisé. En fait, ce modèle ne prend en compte le contexte des mots que l'apprentissage du modèle. Au moment du test on ne recalcule pas les «WEs» mais on fait juste appels a ceux obtenus après apprentissage (Word2vec n'apprend pas du contexte des mots). De plus, étant donné que pendant l'apprentissage Word2vec produit des vecteurs de mots, si dans les données de test il y'a un mot absent du vocabulaire, il n'y aura pas de vecteur numérique pour ce mot. En d'autres termes, word2vec fonctionne mieux pour les mots du vocabulaire et quand ceux-ci sont utilisés dans le même contexte dans le corpus d'apprentissage et dans les données de test **[9]** .

CBOW

L'architecture CBOW est un réseau de neurones doit prédire le poids d'un mot i en fonction de son contexte, qui correspond aux mots précédents et aux mots suivants. Dans cette architecture, la couche de projection est partagée par tous les mots : tous les mots sont projetés dans la même position.

Ce modèle est nommé CBOW pour sac-de-mots continus ou Continuous Bag-of- Words pour deux raisons. D'une part, l'appellation sac-de-mots indique que l'ordre des mots du

contexte n'a pas d'influence sur la projection. D'autre part, l'objectif souligne l'utilisation d'une représentation continue (WEs) des mots en contexte, ce qui diffère des modèles sac-de-mots standards.

Dans ce cas, l'apprentissage des WEs consiste à prédire un mot en fonction de son contexte. Ceci est fait en calculant la somme des WEs du contexte, puis en appliquant sur le vecteur résultant un classifieur log-linéaire pour prédire le mot cible. En fin, le modèle compare sa prédiction avec la réalité et corrige la représentation vectorielle du mot par rétro-propagation du gradient. Ce modèle cherche à maximiser l'équation suivante :

$$S = \frac{1}{I}\sum_{i=1}^{I} log\ p(m_i/m_{i-n}, \dots\dots, m_{i-1}, m_{i+1} \dots\dots, m_{i+n}) \qquad (1)$$

où *I* correspond au nombre de mots dans le corpus, le modèle reçoit une fenêtre de n mots autour du mot cible m_i.

Cette architecture présente plusieurs avantages : en effet, en plus d'être efficiente d'un point-de-vue algorithmique **[29]**, elle permet à la fois une meilleure modélisation des mots fréquents et une meilleure capture des relations syntaxiques **[10]**.

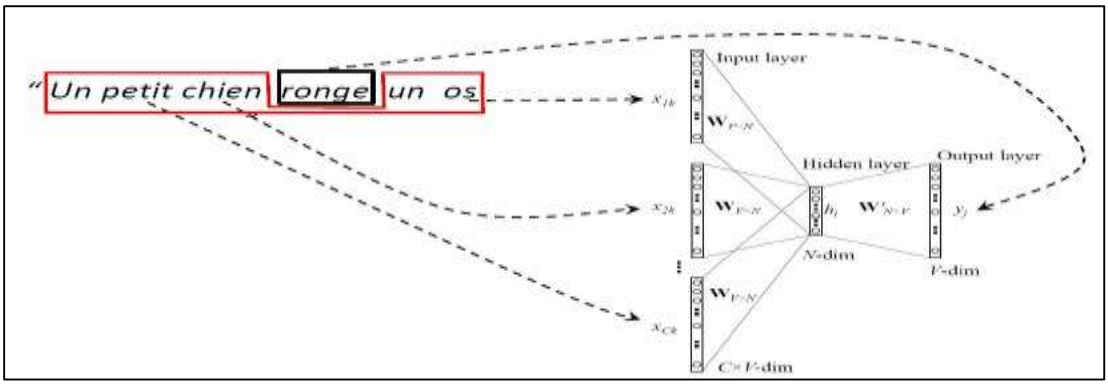

FIGURE 5 : Architecteur de CBOW. **[11]**

Skip-Gram (SG)

L'architecture Skip-Gram tente de prédire, pour un mot donné, le contexte dont il est issu. La couche d'entrée de ce réseau est alors un vecteur ne contenant qu'un seul mot. Le mot

est projeté dans la couche cachée puis dans la couche de sortie. Le contexte est ensuite réduit de façon aléatoire à chaque itération. Le vecteur de sortie est ensuite comparé à chacun des mots du contexte réduit et le réseau se corrige par rétro-propagation du gradient. De cette manière, la représentation du mot d'entrée va se rapprocher de chacun des mots présents dans le contexte.

Le réseau de neurones Skip-gram essaie de maximiser une variation de l'équation 1 comme suit :

$$S = \frac{1}{I}\sum_{i=1}^{I}\sum_{-n\leq j\leq n,\neq 0} logp\,(m_i/m_{i+j}) \qquad (2)$$

Par rapport au CBOW, cette architecture permet une meilleure modélisation des mots peu fréquents et permet de mieux capturer les relations sémantiques **[29] [7]**.

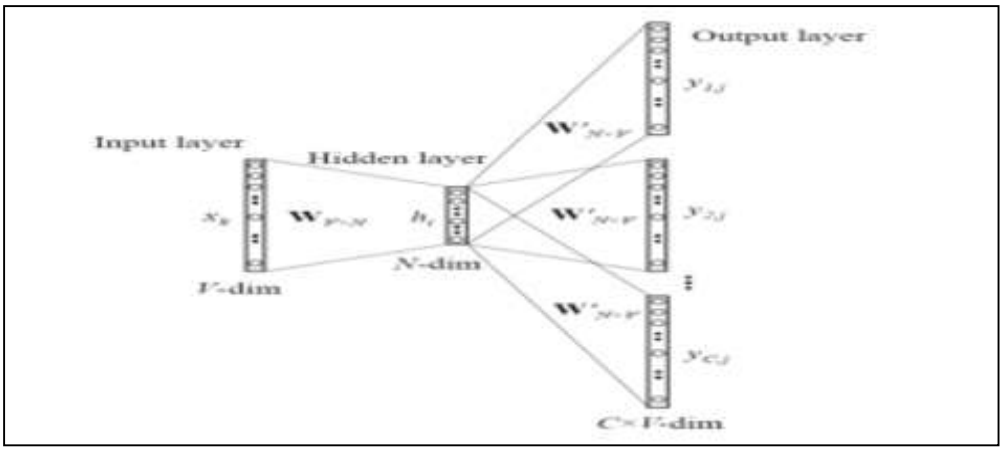

FIGURE 6 : Architecteur de SG. **[12]**

3.2.1.2 Glove

GloVe (**Glo**bal **Ve**ctors for Word Representation) **[31]**, est un modèle d'apprentissage non supervisé proposé par l'équipe TLN de l'université de Stanford qui prend en compte toute l'information portée par le corpus et non pas la seule information portée par une fenêtre de mots.

Cette approche repose sur la construction d'une matrice de co-occurrence globale des mots (MG), en traitant le corpus en utilisant une fenêtre contextuelle glissante. Ici, chaque élément MG_{ij} représente le nombre de fois où le mot m_j apparaît dans le contexte du mot m_i.

Une fois la matrice MG calculée, un modèle de régression par moindres carrés est entrainé pour construire des représentations vectorielles $\vec{m}i$ et $\vec{m}_j$. Ces représentations doivent conserver des informations utiles sur la co-occurrence de paire de mots m_i et m_j , telles que :

$$\vec{m}i \,.\vec{m}_j + b_i + b_j = \log(MG_{ij})$$

Où MG_{ij} représente le nombre de fois le mot j se produit dans le contexte du mot *i*. b_i et b_j sont des biais scalaires associés aux mots *i* et *j* respectivement **[10][13]**.

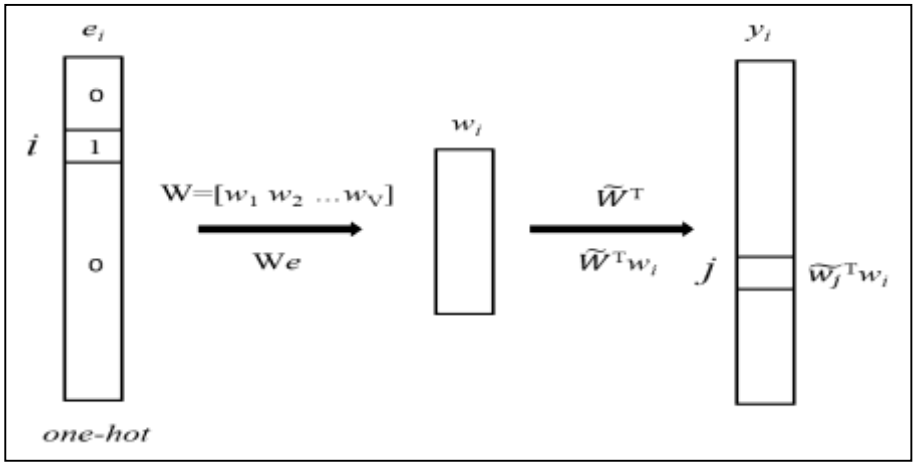

FIGURE 7 : L'architecture du modèle de GloVe. L'entrée est une représentation one-hot d'un mot. Les matrices WEs servent de matrices de poids dans le modèle et ainsi la sortie du modèle est un vecteur de produits internes des vecteurs de mots.**[14]**

3.2.1.3 Fast TEXT

FastText **[32]** de Facebook. est basé sur le modèle skipgram de word2vec **[28]**, qui consiste à apprendre des représentations de mots pour qu'elles optimisent une tâche de prédiction du contexte des mots. Cet outil permet de générer très rapidement et facilement une matrice de représentations latentes pour tous les mots d'un vocabulaire, mais, contrairement à des implémentations comme Word2vec, elle possède une fonctionnalité supplémentaire, FastText construit également des vecteurs pour des parties de mots (n-grammes de caractères, par défaut avec n = 3,…...,6). Cela offre la capacité au modèle entraîné de pouvoir éventuellement gérer des mots hors-vocabulaires (qui n'ont jamais été rencontrés durant l'apprentissage), dont la représentation sera alors calculée à l'aide des représentations apprises pour les parties de mots qui les composent. Ceci permet également de rendre le modèle plus cohérent avec les différentes écritures d'un même mot (par exemple, les pluriels ou les accords en genre, qui, dû à la nature de la construction des modèles de word embeddings, pourraient, à tort, voir leurs représentations être entachées de biais indésirés, dû au langage employé dans un corpus), le rendant ainsi potentiellement plus robuste aux fautes de frappes et d'orthographe qui sont présentes dans notre corpus **[15][16]**.

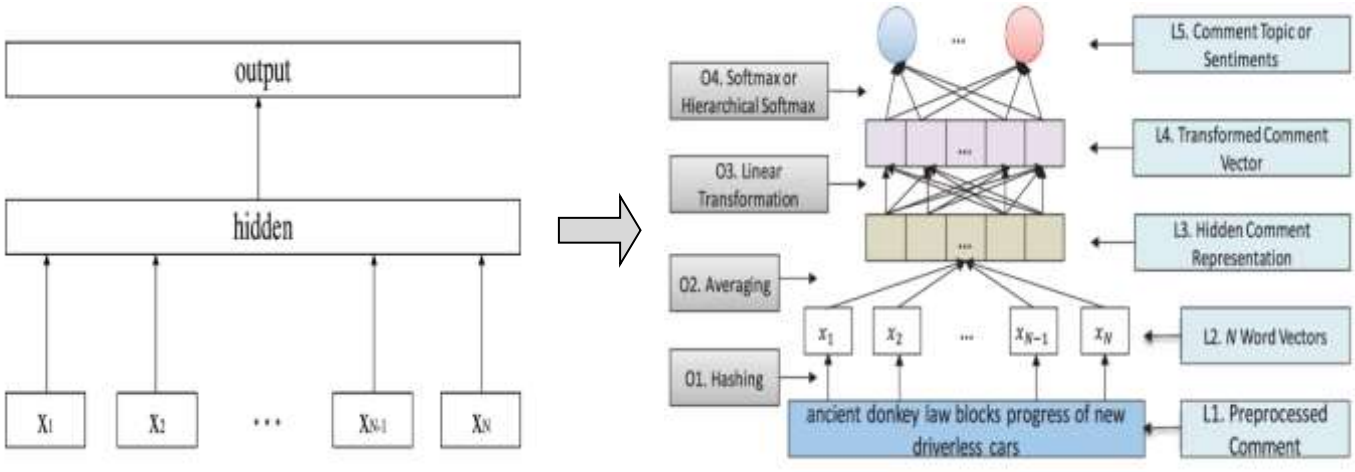

FIGURE 8: Architecture de modèle de fastText, où L (en L1) représente couche et O (en O1) l'opération.**[17][18]**

3.2.2 WE Contextuelle

3.2.2.1 ELMO

Au lieu d'utiliser une WE fixe pour chaque mot, comme le font les modèles comme GloVe, ELMo examine la phrase entière avant de lui attribuer son WE. Il utilise un LSTM bidirectionnel formé sur une tâche spécifique, pour pouvoir créer une WE contextuelle. ELMo a fourni un pas de géant vers une meilleure modélisation et compréhension du langage. L'ELMo LSTM après avoir été entraîné sur un ensemble de données massif, peut ensuite être utilisé en tant que composant dans d'autres modèles TNL destinés à la modélisation de langage.

ELMo signifie (**E**mbeddings from **L**anguage **Mo**del) **[33],** et par conséquent, il a également la capacité de prédire le mot suivant dans une phrase ce qui est essentiellement ce que font les modèles de langage. Lorsqu'il est formé sur un grand ensemble de données, le modèle commence également à détecter les modèles de langage.

Il est peu probable qu'il devine avec précision le mot suivant dans l'exemple. De tels modèles permettent de déterminer que, si vous voyez l'expression comme « *je vais écrire avec un*» , le mot «*crayon*» semble être un mot suivant plus raisonnable que « *grenouille*» .

Elmo utilise le LSTM bidirectionnel dans l'apprentissage, de sorte que son modèle de langage comprend non seulement le mot suivant, mais aussi le mot précédent de la phrase. Il contient une dorsale LSTM bidirectionnelle à 2 couches. La connexion résiduelle est ajoutée entre la première et deuxième couches. Les connexions résiduelles sont utilisées pour permettre aux gradients de traverser un réseau directement, sans passer par les fonctions d'activation non linéaires. L'intuition de haut niveau est que les connexions résiduelles aident les modèles profonds à mieux s'entraîner.

Principales caractéristiques d' ELMO sont :

- Les représentations de mots ELMo sont purement basées sur des caractères, ce qui permet au réseau d'utiliser des indices morphologiques pour former des représentations robustes pour des jetons hors vocabulaire invisibles pendant l'apprentissage.
- Contrairement aux autres WEs, il génère des vecteurs de mots lors de l'exécution.
- Il donne WE de tout ce que vous mettez : des caractères, des mots, des phrases, des paragraphes, mais il est conçu pour les WEs de phrases à l'esprit **[19]**.

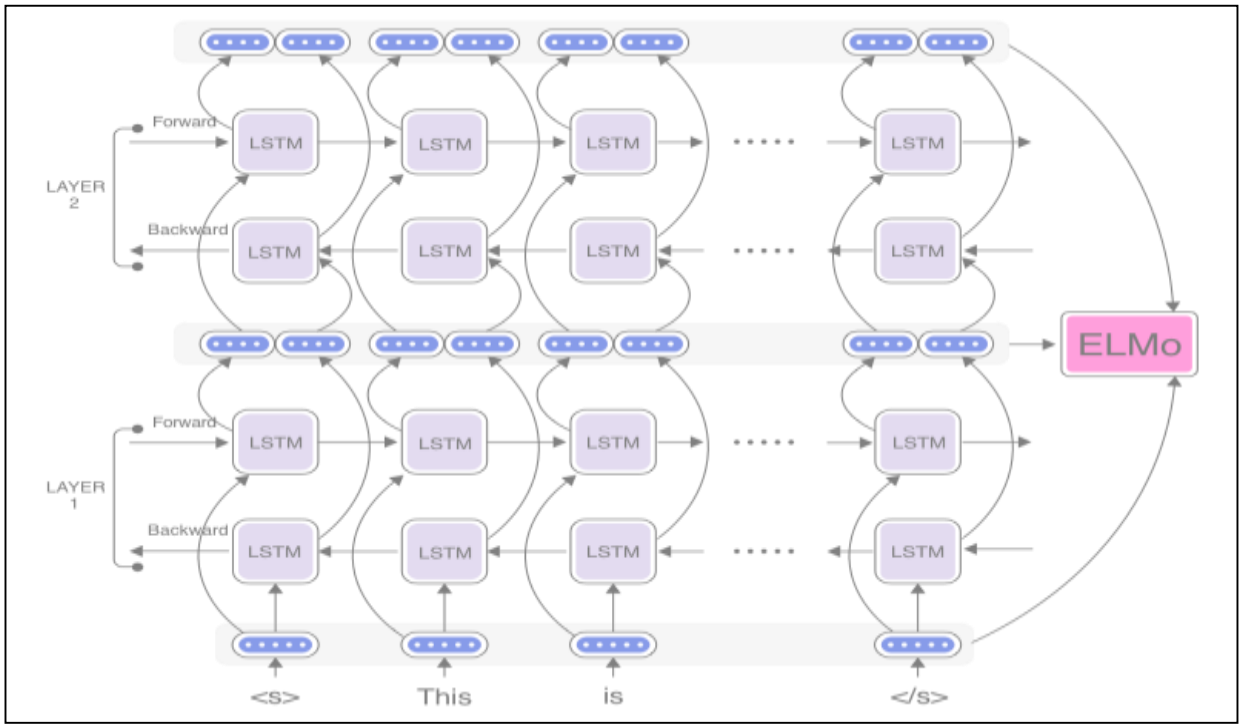

FIGURE 9 : L'architecture du modèle de ELMO.**[19]**

Ce diagramme est un exemple d'architecture ELMO à deux couches. Plus vous avez de couches, plus vous pouvez apprendre de contexte à partir de l'entrée. Les niveaux inférieurs identifieraient la grammaire et les règles syntaxiques de base, tandis que les couches supérieures extrairaient une sémantique contextuelle supérieure. L'autre aspect d'ELMO qui lui a permis d'être plus précis est qu'il a utilisé la modélisation bidirectionnelle du langage .

Ainsi, au lieu de simplement lire une entrée du début à la fin, il la lit également de la fin au début. Cela lui permet de capturer le contexte complet des mots dans une phrase **[19]**.

3.2.2.2 BERT

L'acronyme BERT correspond à (Bidirectional Encoder Representations from Transformers) **[34]**. Comme son nom l'indique, il s'agit d'un modèle de représentation du langage qui s'appuie sur un module qu'on appelle «Transformer». Les Transformers sont des composants qui s'appuient sur des méthodes d'attention. Un Transformer est construit sur la base d'un encodeur et d'un décodeur. Un encodeur se compose de couches d'attentions reliées à l'encodeur précédent (ou l'entrée pour le premier) et de couches denses. Un décodeur se compose, de manière similaire, de couches d'attentions reliées au décodeur précédent et de couches denses ainsi que d'une nouvelle couche d'attention supplémentaire intercalée entre ces deux couches et reliée à la sortie du dernier encodeur (en parlerons en détail dans le point suivant). BERT se compose lui d'une suite d'encodeurs seulement (N = 12 ou 24 suivant la version : Base ou Large) **[35]**.

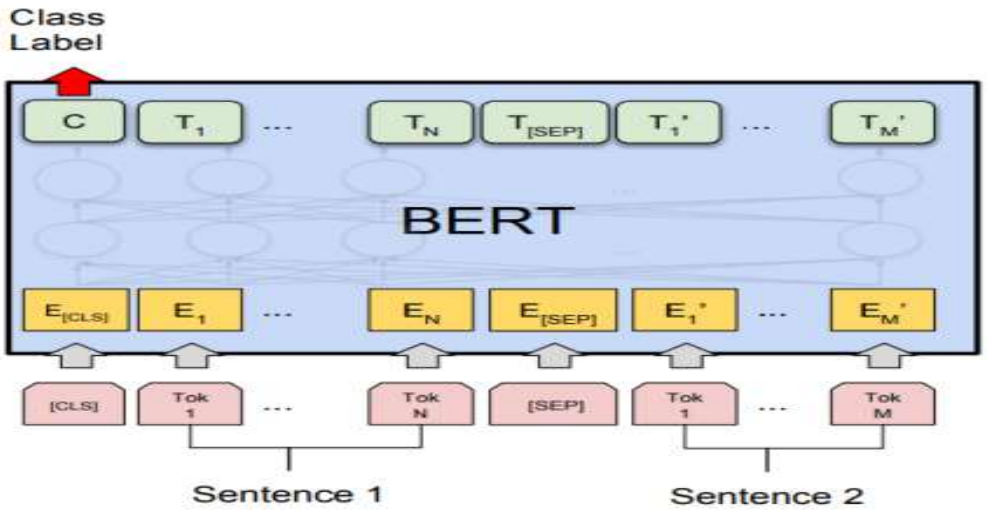

FIGURE 10: L'architecture du modèle de BERT.**[36]**

4. BERT

BERT est le surnom donné par Google à un changement majeur dans la manière dans fonctionnement de son moteur de recherche. Fondamentalement, BERT doit permettre de comprendre le « sens » d'une requête et, donc, la signification des mots qui sont utilisés dans un contexte précis. En d'autre treme, il ne s'agit plus de prendre les termes isolément, mais de les comprendre selon leur voisinage **[20]**.

Il s'agit là encore d'une variante du Transfer Learning. Le mode principal de fonctionnement de BERT correspond à un transfert par fine-tuning, il est aussi utilisable en mode transfert par extraction de caractéristique comme ELMo.

Schématiquement BERT se base sur deux innovations suivantes :

1. L'utilisation de l'architecture Transformer [AYN] qui constitue une alternative récente et performante aux RNN pour réaliser un préapprentissage bidirectionnel profond, des termes sur lesquels nous reviendrons. C'est l'innovation principale de BERT.
2. L'utilisation de deux nouvelles tâches pour le préapprentissage, l'une au niveau des mots et l'autre au niveau des phrases **[21]**.

4.1 L'architecture Transformer et l'apprentissage bidirectionnel

L'architecture Transformer a été publiée fin 2017 dans un article au titre accrocheur : (AYN: Attention is All You Need !). A terme, cette architecture a vocation à remplacer les RNN comme encodeurs de séquences. La motivation des concepteurs du Transformer était de pallier un des principaux inconvénients des RNN à savoir l'impossibilité de paralléliser leurs traitements en raison du caractère séquentiel des calculs qu'ils effectuent. Chaque cellule d'un RNN a en effet besoin d'accéder au résultat calculé par la cellule précédente, voir la figure 10 **[21]**.

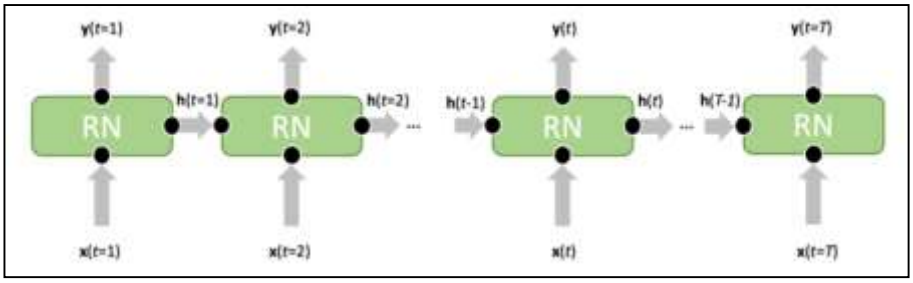

FIGURE 11: Le caractère séquentiel d'un RNN empêche la parallélisation des calculs.**[21]**

A l'échelle macroscopique un Transformer fonctionne comme un encodeur-décodeur utilisé dans un système de traduction automatique, voir la figure 11 (a). Il convertit une suite de WE (ou de token) en une autre suite, de longueur différente, de vecteurs. L'encodeur empile plusieurs petits encodeurs (six dans la figure 11) qui sont autant de couches d'abstraction et de même pour le décodeur qui empile plusieurs petits décodeurs. Seul l'encodeur du Transformer est utilisé en phase de préapprentissage par BERT.

La figure 11 (b) fait un zoom sur un des petits encodeurs. Après l'apprentissage, la couche de self-attention détecte les relations sémantiques qui existent entre différents vecteurs qu'elle reçoit en entrée (représentés en vert) et incorpore cette information dans les vecteurs qu'elle produit en sortie (représentés en rose). Ce mécanisme de self-attention intervient à chaque niveau d'abstraction de l'encodeur et du décodeur. Il intervient également entre l'encodeur et le décodeur mais ceci ne nous concerne pas pour BERT. Chaque vecteur en sortie de la couche de self-attention alimente un (même !) simple NN dense qui ajoute une couche d'abstraction à l'encodeur. Chaque instance de ce NN est indépendante des autres si bien qu'il est dorénavant trivial de paralléliser ces traitements.

Une difficulté à surmonter est qu'il n'existe pas de notion naturelle d'ordre des tokens en entrée d'un Transformer, chaque token étant équivalent aux autres par contraste avec un RNN qui définit explicitement un tel ordre. Il faudra par conséquent injecter explicitement l'information de position de chaque token dans son embedding, faute de quoi le Transformer

percevrait une phrase comme une simple collection désordonnée de mots (un « bag of words »). C'est ce que représente la figure 11 (c). L'encodeur du Transformer ne privilégie cependant aucune des deux directions de lecture d'une suite, en ce sens il est intrinsèquement bidirectionnel **[21]**.

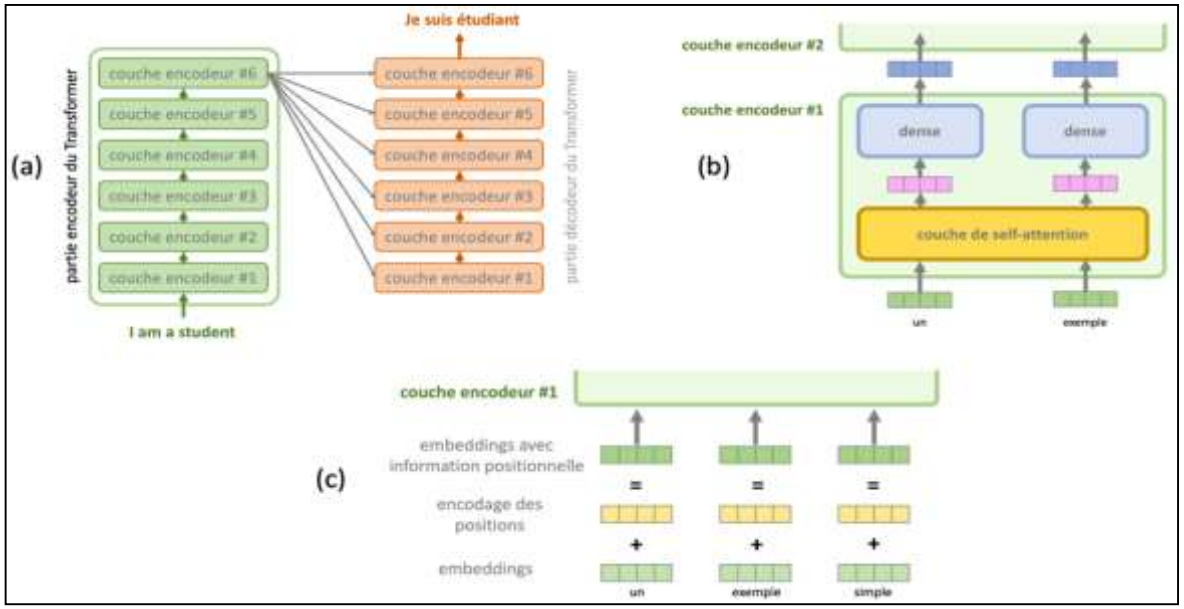

FIGURE 12: Les principaux éléments de l'architecture du Transformer.**[21]**

(a): Un encodeur, formé d'un empilement de couches de petits encodeurs, convertit une suite de WE en une suite qui de vecteurs qui incorporent des informations contextuelles riches pour chaque token.

(b): Un zoom sur une couche d'encodeur.

(c): Les WE utilisés par le Transformer ajoutent aux WE initiaux une information de position **[21]**.

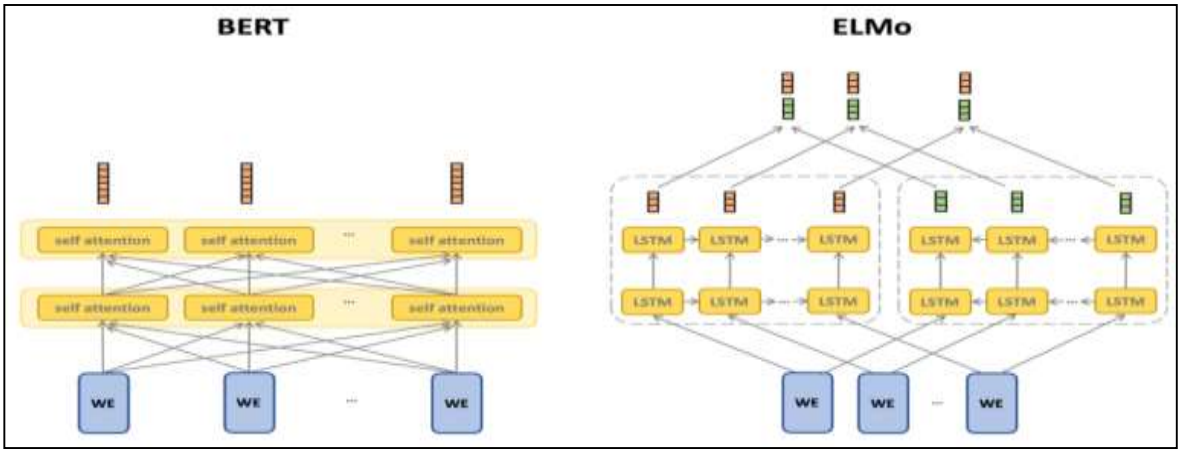

FIGURE 13: Comparaison entre l'architecture BERT réellement bidirectionnelle et l'architecture ELMo qui est une concaténation deux RNN unidirectionnels.**[21]**

ELMo était aussi bidirectionnel mais d'une manière plus superficielle puisqu'il était constitué de deux RNN unidirectionnels dont calculait la moyenne des prédictions. La figure 12 illustre cette différence entre ELMo et BERT **[21]**.

4.2 Les deux nouvelles tâches pour le préapprentissage

Le LM (Language Model) utilisés jusqu'ici sont des modèles unidirectionnels même si ELMo procède à une moyenne des deux directions. Dans BERT le Transformer est préentraîné simultanément sur deux tâches TLN qui, d'une part, ne nécessitent aucun travail de supervision manuel et, d'autre part, ne privilégient aucune direction des deux directions de lectures et mettent donc à profit le caractère bidirectionnel du modèle.

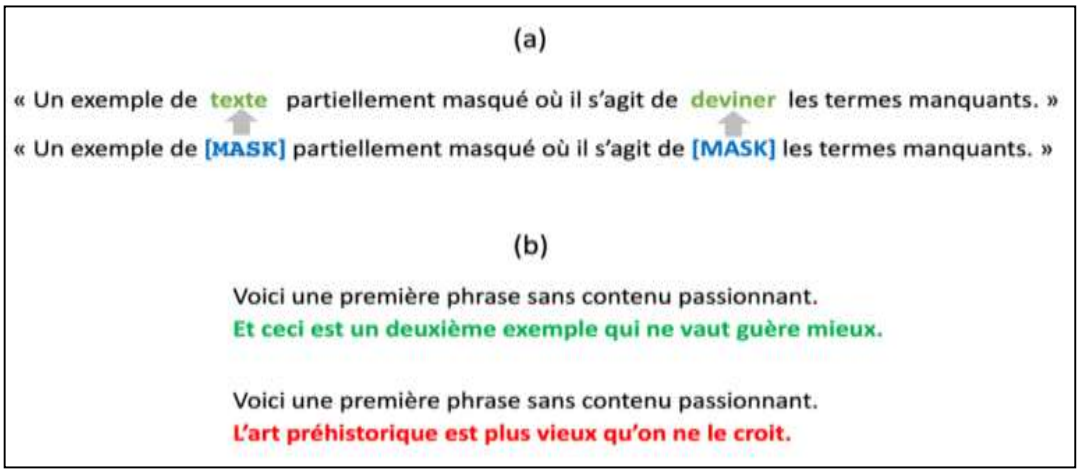

FIGURE 14: Les deux tâches utiliser dans BERT pour le préentraînement du Transformer. La première consiste à prédire des probabilités de mots masqués. La seconde consiste à déterminer si deux phrases sont consécutives.

La première de ces tâches consiste à entraîner le Transformer à prédire un certain nombre de termes (15% dans BERT) masqués et choisis aléatoirement dans le corpus source (MLM: Masked Language Model). Comme pour le LM cette tâche force le modèle à encapsuler une part importante de TNL aussi bien sur des aspects de syntaxe que de sémantique. Pour réaliser un tel modèle il suffit d'ajouter un softmax qui convertit le vecteur caché de la dernière couche du Transformer en une probabilité sur les mots du vocabulaire.

La deuxième tâche a vocation à apprendre à cerner les relations sémantiques entre deux phrases, une aptitude essentielle par exemple pour évaluer le degré de proximité sémantique entre deux phrases, pour identifier des relations logiques ou pour à répondre à des questions. On crée à cet effet un ensemble d'aprentissage formé de couples de phrases dont 50% sont effectivement consécutives et 50% sont choisies au hasard dans le corpus.

Le Transformer est entraîné simultanément sur les deux tâches avec une fonction de coût égale à la somme des fonctions de coût associées aux deux tâches **[22]**.

4.3 Les Avantages de BERT

- Plus performant que ses prédécesseurs en terme de résultats.
- Plus performant que ses prédécesseurs en terme de rapidité d'apprentissage.
- Une fois pré-entraîné, de façon non supervisée (initialement avec avec tout - absolument tout - le corpus anglophone de Wikipedia), il possède une "représentation" linguistique qui lui est propre. Il est ensuite possible, sur la base de cette représentation initiale, de le customiser pour une tâche particulière. Il peut être entraîné en mode incrémental (de façon supervisée cette fois) pour spécialiser le modèle rapidement et avec peu de données **[37]**.

4.5 Les Inconvénients de BERT

- L'une des limites de BERT est le manque de capacité à gérer une longue séquence de texte. Par défaut, BERT prend en charge jusqu'à 512 tokens.
- BERT n’a pas toujours bien saisi le sens de la question.

  exemple: À l’internaute qui demande « quel est l’État au sud du Nebraska », BERT ne donnait plus la réponse « Nebraska », avec un extrait de la page Wikipédia dans lequel on pouvait lire que le Kansas est l’État au sud. À la place, BERT évoquait « South Nebraska », une localité qui se trouve en Floride.
- Problème de langue. sur une recherche classique en espagnol, les résultats sont d’ordinaire donnés en espagnol. Mais avec BERT, ce sont aussi des pages en anglais qui ont été retournées, alors que la formulation de l’internaute suggère qu’il comprend l’espagnol et qu’il peut donc avoir une réponse dans cette langue.
- Avec BERT, les résultats étaient plutôt textuels, alors qu’un support visuel peut être plus utile **[38]**.

4.6 les solutions proposées

La longueur de séquence maximale pour BERT est de 512. Le premier problème pour implémenter BERT pour classer du texte est de savoir comment il gère le texte d'une longueur supérieure à 512. Nous expérimentons les méthodes suivantes pour travailler avec de longs articles.

Méthodes de troncature

Habituellement, les informations de base de l'article se trouvent au début et à la fin. Nous utilisons trois méthodes de texte tronqué différentes pour effectuer l'ajustement BERT.

- head-only: prenez les 512 premiers tokens.
- tail-only: prenez les 512 derniers tokens.
- head+tail: Trial a défini les 128 premières tokens et les 382 dernières tokens **[27]**.

Méthodes hiérarchiques

Le texte d'entrée est d'abord divisé en k = L / 510 fractions, qui est introduit dans BERT pour obtenir la représentation des k fractions de texte. La représentation de chaque fraction est l'état caché des jetons [CLS] de la dernière couche. Ensuite, nous utilisons la mise en commun moyenne, la mise en commun maximale et l'attention personnelle pour combiner les représentations de toutes les fractions.

La méthode de troncature head+tail permet d'obtenir les meilleures performances sur les ensembles de données IMDb et Sogou **[27]**.

Longformer (The Long-Document Transformer)

Les modèles basés sur des transformateurs sont incapables de traiter de longues séquences en raison de leur opération self-attention, qui évolue de manière quadratique avec

la longueur de la séquence. Pour remédier à cette limitation, nous introduisons le Longformer avec un mécanisme d'Attention qui évolue linéairement avec la longueur de la séquence, ce qui facilite le traitement de documents de milliers de token ou plus. Le mécanisme d'attention de Longformer est un remplacement instantané de standard self-attention et combine une local windowed attention avec une global attention induisant une tâche.

Longformer a été évalué sur la modélisation du langage au niveau des caractères sur text8 et enwik8 **[39]**.

BERT-AL (BERT FOR ARBITRARILY LONG DOCUMENT UNDERSTANDING)

appliqué Transformer pour modéliser uniquement la dépendance locale et capturer de façon récurrente une longue dépendance en insérant du LSTM multi-channel dans chaque couche de BERT. Le modèle proposé est nommé BERT-AL (BERT for Arbitrarily Long Document Understanding) et il peut accepter une entrée arbitrairement longue sans re-pretraining à partir de zéro. Nous démontrons l'efficacité de BERT-AL sur la synthèse de texte en menant des expériences sur l'ensemble de données CNN / Daily Ma.**[40]**

Conclusion

Dans ce chapitre Nous nous sommes intéressés dans un premier temps à présenter WE qui sont les représentations actuelles les plus utilisées dans les dernières méthodes de traitement du langage. C'est devenu un outil incontournable à tester lors de manipulations de texte, Nous avons également abordé l'explication de leurs différents types basées sur le comptage ou basées sur les réseaux de neurones.

Chapitre 2

CONCEPTION

1. Introduction

Dans le chapitre précédent nous avons présenté un aperçu sur le Word embedding. Dans ce chapitre on va présenter la conception de notre système, accélération du processus d'intégration de termes dans le deep learning. cette phase est importantes pour atteindre l'objectif de notre système, car la phase de conception est la clé d'un développement du logiciel efficace, facile à réaliser et à maintenir.

Ce chapitre est consacré à l'étude conceptuelle du système, toutes les étapes et les éléments nécessaires pour atteindre notre objectif. Alors l'organisation de ce chapitre sera comme suit :

D'abord, on présente une description brève de notre système. Ensuite nous construisons sa conception globale qui décrit son architecture globale. En fin, une conception détaillée qui explique les différentes étapes du système, chacune indépendamment.

2. Description du système

L'objectif de notre système est accélération du processus d'intégration de termes dans le deep learning. A l'aide d'un modèle BERT. Nous utiliserons un ensemble de données public de la BBC News , Cette base de données contient également Text summarization.

Après charger BBC News, on doit transférer les fichiers de l'ensemble de données vers des listes de données, ensuite supprimer quelques unes qui sont inutiles dans ces listes pour que nous puissions y travailler, et extraire l'embedding de texte en utilisant différentes méthodes, nous expliquerons plus tard. Le choix de la méthode appropriée peut être basé sur le temps de d'exécution le plus court ou sur la base de la méthode la plus proche du contexte, ou peut-être les deux.

3. Conception globale

D'abord, en spécifie l'entrée unique du système qui est l'ensemble de données, ensuite on passe à la phase de prétraitement du l'ensemble de données, qui sert à charger et organiser ces données (des fichiers.txt) puis, la phase de tokenization qu'elle fait référence à la division d'une phrase en mots individuels en suite la phase extraire embedding que nous utiliserons pour découvrir les meilleures méthodes proposées.

Le schéma ci-dessous présente l'architecture générale du système, en montrant les entrées du système, les différentes fonctionnalités et les sorties.

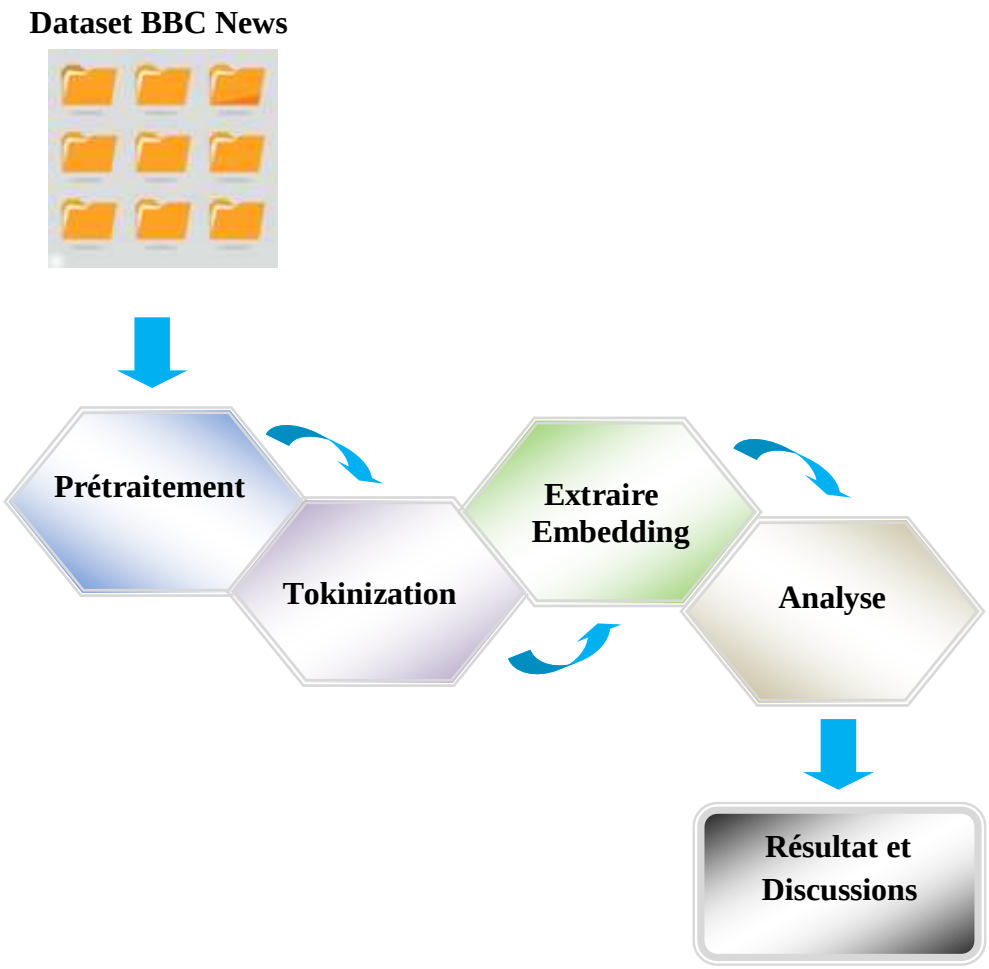

FIGURE 15: schémas générale du système.

4. Conception détaillée

Dans la partie de conception détaillée, on va expliquer l'architecture générale du système signalée dans la section précédente.

4.1 Les entrées du système

Avant tout, doit apporter l'ensemble de données (Dataset), Nous utiliserons un ensemble de données public de la BBC , cet ensemble de données a été créé à l'aide d'un ensemble de données utilisé pour la catégorisation des données qui contient 2225 documents du site Web d'actualités de la BBC correspondant à des articles dans cinq domaines d'actualité (business, entertainment, politics, sport et tech.) de 2004 à 2005 utilisés dans l'article de

D.Greene et P.Cunningham "Practical Solutions to the Problem of Diagonal Dominance in Kernel Document Clustering", Proc. ICML 2006 **[23]**.

Cette base de données contient également Text summarization , Text summarization est un moyen de condenser la grande quantité d'informations sous une forme concise par le processus de sélection des informations importantes et de rejet des informations sans importance et redondantes. Avec la quantité d'informations textuelles présentes sur le World Wide Web, le domaine de Text summarization devient très important. l'extractive summarization est celui où les phrases exactes présentes dans le document sont utilisées comme résumés. L' extractive summarization est plus simple et est la pratique courante chez les chercheurs en récapitulation automatique de texte à l'heure actuelle. Le processus de l'extractive summarization consiste à attribuer des scores aux phrases en utilisant une méthode, puis à utiliser les phrases qui obtiennent les scores les plus élevés en tant que résumés **[24]**.

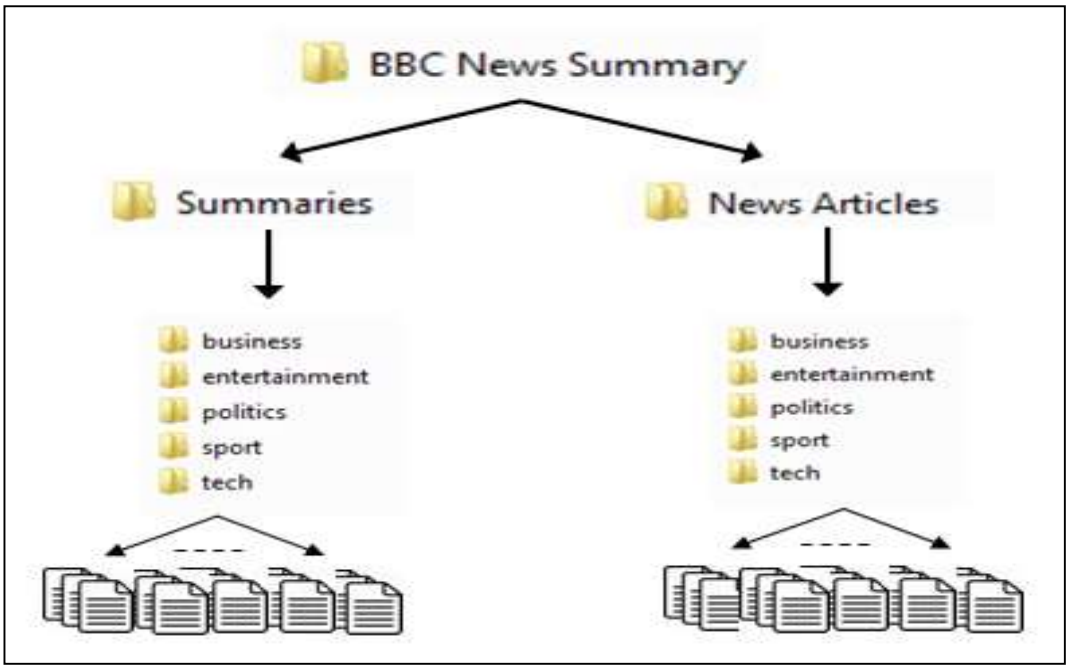

FIGURE 16: Ensemble de donnees BBC NEWS.

4.2 Prétraitement

La phase prétraitement de données se compose de chargement et organisation les données.

notre base de données est composé de ensemble:

News Articles: Composé de 2225 documents du site Web d'informations de la BBC correspondant à des articles dans cinq domaines d'actualité durant periode 2004 à 2005.

Summaries: Composé de 2225 résumés (summaries) pour chaque News Article.

Comme une première étape dans notre système après l'opptention du l'ensemble de données , nous le chargerons et le nettoyons dans un second étape, c'est-à-dire qu' à chaque fois que nous chargerons un fichier de News Article, nous chargerons son Summarie afin que nous puissions accéder facilement à News Article et à son Summarie. Le nettoyage consiste à supprimer News Article qui contient nombre de tokens au dessous 510, et summaries au dessus 510.

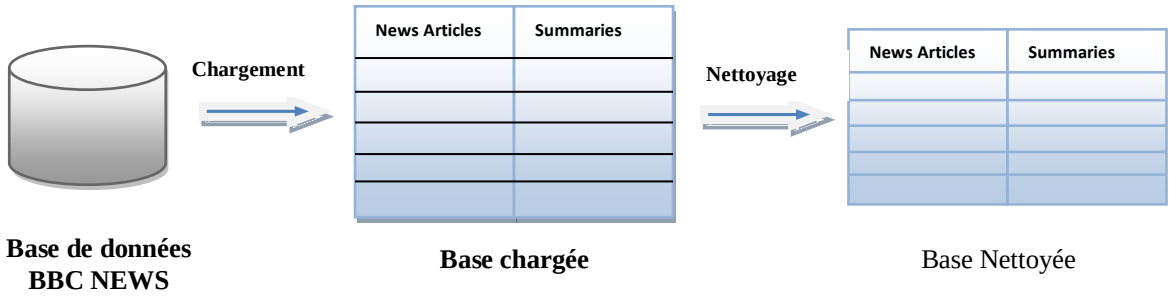

FIGURE 17: La phase de prétraitement.

4.3 Tokinization

La tokenization divise le texte brut en mots appelées tokens. Ces tokens aident à comprendre le contexte ou à développer le modèle de la TLN. La tokenization aide à interpréter la signification du texte en analysant la séquence des mots.

Il existe différentes méthodes et bibliothèques disponibles pour effectuer la tokenization. Nous utiliserons La tokenization effectuée par le tokenizer inclus avec BERT pour d'abord diviser le mot en tokens. Ensuite, nous ajoutons les tokens spéciaux nécessaires (ce sont [CLS] à la première position, et [SEP] à la fin de la phrase).

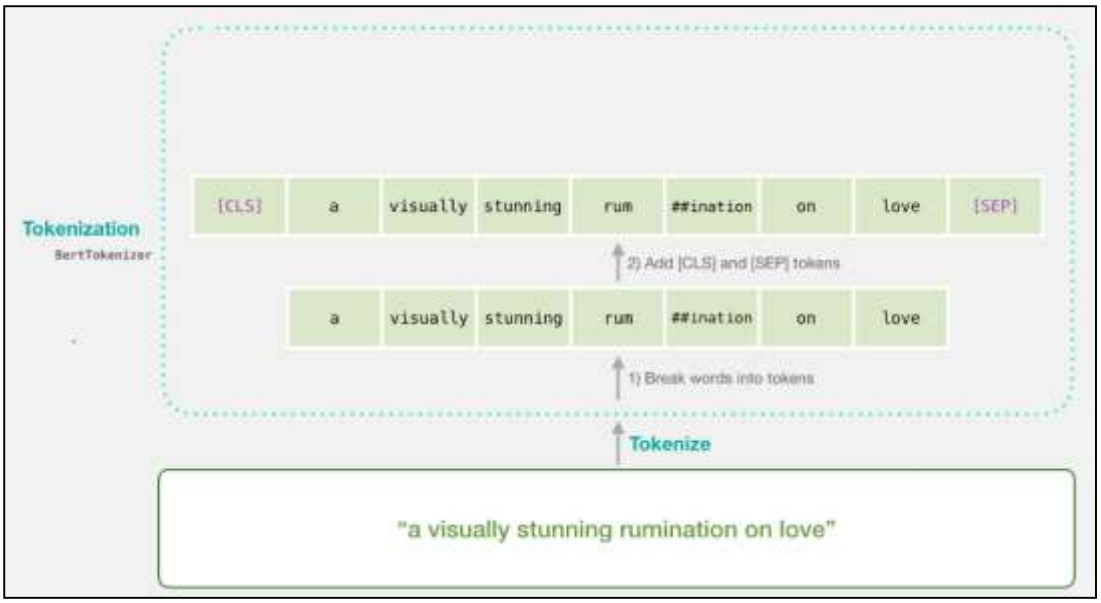

FIGURE 18: Tokinization.**[26]**

Le mot original a été divisé en sous-mots et caractères plus petits. Les deux signes de hachage précédant certains de ces sous-mots ne sont que la façon dont notre tokenizer indique que ce sous-mot ou caractère fait partie d'un mot plus grand et est précédé d'un autre sous-mot. par exemple le mot "rumination" -----> "rum", "##ination"

Le tokenizer fait est de remplacer chaque token par son id de la table d'embedding qui est un composant que nous obtenons avec le modèle entraîné.

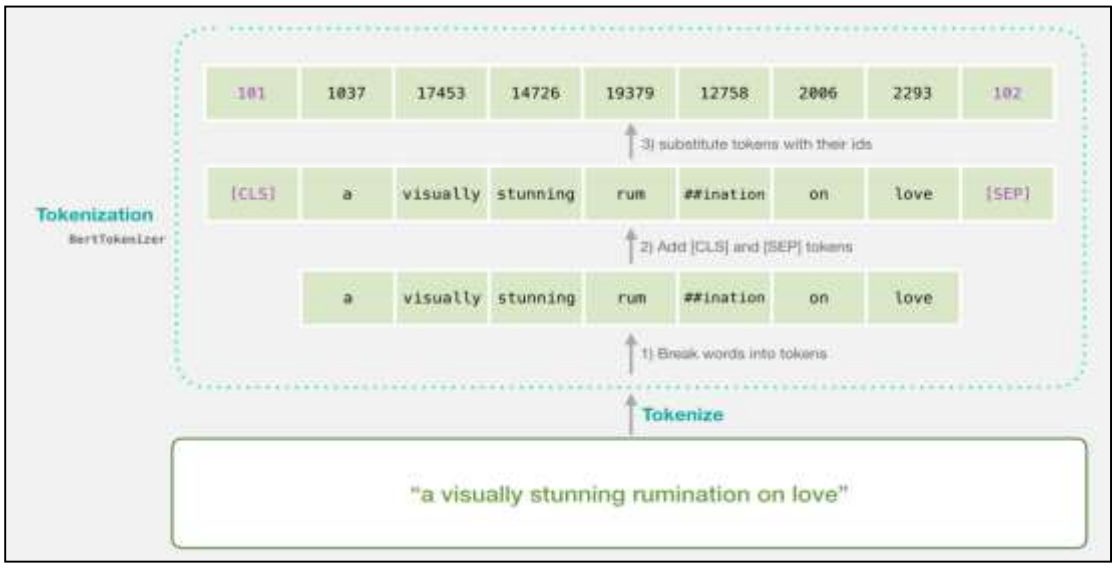

FIGURE 19: Tokinization+ids.**[26]**

4.4 Extraire Embedding

Les modèles TLN tels que les LSTMs ou les CNNs nécessitent des entrées sous forme de vecteurs numériques, ce qui signifie généralement la traduction de fonctionnalités telles que le vocabulaire et des parties du discours en représentations numériques. Dans le passé, les mots étaient représentés soit comme des valeurs indexées de manière unique (one-hot encoding), soit plus utilement comme des embeddings de mots neuronaux où les mots de vocabulaire sont comparés aux embeddings d'entités de longueur fixe qui résultent de modèles tels que Word2Vec ou Fasttext. BERT offre un avantage par rapport aux modèles comme Word2Vec, car si chaque mot a une représentation fixe sous Word2Vec quel que soit le contexte dans lequel le mot apparaît, BERT produit des représentations de mots qui sont dynamiquement informées par les mots qui les entourent. [25]

nous utiliserons BERT pour extraire des fonctionnalités, à savoir des vecteurs embedding de mots et de phrases, à partir de données textuelles. ces embeddings sont utiles pour l'expansion de mots-clés / recherches, la recherche sémantique et l'extraction d'informations.

La sortie serait un vecteur pour chaque token d'entrée. chaque vecteur est composé de 768 nombres (flottants).

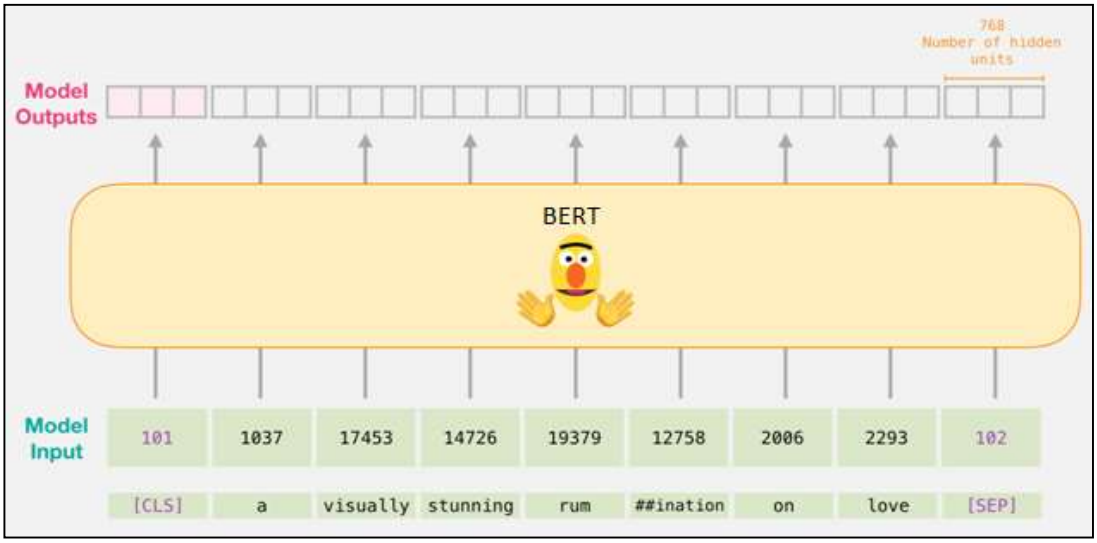

FIGURE 20: Extraction Embeddings**.[26]**

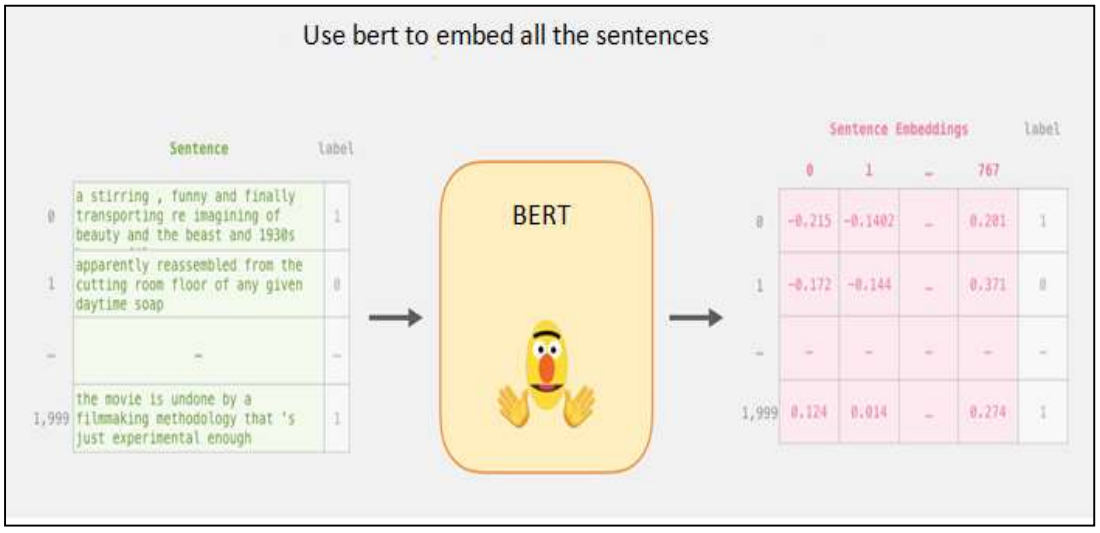

FIGURE 21: Extraction Embeddings toutes les phrases.**[26]**

La longueur de séquence maximale de BERT est de 512. Le premier problème de l'application de BERT est de savoir comment traiter le texte d'une longueur supérieure à 512. Nous essayons les méthodes suivantes pour traiter les articles longs:

- **Méthodes de Troncature (Truncation en anglais):**

  Habituellement, les informations clés d'un article se trouvent au début et à la fin ou milieu. Nous utilisons quatre méthodes différentes de troncature (Truncate) de texte pour effectuer un réglage fin de BERT.

  - head-only: prenez les 510 premiers tokens.
  - tail-only: prenez les 510 derniers tokens.**[27]**
  - midell: prenez les 510 tokens du milieu **.**
  - mix: prenez quelques parties du début, du milieu et de la fin.

- **Méthode de Division:**

  tokenize le long texte puis placez chaque 510 de jetons dans une liste puis obtenez embedding de chacun, en appliquant Averaging Method (Les méthodes les plus courantes basées sur BERT pour générer des embeddings de phrases en faisant simplement Averaging **[28]**.) lors d'embedding de la liste des morceaux.

4.5 Analyse

lors l'extraction d'embedding avec les différent méthodes qui nous avons proposé déjà, nous calculons le temps d'exécution de chaque méthode. Une fois l'extraction d'embedding terminée, nous calculons la similitude entre deux motifs vectoriels, c'est-à-dire entre l'embedding du Summaries et l'embedding de toutes les méthodes proposées précédemment. Il s'agit de savoir laquelle de ces méthodes est la plus rapide et lesquelles préservent les informations et le contexte.

4.6 Résultats et discussion.

Après avoir appliqué les étapes et méthodes susmentionnées, nous obtiendrons différents résultats qui nous permettront de savoir laquelle est la meilleure à utiliser.

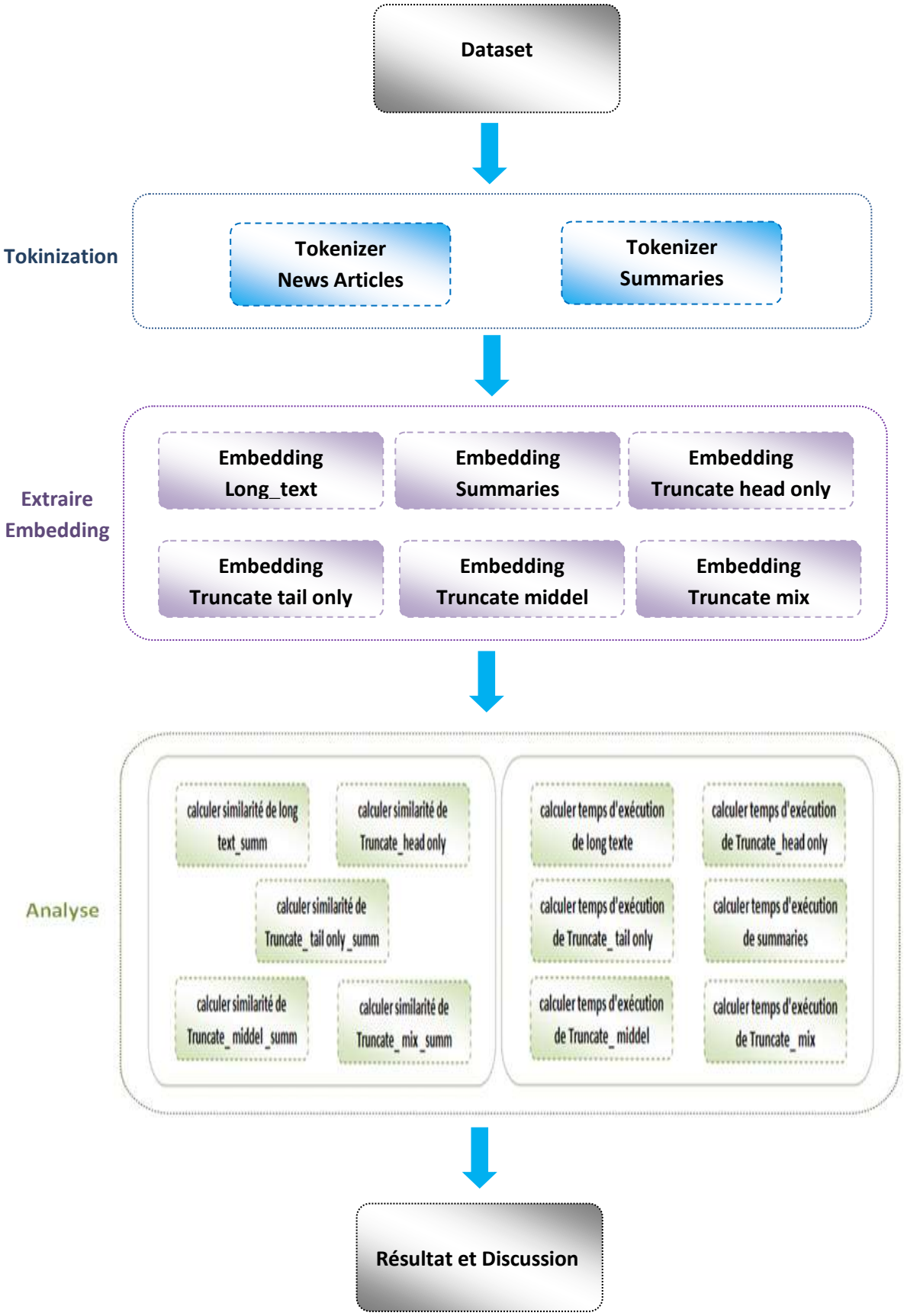

FIGURE 22: schémas détaillée du système.

Conclusion

Dans ce chapitre, on a présenté le cadre général de notre système, les entrées, les différentes fonctionnalités et les résultats. Ensuite on essayé de faire une explication conceptuelle sur le système visé.

Le chapitre qui se suit, est consacré pour la réalisation du système, en expliquant les outils utilisés et les différentes tâches réalisées.

Chapitre 3

IMPLE´MENTATION ET RE´ SULTAT

1. Introduction

Dans ce chapitre, nous décrivons la manière dont nous avons implémentés chaque module, ainsi les structures de donnés utilisés et les algorithmes de fonctionnement.

D'abord on commence par la présentation de l'environnement logiciel et les raisons pour lesquelles on a choisi le langage de programmation, ensuite une grande partie sera consacrée à la présentation de la réalisation de notre application et les résultats obtenus.

2. Outils D'implémentation

2.1 Environnement Logiciel

Colaboratory, souvent raccourci en "Colab", est un produit de Google Research. Colab permet à n'importe qui d'écrire et d'exécuter le code Python de son choix par le biais du navigateur. C'est un environnement particulièrement adapté au machine learning, à l'analyse de données et à l'éducation. En termes plus techniques, Colab est un service hébergé de notebooks Jupyter qui ne nécessite aucune configuration et permet d'accéder gratuitement à des ressources informatiques, dont des GPU.

FIGURE 23: Colab logo.

2.2 Choix du langage de programmation

Python

Python est un langage de programmation interprété, multi-paradigme et multiplateformes. Il favorise la programmation impérative structurée, fonctionnelle et orientée objet. Il est doté d'un typage dynamique fort, d'une gestion automatique de la mémoire par ramasse-miettes et d'un système de gestion d'exceptions ; il est ainsi similaire à Perl, Ruby, Scheme, Smalltalk et Tcl.

Le langage Python est placé sous une licence libre proche de la licence BSD5 et fonctionne sur la plupart des plates-formes informatiques, des smartphones aux ordinateurs centraux, de Windows à Unix avec notamment GNU/Linux en passant par macOS, ou encore Android, iOS, et peut aussi être traduit en Java ou .NET. Il est conçu pour optimiser la productivité des programmeurs en offrant des outils de haut niveau et une syntaxe simple à utiliser.

Python est un langage qui peut s'utiliser dans de nombreux contextes et s'adapter à tout type d'utilisation grâce à des bibliothèques spécialisées. Il est cependant particulièrement utilisé comme langage de script pour automatiser des tâches simples mais fastidieuses, comme un script qui récupérerait la météo sur Internet ou qui s'intégrerait dans un logiciel de conception assistée par ordinateur afin d'automatiser certains enchaînements d'actions répétitives . On l'utilise également comme langage de développement de prototype lorsqu'on a besoin d'une application fonctionnelle avant de l'optimiser avec un langage de plus bas niveau. Il est particulièrement répandu dans le monde scientifique, et possède de nombreuses bibliothèques optimisées destinées au calcul numérique **[40]**.

FIGURE 24: Python logo.

NumPy

NumPy est le package fondamental pour le calcul scienti que avec Python. Outre ses utilisations scienti ques évidentes, NumPy peut également être utilisé comme un conteneur multidimensionnel e cace de données génériques **[41]**.

FIGURE 25: Numpy logo.

Pandas

Pandas est un package Python fournissant des structures de données rapides, flexibles et expressives conçues pour faciliter le travail avec des données relationnelles ou étiquetées **[42]**.

FIGURE 26: Pands logo.

matplotlib

matplotlib est une bibliothèque de traçage (plotting) pour le langage de programmation Python et son extension de mathématiques numériques NumPy. Il fournit une API orientée objet pour embedding des tracés dans des applications à l'aide de boîtes à outils GUI à usage général telles que Tkinter, wxPython, Qt ou GTK **[43]**.

FIGURE 27: matplotlib logo.

3. Présentation de l'Application

3.1 Importer des bibliothèques

La première étape consiste à importer ce dont nous avons besoin à partir des bibliothèques Python, et ce sont les bibliothèques utilisées comme indiqué dans le Figure 28:

```
import os
import re
import sys
import torch
import time
import fnmatch
import random
import textwrap

import numpy as np
import pickle as p
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns


from sklearn.model_selection import train_test_split
from keras.preprocessing.sequence import pad_sequences

!pip install textwrap3
from textwrap3 import wrap
```

FIGURE 28: Importer des bibliothèques.

3.2 Installation de Transformers

Les Transformers (anciennement appelés pytorch-transformers et pytorch-pretrained-bert) fournissent des architectures à usage général de pointe (BERT, GPT-2, RoBERTa, XLM, DistilBert, XLNet, T5, CTRL ...) pour Natural Language Understanding (NLU) et Natural Language Generation (NLG) avec plus de milliers de modèles pré-entraînés dans plus de 100 langues et une interopérabilité profonde entre PyTorch et TensorFlow 2.0.

```
# Installer les transformateurs de la bibliothèque huggingface nous donnera une interface pytorch pour travailler avec BERT
!pip install transformers
```

FIGURE 29: Installation de Transformers.

3.3 Chargement l'ensemble de données BBC News

Nous chargeons l'ensemble de données public de la BBC dans deux listes. on utilise la méthode listdir () qui retourne une liste contenant les noms des entrées du répertoire donné par path. La liste est dans un ordre arbitraire.

La liste retournée nous aide à accéder aux fichiers News Articles Et son Summaries.

```
nom_fichier = []
for file in os.listdir(chemin_n):
  if fnmatch.fnmatch(file.upper(), '*.TXT'):
    nom_fichier.append(file)
```

FIGURE 30: Méthode listdir ().

Après le chargement l'ensemble de données BBC nous l'avons mis dans le data frame pour faciliter l'accès et le suivi.

	News Articles	Summaries
0	Japan economy slides to recession\n\nThe Japan...	Japan's economy grew 2.6% overall last year - ...
1	Sluggish economy hits German jobs\n\nThe numbe...	But officials said stagnant growth was still s...
2	SA unveils 'more for all' budget\n\nThe South ...	The South African government has put tax cuts ...
3	Peugeot deal boosts Mitsubishi\n\nStruggling J...	Struggling Japanese car maker Mitsubishi Motor...
4	Japanese banking battle at an end\n\nJapan's S...	Japan's Sumitomo Mitsui Financial has withdraw...

FIGURE 31: Chargement de BBC dnas Data Frame .

3.4 Tokenizer BERT

Pour transmettre notre texte (News_Articles & Summaries) à BERT, il doit être divisé en tokens, puis ces tokens doivent être mappés à leur index dans le vocabulaire du tokenizer.

La tokenization doit être effectuée par le tokenizer inclus avec BERT - la cellule ci-dessous le téléchargera pour nous. Nous allons utiliser la version "non casée" ici.

```
from transformers import BertTokenizer, BertModel

tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertModel.from_pretrained('bert-base-uncased')
```

FIGURE 32: Tokenizer BERT.

Ensuite, nous obtenons la plage de valeurs de token (voir les Figure 33 et 34)

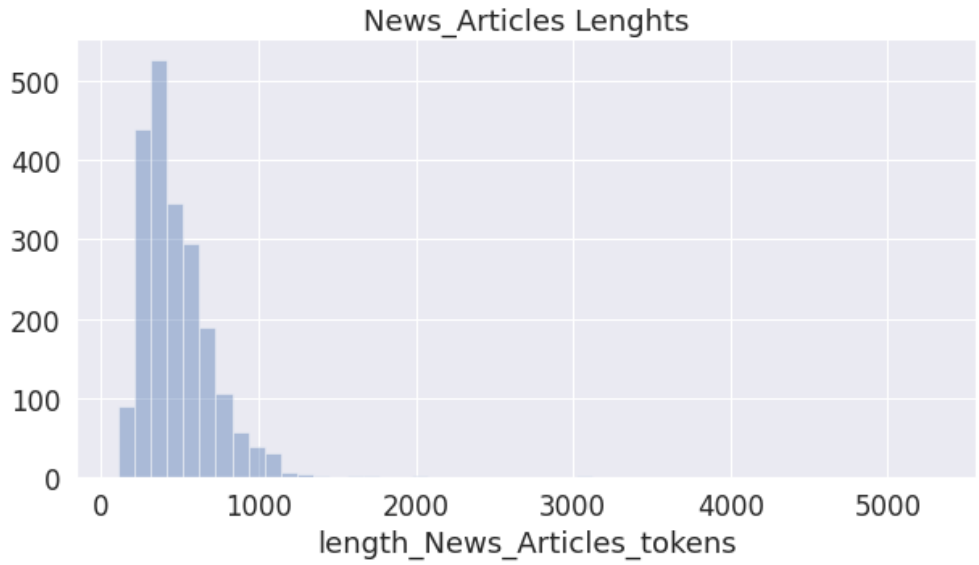

FIGURE 33: Histogramme de nombre de tokens dans News Articles.

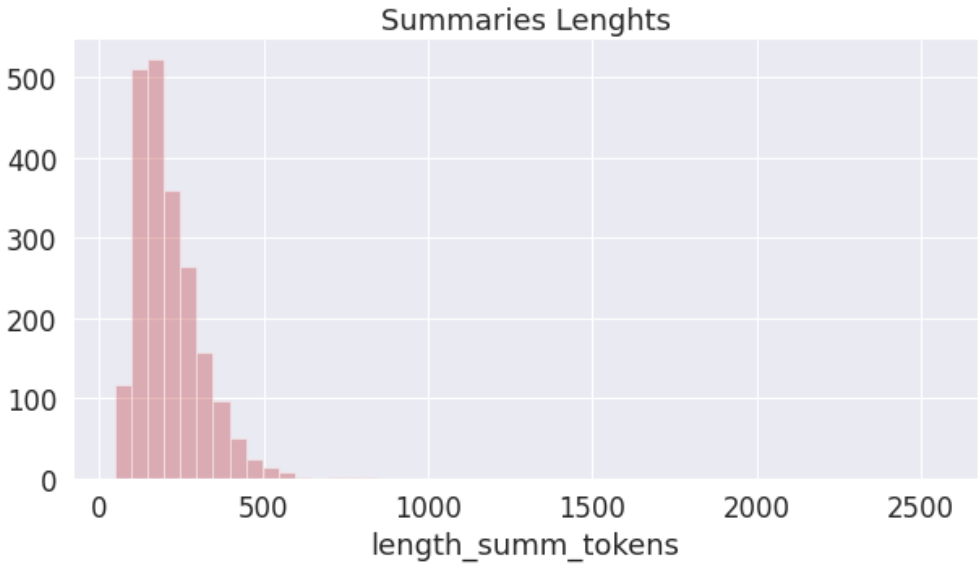

FIGURE 34: Histogramme de nombre de tokens dans Summaries.

3.5 Extraire Embedding

Bert prouvé le manque de capacité à gérer une longue séquence de texte. Par défaut, BERT prend en charge jusqu'à 512 tokens, Par conséquent, nous concentrerons ici sur ceux dont la longueur de token est supérieure ou égale à 510 dans les Articles, et inférieure ou égale à 510 dans le Summaries.

Sur cette base, nous préparerons les données sur lesquelles nous travaillerons en supprimant tous les News Articles dont la longueur de tokens ne dépasse pas 510 et les Summaries dont la longueur de tokens dépasse 510, comme le montrent les Figures 35 et 36.

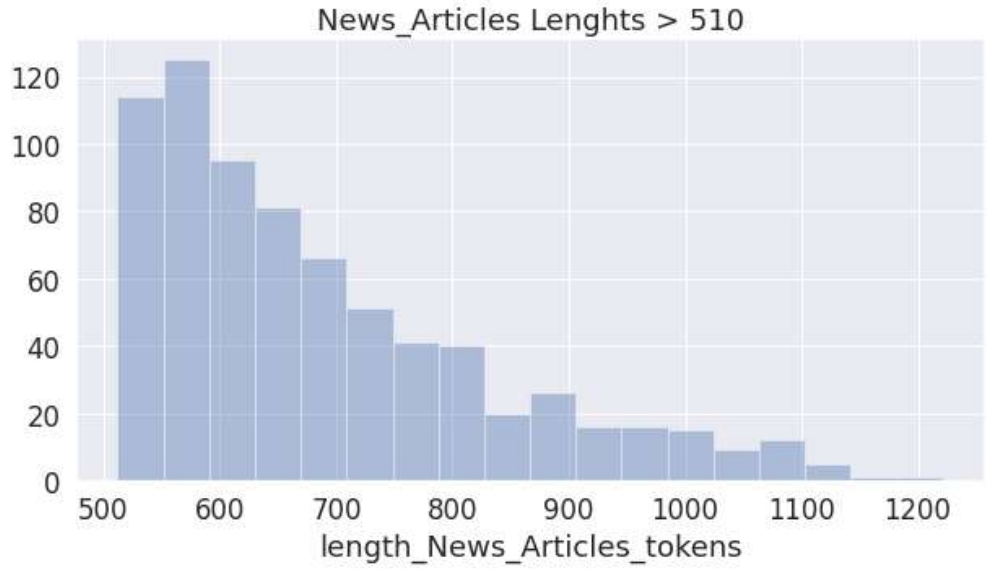

FIGURE 35: Histogramme de nombre de tokens > 510 dans News Articles.

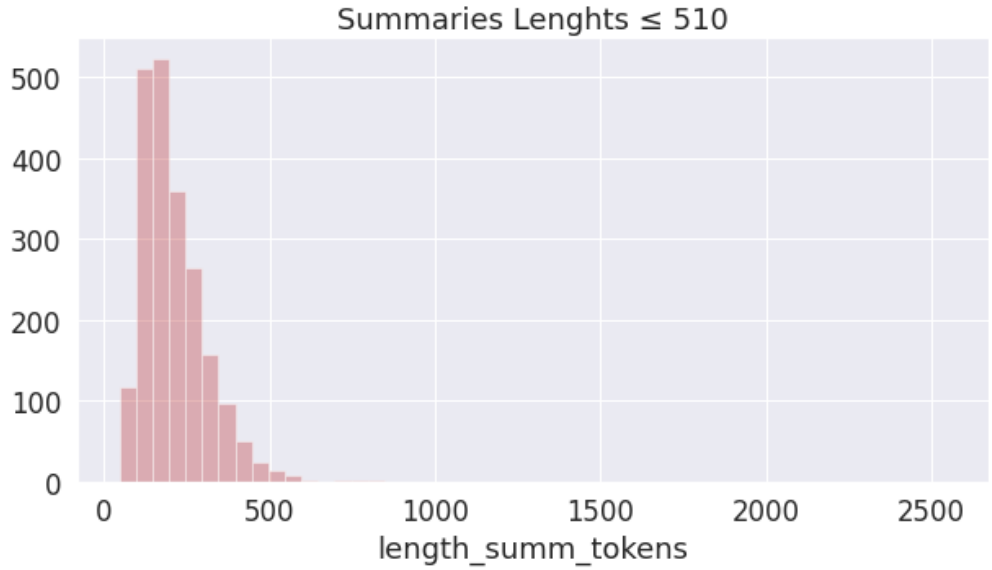

FIGURE 36: Histogramme de nombre de tokens < 510 dans Summaries.

Nous essayons les méthodes suivantes pour extraire embedding et traiter les News Articles longs :

- Méthodes de Troncature:
  - **head-only**: prenez les 510 premiers tokens.
  - **tail-only:** prenez les 510 derniers tokens.
  - **midell:** prenez les 510 tokens du milieu **.**
  - **mix:** prenez quelques parties du début, du milieu et de la fin.
- Méthode de Division long texte.
- Text summarization.

3.6 Calculer la similarité

Nous comparerons l'embedding générée afin d'obtenir une similitude sémantique entre les deux vecteurs. Cosine similarity C'est la méthode la plus utilisée pour comparer deux

vecteurs. C'est un produit ponctuel entre deux vecteurs. Nous trouverons l'angle cosinus entre les vecteurs. Pour le degré 0, le cosinus vaut 1 et est inférieur à 1 pour tout autre angle.

On utilise fonction cosine_similarity pour calculer la similarité entre embedding summories et le reste embeddings c.à.d. calculer simularite entre embedding summories et embedding head only, entre embedding summories et embedding middel, ... et ainsi de suite.

```
from sklearn.metrics.pairwise import cosine_similarity
def get_cosine_similarity(feature_vec_1, feature_vec_2):
    return cosine_similarity(feature_vec_1.reshape(1, -1), feature_vec_2.reshape(1, -1))
```

FIGURE 37: Fonction cosine_similarity.

4. Résultats

4.1 Temps d'exécution

Après avoir calculé le temps d'exécution pour chacune des méthodes proposées, nous avons voulu clarifier les différentes valeurs en calculant chacune des moyennes arithmétiques , max et min pour chaque méthode comme indiqué dans tableau 3.

Méthodes	Mean	Min	Max
texte long	3.868012	3.697293	5.848661
truncate head only	1.891698	1.848812	2.850667
truncate tail only	1.895621	1.85394	3.116910
truncate middle	1.8880378	1.845902	2.834089
truncate mix	1.894305	1.854464	2.976481

Tableau 03 : Temps d'execution.

Figure suivante représente le temps d'exécution selon la méthode de texte long:

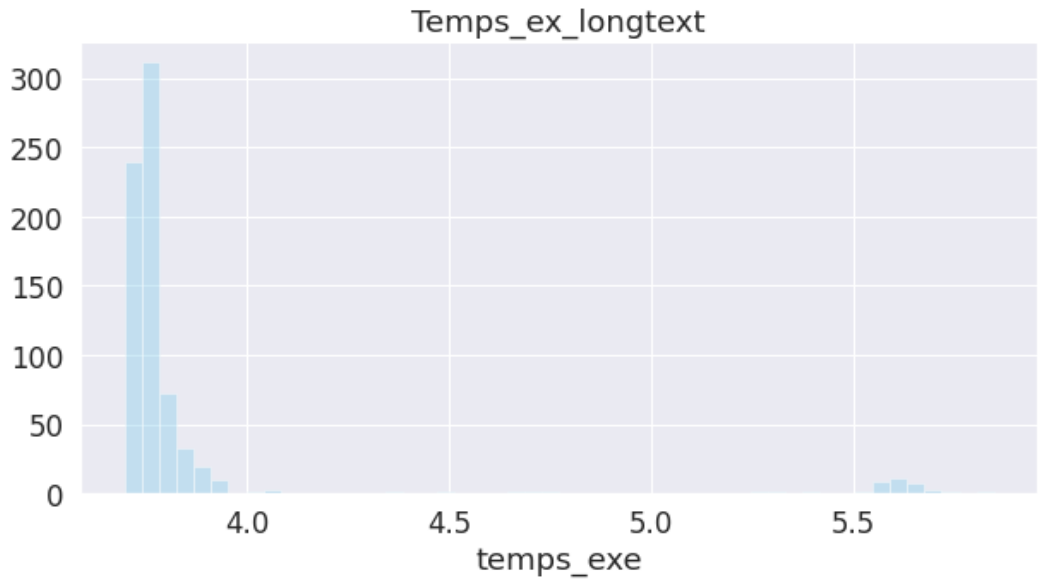

FIGURE 38: Histogrammes Temps d'exécution de méthode long texte.

La Figure suivante représente le temps d'exécution selon la méthode de truncate head only:

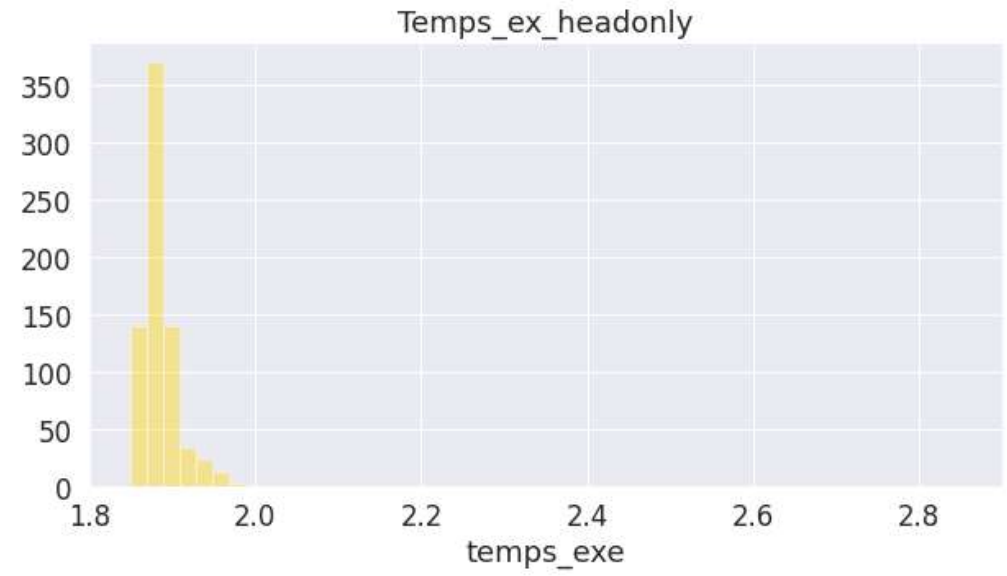

FIGURE 39: Histogrammes Temps d'exécution de méthode head only.

La Figure suivante représente le temps d'exécution selon la méthode de truncate tail only:

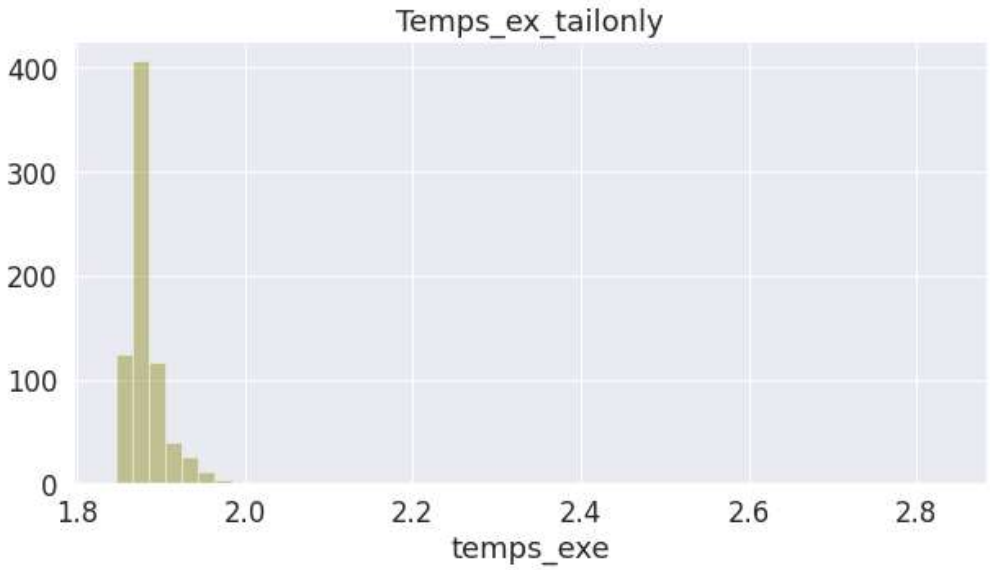

FIGURE 40: Histogrammes Temps d'exécution de méthode tail only.

La Figure suivante représente le temps d'exécution selon la méthode de truncate middle:

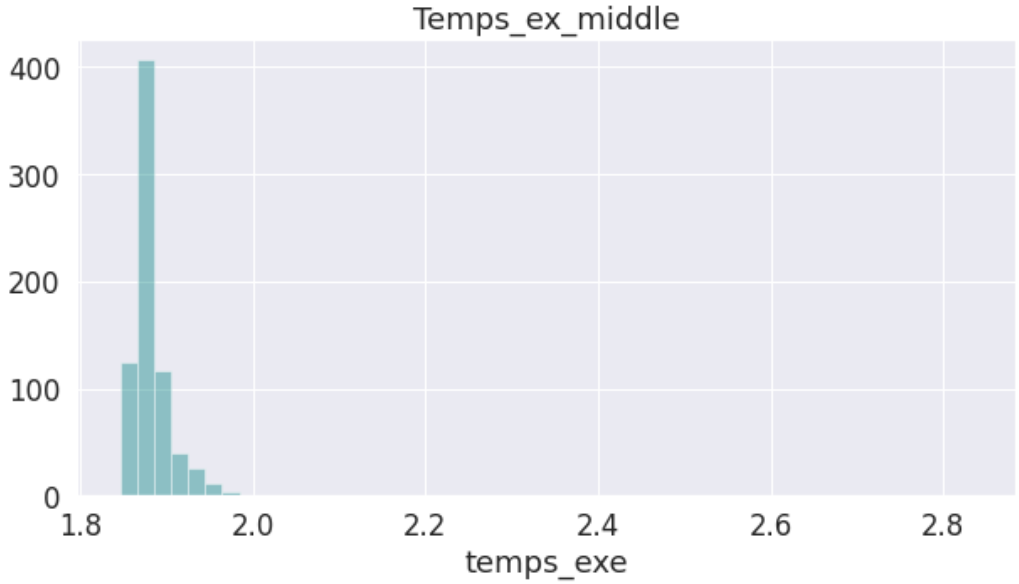

FIGURE 41: Histogrammes Temps d'exécution de méthode middel.

La Figure suivante représente le temps d'exécution selon la méthode de truncate mix :

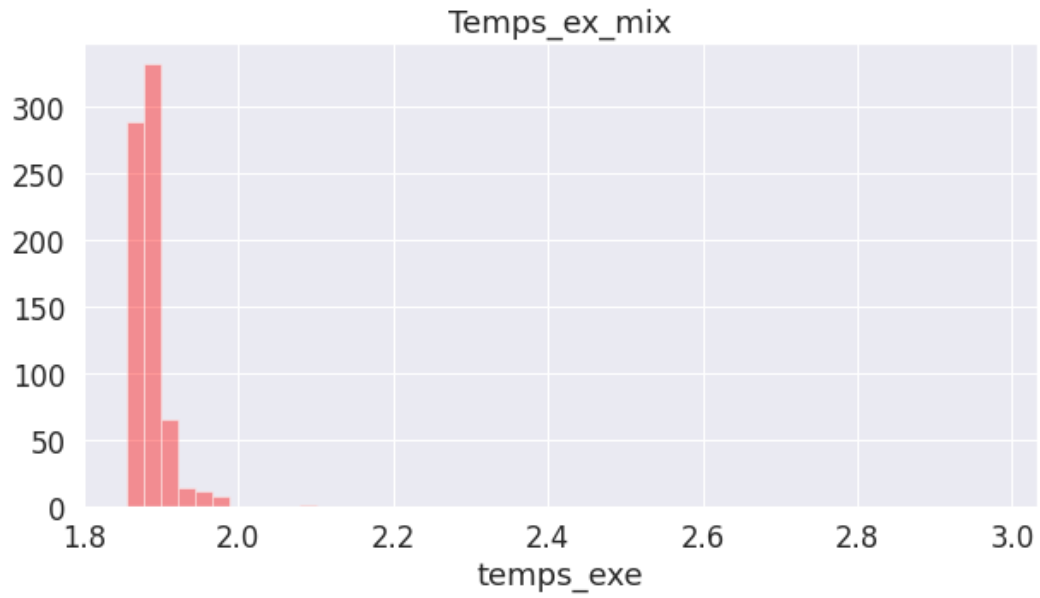

FIGURE 42: Histogrammes Temps d'exécution de méthode mix.

La figure suivante représente les temps d'execution de toutes les méthodes long texte , truncate head only, truncate middle, truncate tail only et truncate mix :

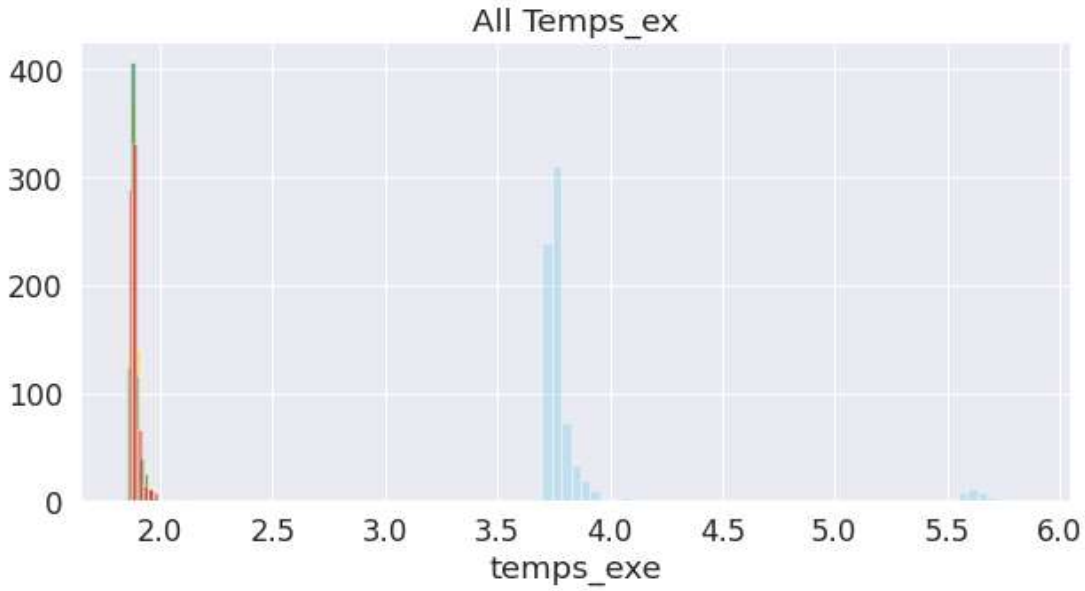

FIGURE 43: Histogrammes combine toutes les Temps d'Exécutions

4.2 Similarité:

La figure suivante représente la similarité entre long texte et summaries :

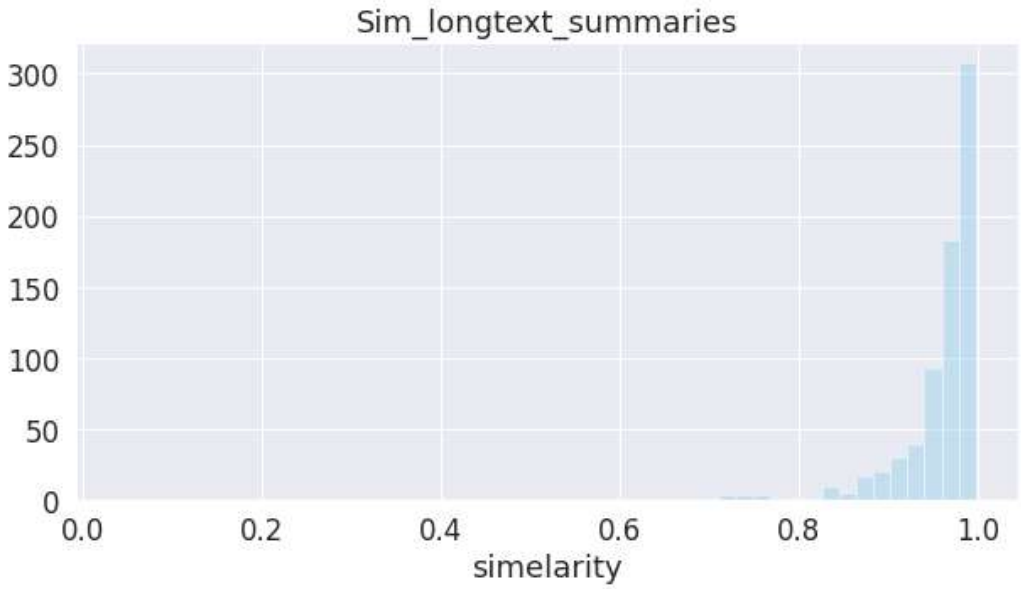

FIGURE 44: Histogrammes similarité entre long texte et summaries.

L
a figure suivante représente la similarité truncate head only et summaries :

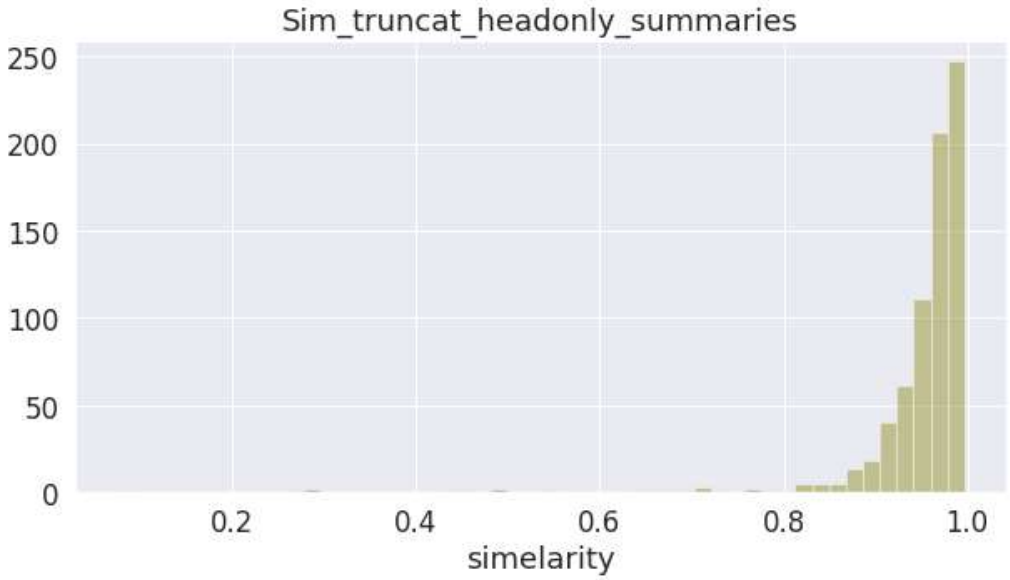

FIGURE 45: Histogrammes similarité entre truncate head only et summaries.

La figure suivante représente la similarité entre truncate tail only et summaries

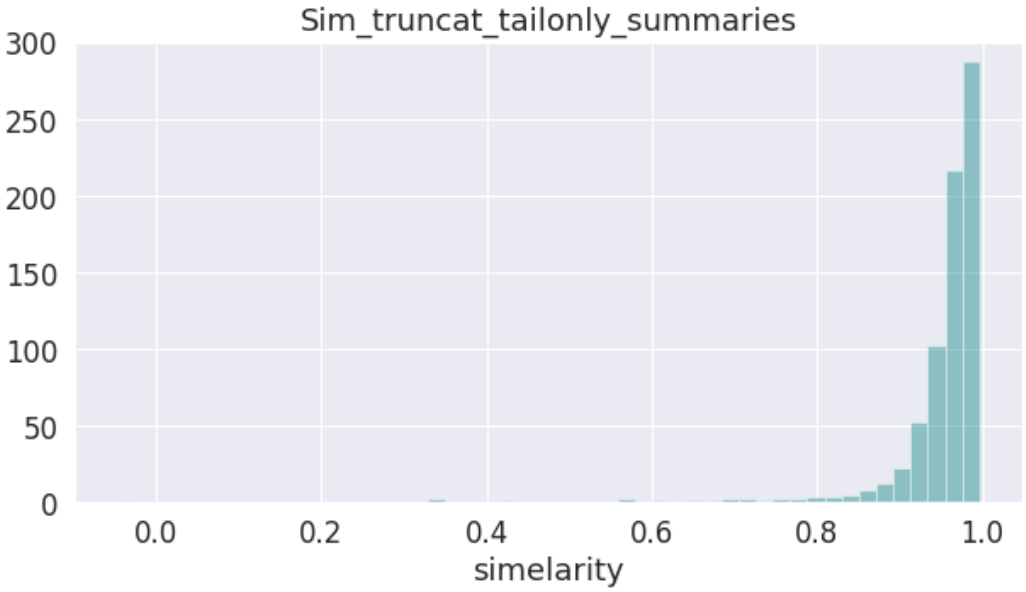

FIGURE 46: Histogrammes similarité entre truncate tail only et summaries.

La figure suivante représente la similarité entre truncate middel et summaries

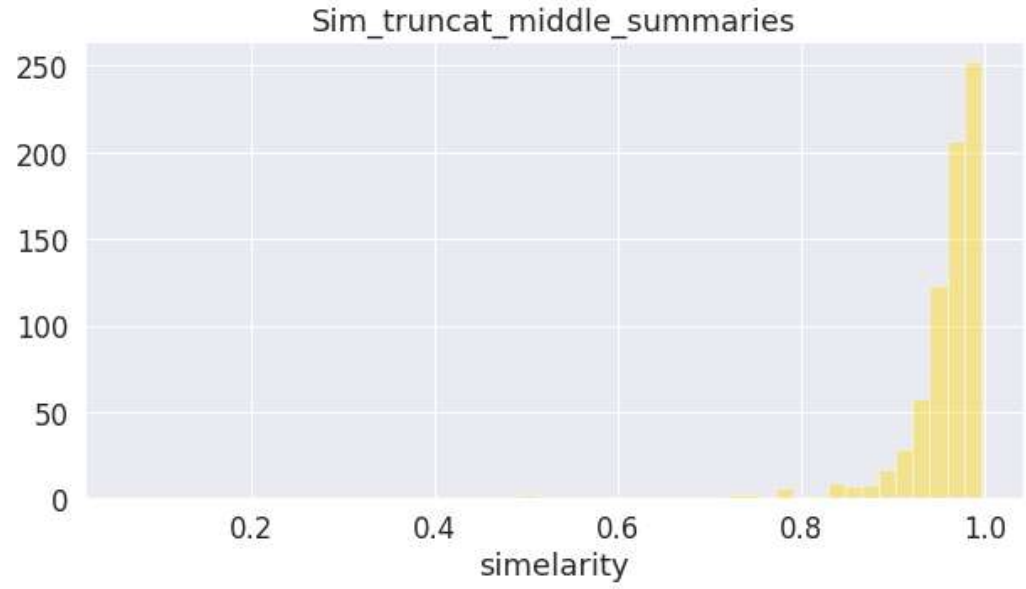

FIGURE 47: Histogrammes similarité entre truncate middel et summaries.

La figure suivante représente la similarité entre truncate mix et summaries

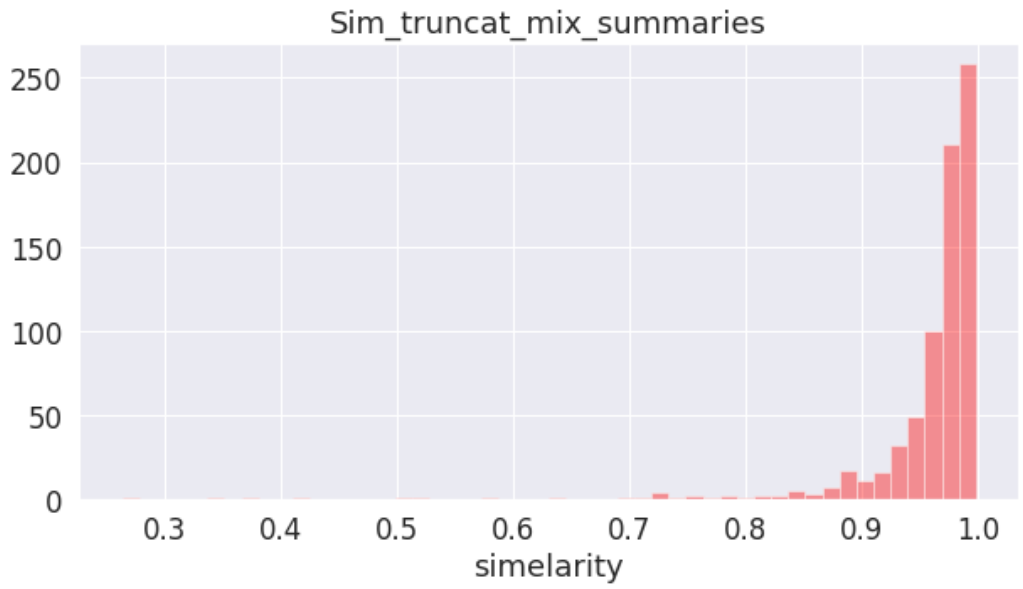

FIGURE 48: Histogrammes similarité entre truncate mix et summaries.

La figure suivante représente toutes les similarités, la similarité entre long texte , truncate head only, truncate middle, truncate tail only, truncate mix et summaries

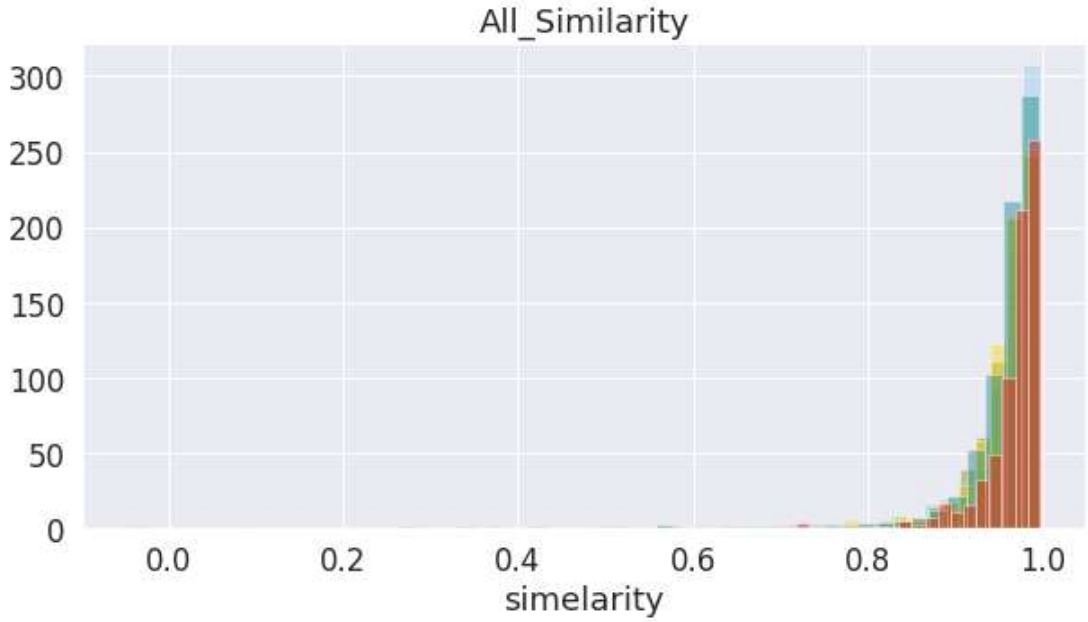

FIGURE 49: Histogrammes combine toutes les similarités

5. Discussion

La similarité cosinus mesure le degré auquel deux vecteurs pointent dans la même direction, quelle que soit leur amplitude.

Lorsque les vecteurs pointent dans la même direction, la similarité cosinus est 1; lorsque les vecteurs sont perpendiculaires, la similitude cosinus est égale à 0; et lorsque les vecteurs pointent dans des directions opposées, la similitude cosinus est -1.[27]

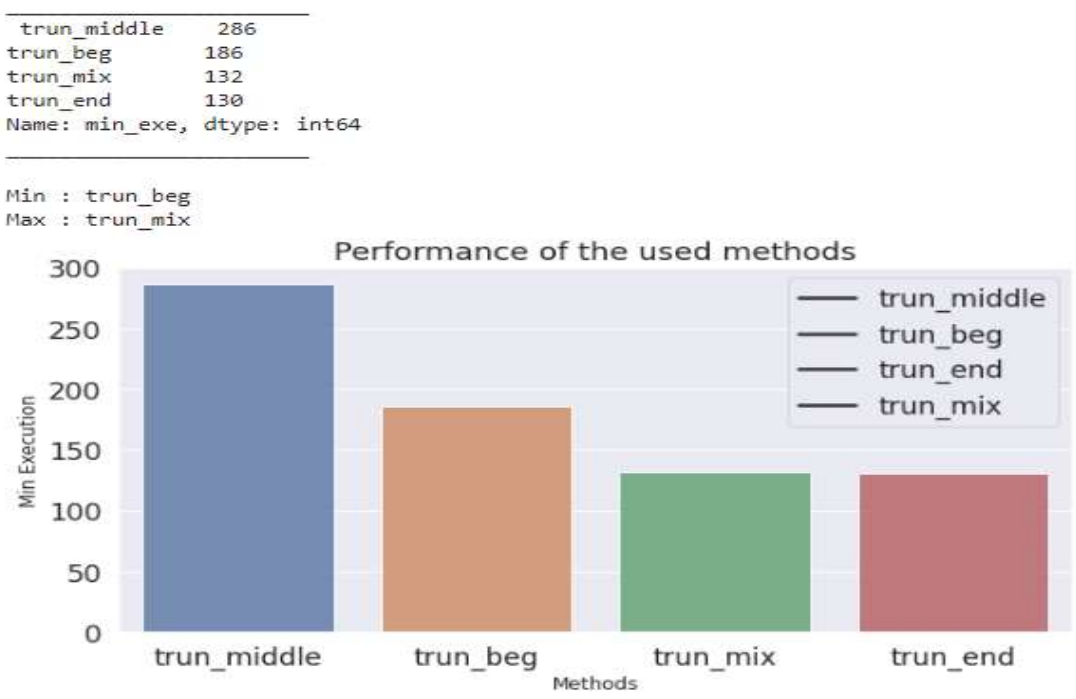

FIGURE 51: Histogrammes Performance des méthodes utilisées (Temps d'Excution)

```
sim_trun_mix       291
sim_larg_text      151
sim_trun_end       116
sim_trun_beg        91
sim_trun_middle     85
Name: Max_sim, dtype: int64
```

```
Min : sim_larg_text
Max : sim_trun_mix
```

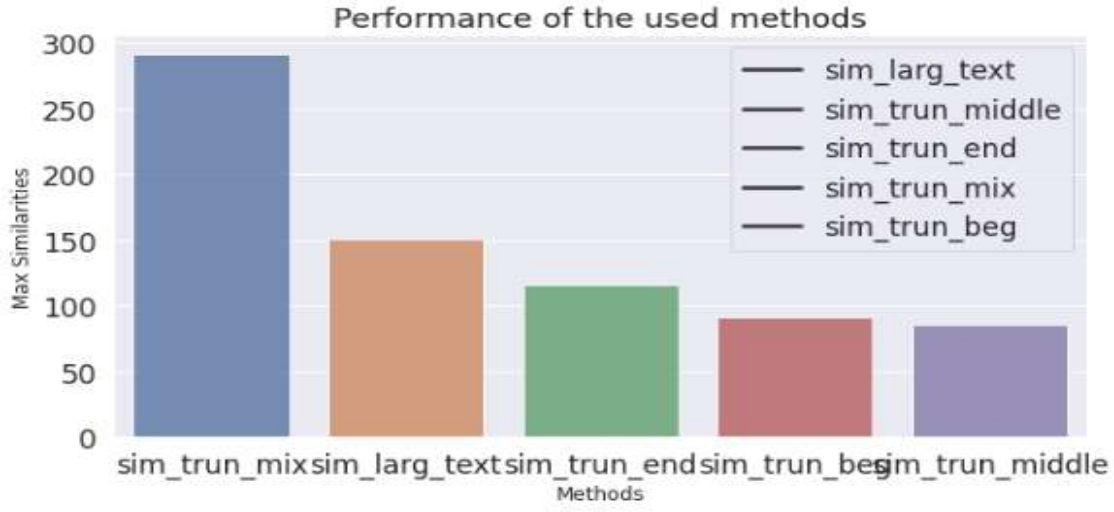

FIGURE 50: Histogrammes Performance des méthodes utilisées (Similarités)

Methodes	**Temps d'Exécution**	**Similarités**
texte long	/	20.57%
truncate head only	25.34%	12.39%
truncate tail only	17.71%	15.80%
truncate middle	38.96%	11.58%
truncate mix	17.98%	39.64%

Tableau 03 : Percentage temps d'execution et similarité.

A partir du l'Histogrammes Performance des méthodes utilisées (Similarités) et tableau 3, on remarque que la similarités entre les méthodes truncate mix et summaries, et similarités entre les méthodes Division long text et summaries est les plus elvé par rapport les autres,

comme pour le reste des méthodes proposées (truncate tail only, truncate head only ,truncate midelle) la disparité entre elles Ce n'est pas grand, et nous en concluons que méthodes truncate mix et Division long text sont les méthodes les plus proches du contexte des News Articles.

Quant à l'Histogrammes Performance des méthodes utilisées (Temps d'Excution) et tableau 3, on remarque que le temps d'execution de les methodes truncte middel et truncte head only est les plus elvé par rapport les autres, De là, nous concluons que les deux méthodes garantissent le moins de temps d'execution en œuvre possible par rapport aux autres.

Conclusion

Ce chapitre été consacré pour la dérivation de la manière dont on a réalisé notre système. D'abord on a présenté l'environnement utilisé pour la réalisation du logiciel, en montrant les raisons pour lesquelles on a fait ce choix. Ensuite on a présenté en détail les tâches réalisées et leurs résultats.

Chapitre 4

CONCLUSION GÉNÉRALE

Nombreuses sont les problématiques auxquelles on peut aujourd'hui répondre grâce à Traitement Automatique du Langage NLP, Nous avons étudié dans ce travail les différents modèles et proposé un travail basé sur le modèle de Bert.

Nous avons conçu et proposé dans ce travail des solutions pour éliminer l'un des problèmes du modèle Bert . C'est l'incapacité à gérer une longue séquence de texte. diviser le texte que ce soit en le divisant en parties ou en utilisant son résumé, ou en en prenant des parties, que ce soit au recto, au verso ou au milieu, ou en combinant les parties précédentes. Il s'agit de trouver la méthode la plus appropriée en termes de temps ou de contexte ou les deux.

Enfin comme perspective pour ce travail, nous proposons:

Si notre intérêt se concentre sur la vitesse de d'execution, alors la méthode appropriée est la méthode d'utilisation milieux de texte avec un pourcentage de 38,96%, suivie par l'utilisation de la premier partie du texte avec un pourcentage 25.34%.

S'il se concentre sur le context, alors la méthode appropriée est la méthode d'utilisation des parties mixte de text avec un pourcentage de 38.96%, suivie par l'utilisation de la division de texte en parties avec un pourcentage 20.57%.

Références

[1] **WORD EMBEDDIN**. URL: https://dataanalyticspost.com/Lexique/word-embedding/ (visité le: 24/02/2020).

[2] Josiane Mothe, Michel Rajoelina, Faneva Ramiandrisoa, Hary Razakasoa . **Intégration des plongements de mots dans les méthodes, supervisées et non supervisées, d'extraction automatique de mots clés** . 5e Seminaire Veille Strategique Scientifique et Technologique (Seminaire VSST 2018), Jun 2018, Toulouse, France.

[3] Benoit Cayla. **Les sacs de mots**. 28octobre 2018. URL :https://www.datacorner.fr/les-sacs-de-mots-bag-of-words/ (visité le: 08/03/2020)

[4] Hinrich Sch ¨utze. **Introduction to Information Retriev.** Institute for Natural Language Processing, University of Stuttgart. 29-08-2011.

[5] Richard Socher Mundra, Richard Socher . **Deep Learning for NLP1 1**. Notes de cours: Partie I2 2 Auteurs: Francois Chaubard, Rohit ,Printemps 2016.

[6] David S. Batista. **Language Models and Contextualised Word Embeddings**. 28octobre 2018. URL: http://www.davidsbatista.net/blog/2018/12/06/Word_Embeddings/ (visité le: 08/04/2020).

[7] Killian Janod , Mohamed Morchid , Richard Dufour , Georges Linares . **Apport de l'information temporelle des contextes pour la représentation vectorielle continue des mots** . LIA University of Avignon (France) ,ORKIS - Aix en Provence (France),2015

[8] Pirmin LEMBERGER ,Thomas SCIALOM. **DEEP TRANSFER LEARNING – LE TRAITEMENT DU LANGAGE À L'AUBE D'UNE RÉVOLUTION ?**. 28 nouvembre 2018. URL: https://weave.eu/deep-transfer-learning-nlp-revolution// (visité le: 08/04/2020).

[9] Grace Bouala. **Word Embedding Models : Word2vec, Camembert, USE**. Tuesday 28 January 2020. URL: https://blog.baamtu.com/word2vec-camembert-use-embedding-models/ (visité le: 010/04/2020).

[10] Sahar Ghannay. **Etude sur les representations continues de mots appliquees à la detection automatique des erreurs de reconnaissance de la parole**. Informatique et langage [cs.CL]. Thèse de doct. Université du Maine, 2017. Français. NNT : 2017LEMA1019.

[11] Arnaud Ferré. **Relier automatiquement des entités textuelles à des concepts d'une**

ontologie par apprentissage avec (presque) aucune donnée. INRA, BIBLIOME, MaIAGE Université Paris-Saclay, ILES, LIMSI CNRS, Université Paris Saclay.

[12] Jie Hu, Shaobo Li, Yong Yao, Liya Yu, Guanci Yang, Jianjun Hu. **Patent Keyword Extraction Algorithm Based on Distributed Representation for Patent Classification**. Key Laboratory of Advanced Manufacturing Technology of Ministry of Education, Guizhou University, Department of Computer Science and Engineering, University of South Carolina, Columbia, SC 29208, USA, School of Mechanical Engineering, Guizhou University, Guiyang 550025, China. Received: 8 November 2017; Accepted: 30 January 2018; Published: 2 February 2018.

[13] Jeffrey Pennington, Richard Socher, Christopher D. Manning. **GloVe: Global Vectors for Word Representation**. Computer Science Department, Stanford University, Stanford, CA 94305.

[14] Chang Liu, Pengyuan Zhang, Ta Li, Yonghong Yan. **Semantic Features Based N-Best Rescoring Methods for Automatic Speech Recognition**. Key Laboratory of Speech Acoustics and Content Understanding, Institute of Acoustics, Chinese Academy of Sciences, Beijing 100190, China; University of Chinese Academy of Sciences, Beijing 100049, China, Xinjiang Key Laboratory of Minority Speech and Language Information Processing, Xinjiang Technical Institute of Physics and Chemistry, Chinese Academy of Sciences, Urumqi 830011, China. Received: 16 October 2019; Accepted: 18 November 2019; Published: 22 November 2019.

[15] Farah Benamara, Cyril Grouin, Jihen Karoui, Véronique Moriceau, Isabelle Robba (Eds.). **Actes de l'atelier « Défi Fouille de Textes (DEFT 2017)**. 24e Conférence sur le Traitement Automatique des Langues Naturelles (TALN). Orléans, France – 26-30 juin 2017.

[16] Tom Bourgeade. **Représentation sémantique et structurelle de conversations par chat** . Université Paul Sabatier Institut de Recherche en Informatique de Toulouse.27 août 2018.

[17] Rui Zhu, Delu Yang, and Yang Li. **Learning Improved Semantic Representations with Tree-Structured LSTM for Hashtag Recommendation: An Experimental Study**. College of Information and Computer Engineering, Northeast Forestry University, Harbin 150040, China; Received: 20 January 2019; Accepted: 3 April 2019; Published: 6 April 2019.

[18] Feng Zhou, X. Jessie Yang, , and Xin Zhang. **Takeover Transition in Autonomous Vehicles: A YouTube Study**. Department of Industrial and Manufacturing Systems Engineering, University of Michigan, Dearborn, Michigan, USA; Department of Industrial and

Operations Engineering, University of Michigan, Ann Arbor, Michigan, USA; Ross School of Business, University of Michigan, Ann Arbor, Michigan, USA. Published online: 01 Jul 2019.

[19] Hasato Hagiwara. **Improving a Sentiment Analyzer using ELMo — Word Embeddings on Steroids.** Posted on Sat 27 October 2018 in Sentiment Analysis. URL: http://www.realworldnlpbook.com/blog/improving-sentiment-analyzer-using-elmo.html(visité le: 09-05-2020).

[20] Julien Lausson. **Google BERT : comprendre ce qui va changer sur le moteur de recherche en 5 questions** , 09 décembre 2019 - Tech. URL: https://www.numerama.com/tech/578245-google-bert-comprendre-ce-qui-va-changer-sur-le-moteur-de-recherche-en-5-questions.html (visité le:11-05-2020)

[21] Pirmin LEMBERGER, Thomas SCIALOM . **DEEP TRANSFER LEARNING – LE TRAITEMENT DU LANGAGE À L'AUBE D'UNE RÉVOLUTION ?**. IA LAB 22 novembre 2018.

[22] Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova . **BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding** . Google AI Language – 24 May 2019.

[23] **BBC Datasets**. URL: http://mlg.ucd.ie/datasets/bbc.html. (visité le:20-07-2020)

[24] **BBC NEWS**. URL: https://www.kaggle.com/c/learn-ai-bbc. (visité le:28-07-2020)

[25] Chris McCormick. **BERT Word Embeddings Tutorial** .14 May 2019. URL: https://mccormickml.com/2019/05/14/BERT-word-embeddings-tutorial/. (visité le:22-08-2020)

[26] Chi Sun, Xipeng Qiu_, Yige Xu, Xuanjing Huang. **«How to Fine-Tune BERT for Text Classification?»,** Shanghai Key Laboratory of Intelligent Information Processing, Fudan University School of Computer Science, Fudan University 825 Zhangheng Road, Shanghai, China. 5 Feb 2020.

[27] **How to find the cosine similarity between tow vectors in python.**
URL: https://www.kite.com/python/answers/how-to-find-the-cosine-similarity-between-two-vectors-inpython (visité le:22-08-2020)

[28] Mikolov et al., 2013

[29] Mikolov et al., 2013a

[30] Mikolov et al., 2013c

[31] Pennington et al., 2014

[32] Bojanowski et al., 2017

[33] Peters, et al, 2018

[34] Devlin et al., 2018

[35] **BERT de Google AI sur le banc de test !.** URL: https://www.quantmetry.com/bert-google-ai-banc-de-test/ (visité le:22-08-2020)

[36] **ILLUSTRATION DE BERT.** URL: https://lbourdois.github.io/blog/nlp/BERT/ (visité le:22-08-2020)

[37] Paul Denoyes. **BERT : Faire comprendre le langage naturel à une machine, en pré-entraînant des Transformers bi-directionnels profonds**. 10/04/2019. URL: https://lesdieuxducode.com/blog/2019/4/bert--le-transformer-model-qui-sentraine-et-qui-represente. (visité le:22-08-2020)

[38] Julien Lausson. **Google BERT : comprendre ce qui va changer sur le moteur de recherche en 5 questions**. 09 décembre 2019. URL: https://www.numerama.com/tech/578245-google-bert-comprendre-ce-qui-va-changer-sur-le-moteur-de-recherche-en-5-questions.html. (visité le:22-08-2020)

[39] Iz Beltagy, Matthew E.Peters, Arman Cohan. **Longformer: The Long-Document Transformer.Allen Institute for Artificial Intelligence**, Seattle, WA, USA. 5 Feb 2020.

[40] URL: https://www.audit-conseil-formation.com/comprendre_python.html (visité le:12-09-2020)

[41] URL: https://data-flair.training/blogs/numpy-features/ (visité le:12-09-2020)

[42] https://data-flair.training/blogs/numpy-features/ (visité le:12-09-2020)

[43] URL: https://github.com/reddyprasade/Matplotlib (visité le:12-09-2020)

[44] Mikolov et al., 2013b, 2010