Université de Paris Nanterre

Mémoire de M2 Traitement Automatique des Langues
Parcours Recherche et Développement

Mesure de similarité entre textes courts - Utilisation de BERT pour un système de recommandation

---

Présenté par :
Alican YÜKSEL

Encadrant :
Damien NOUVEL

Année universitaire 2019-2020

Déclaration sur l'honneur de non-plagiat

Je soussigné Alican Yüksel, déclare avoir rédigé ce travail sans aides extérieures ni sources autres que celles qui sont citées. Toutes les utilisations de textes préexistants, publiés ou non, y compris en version électronique, sont signalées comme telles. Ce travail n'a été soumis à aucun autre jury d'examen sous une forme identique ou similaire, que ce soit en France ou à l'étranger, à l'université ou dans une autre institution, par moi-même ou par autrui.

Date : 22 août 2020

Signature :

Remerciements

Je tiens d'abord à remercier M. Damien NOUVEL de m'avoir encadré tout au long de ce mémoire de recherche, pour sa bienveillance et ses conseils précieux, de m'avoir fait confiance tout au long de ce projet.

Ensuite, je tiens à remercier ma maître de stage, Mme. Ghizlane ECHBARTHI, de m'avoir accompagné tout au long de ce projet, pour son soutien quotidien, ses conseils et sa positivité.

Je n'oublie pas tous mes collègues de Yoomap qui m'ont chaleureusement accueilli, avec qui j'ai passé 6 mois de stage dans une bonne atmosphère du début à la fin.

Je tiens également à remercier toute l'équipe pédagogique du master pluriTAL, pour ces deux années de master, riche en apprentissage, et pour les enseignements qui m'ont beaucoup apporté.

Je remercie aussi, M. Mehmet Ali AKINCI, professeur des universités, de m'avoir fait confiance tout au long de mon parcours universitaire.

Je remercie de plus, mon ami, Alican Kocabıyıkoğlu qui m'a donné le goût du TAL quand j'étais en master de Sciences du Langage. Il a su me donner de précieux conseils et m'a été d'un grand soutien tout au long de la réalisation de ce mémoire.

Je remercie également, mes amis proches, Muhammed Topcu, Cihat Yigit, Yusuf Tavukçu, Kutluhan Ünal qui étaient toujours là pour moi, malgré la distance qui nous sépare, l'amitié est toujours présente.

Je remercie encore, mon oncle qui était toujours à mes côtés, pour son accompagnement et ses précieux conseils, sans qui je n'aurais pas pu trouver un toit en France.

J'aimerais remercier plus particulièrement ma mère, ma sœur, mon père, de m'avoir donné l'opportunité de faire mes études à l'étranger, pour leurs soutiens tout au long de mon parcours

universitaire quand j'en avais besoin, sans qui ce mémoire de recherche me tient à cœur n'aurait pas pu être rédigé.

Je remercie enfin Lucie Coulette et toute sa famille, qui m'ont donné un foyer loin de chez moi, grâce à vous je me suis senti chez moi.

Table des matières

Table des figures

Résumé

La mesure de similarité entre textes courts est considérée difficile à cause du fait que deux phrases sémantiquement liées peuvent ne contenir aucun mot en commun. Cela signifie que les mesures standards de similarité textuelle, qui sont basées sur la cooccurrence des mots et conçues pour fonctionner pour les *documents*, ne sont pas appropriées. Ce mémoire présente la mise en place d'un système de recommandation basé sur la similarité entre textes courts au sein de l'entreprise Yoomap. Pour ce faire, nous nous sommes intéressés à la constitution d'un corpus de référence avec des paires de phrases afin de réaliser le *fine-tuning* d'un modèle pré-entraîné de Transformers (BERT). L'objectif majeur de cette étude est d'observer la pertinence du système d'attention de Transformers pour détecter la similarité entre textes même s'ils sont courts. Les motivations de ce sujet sont multiples puisqu'elles s'inscrivent dans des problématiques actuelles du TAL telles que la mesure de similarité entre textes ou le développement des systèmes de recommandation qui nécessite de comprendre la sémantique des mots. Les expériences sur corpus constitué ont permis de tester la performance de plusieurs modèles, d'ajuster leurs paramètres, et ont montré l'intérêt de ces modèles pour la similarité, mais aussi leur coût du point de vue computationnel. Le système d'attention de Transformers, malgré tout, apporte des perspectives et des nouveautés dans le Traitement Automatique des Langues.

Mots-clés : traitement automatique des langues, similarité textuelle, classification de textes, BERT, fine-tuning.

Introduction

Ce mémoire de recherche, accompagné d'un stage de 6 mois au sein de l'entreprise Yoomap, s'inscrit dans le cadre de mon mémoire de fin d'études du Master Traitement Automatique des Langues, parcours Recherche et Développement. Ce mémoire a été encadré par M. Damien NOUVEL, maître de conférences à l'Inalco. Le stage a été supervisé par Mme. Ghizlane ECHBARTHI.

Yoomap est une startup fondée en 2013 qui développe des solutions et des logiciels en ayant pour but de faire le lien entre les grands groupes et les startups dans le domaine de l'innovation. Grâce à leurs logiciels, les entreprises peuvent travailler de manière plus ouverte et plus transverse en optimisant l'innovation au sein de leur propre organisation.

L'objectif de ce travail est de mesurer la similarité entre des textes courts pour un système de recommandation. La similarité textuelle est utilisée depuis très longtemps dans plusieurs domaines du TAL comme la recherche d'information [6], la classification des textes [7], l'analyse sémantique [8], le système question-réponse [9].

La détection de la similarité textuelle a été introduite par les modèles basés sur les chaînes de caractères. Ces chaînes de caractères sont des mots, des lexèmes qui sont repérés par segmentation du texte en mot et éventuellement analyse linguistique : racinisation, lemmatisation, recherche d'entités etc. Ensuite, les modèles basés sur les ressources externes ont été proposés comme Sac des Mots (en anglais, *Bag of Words*), Analyse sémantique latente (LSA, en anglais, *Latent Semantic Analysis*), WordNet afin de mieux prendre en compte le contexte sémantique. Les travaux récents s'intéressent de plus en plus aux modèles vectorielles dans un espace sémantique calculé par utilisation de réseaux de neurones.

Concernant la difficulté principale pour la similarité textuelle, l'hypothèse est que plus deux textes sont similaires, plus ils ont tendance à avoir des mots en communs. Bien que cette hypothèse paraisse raisonnable, elle n'est pas valable en réalité pour les textes courts (par exemple : les phrases, les tweets, les SMS etc.) ni pour les textes longs (par exemple : les documents) car deux textes peuvent être sémantiquement similaires bien qu'ils aient peu de mots, voire aucun, en commun. De ce fait, la précision (en anglais, *accuracy*) est généralement basse quand il s'agit des textes courts [10].

Les travaux existants quant à cette problématique sont aujourd'hui assez nombreux. Les modèles traditionnels comme, par exemple, l'approche **Sac de Mots** ne donnent pas de résultats pertinents à défaut de l'absence de la prise en compte de la sémantique des mots. En plus du contexte sémantique, appliquer cette approche sur des textes courts ne fourniront pas non plus assez d'informations sur la co-occurrence des termes [11]. Suite à ce besoin, les recherches se sont orientées sur les modèles qui privilégient plus des représentations sémantiques, par exemple en utilisant des plongements de mots traditionnels (Word2Vec) ou contextualisés (BERT) [4].

Concernant les systèmes de recommandation (RS, en anglais, *recommender system*), ils sont généralement classés en trois catégories : les approches basées sur le contenu, celles sur le filtrage collaboratif et les approches hybrides [12][13]. Ils ont pour but de collecter des informations sur les préférences des utilisateurs pour un ensemble de corpus (par exemple, des films, des chansons, des livres, des applications, des sites Web, etc.). Les informations peuvent être acquises explicitement (généralement en collectant le retour des utilisateurs comme par exemple leurs notes etc.) ou implicitement en surveillant le comportement des utilisateurs [14]. Dans notre travail, il s'agira d'un système de recommandation basé sur la similarité textuelle donc sur le contenu textuel.

Dans un premier temps, nous verrons un panorama des travaux existants sur la similarité textuelle. Dans un second temps, nous présenterons notre corpus de travail, sa structure et la façon dont nous l'utilisons. Un troisième temps sera consacré à l'explication de notre méthodologie, de nos choix de l'algorithme, de la préparation des données pour un apprentissage supervisé. Enfin, nous discuterons les résultats obtenus.

Chapitre 1

État de l'art

1.1 Recherche d'information : généralités

La recherche d'information (en anglais, *information retrieval*) est une branche de l'informatique qui s'intéresse à l'acquisition, l'organisation, le stockage, la recherche et la sélection d'information [15].

Les recherches existantes en recherche d'information se sont focalisées plus sur le langage naturel des textes, étant donné l'importance et le volume important des données textuelles sur Internet ou dans des archives. Il peut y avoir bien évidemment d'autres types de données comme par exemple les images, les fichiers audio ou encore les vidéos [16].

Il existe deux grandes catégories de méthodes en recherche d'information : sémantique et statistique. Les approches sémantiques essaient de mettre l'accent sur l'analyse syntaxique et sémantique. Autrement dit, leur objectif est de mieux comprendre le langage naturel de l'humain. Quant à celles statistiques, nous pouvons parler des approches booléennes, vectorielles et probabilistes. Les approches statistiques visent à découper les documents en *termes*. Ces termes sont souvent des mots. L'objectif est de les traiter avec des calculs statistiques.

Aujourd'hui, les approches sémantiques utilisent aussi ou sont souvent en interaction avec les notions statistiques. Durant les années 1960, la recherche d'information a com-

mencé à devenir un domaine scientifique principalement grâce aux contributions de Gérard Salton avec le modèle vectoriel et le système SMART [17] [18]. Son approche vectorielle a donc permis d'introduire la notion de mesure de similarité, notamment la fameuse mesure du cosinus entre vecteurs [18].

1.2 Notion de la similarité : généralités

Estimer la similarité entre deux ou plusieurs entités est un aspect central de la cognition. La similarité joue un rôle fondamental dans les théories de la connaissance et du comportement. C'est un principe organisateur par lequel les individus classent les objets, forment des concepts et catégories et généralisent [19]. La notion de similarité est un sujet très discuté dans la linguistique, la philosophie, la recherche d'information et l'intelligence artificielle. [20].

Du point de vue linguistique, les mots peuvent être similaires lexicalement ou sémantiquement. Si deux mots possèdent des caractères similaires, ils sont similaires lexicalement. Par conséquent, ils sont similaires sémantiquement s'ils portent un sens similaire [21].

Concernant la similarité lexicale, il est nécessaire de souligner que la similarité trouvée entre deux séquences de caractères (des mots) ne prennent pas en compte le contexte ni l'information sémantique. Prenons l'exemple suivant :

— **Phrase 1 :** Le roi tue le peuple.

— **Phrase 2 :** Le peuple tue le roi.

Pour un système qui se base sur la similarité lexicale, il risque de trouver ces deux énoncés similaires alors qu'ils ne signifient pas le même sens sémantiquement. La similarité lexicale a été introduite par les approches basées sur la chaîne de caractère (en anglais, *String-Based*) [21]. Quant à la similarité sémantique, elle a été introduite par les approches basées sur les ressources externes ou sur le corpus, dites (en anglais, *Corpus-Based, Knowledge-Based* que nous expliquerons ultérieurement.

1.3 Similarité basée sur la chaîne de caractère

Dans cette approche, la phrase est considérée comme un ensemble des chaînes de caractère [22].

Il existe des méthodes différentes pour mesurer la similarité basée sur la chaîne de caractère comme *la distance de Levenshtein*, *Q-gram*, *Cosinus*, *Jaccard*.

1.3.1 Distance de Levenshtein

La distance de Levenshtein (en anglais *Distance Levenshtein*) entre deux chaînes de caractère est le nombre minimal d'insertion, de suppressions et de substitutions nécessaires pour transformer une chaîne en l'autre [23].

Prenons les mots "*maison*" et "*maman*" comme exemple : la distance entre ces deux mots est 4. Tout simplement, pour que le mot "maison" puisse se transformer en mot "maman", il faudrait trois opérations : délétion, insertion, substitution.

Pour la transformation de la chaîne de caractère "**maison**" en "**maman**", les étapes sont les suivantes :

1. Le 3ème caractère "I" devient "M". Donc, ma**i**son → ma**m**son
2. Le 4ème caractère "S" devient "A". Donc mam**s**on → mam**a**on
3. Le 5ème caractère "O" doiv être supprimé. Donc mama**o**n → maman

Comme nous avons pu le voir dans les étapes ci-dessus, il s'agit de trois changements de caractère. La distance sera donc 3. Plus la distance est grande, plus les deux chaînes de caractères sont différentes.

1.3.2 Distance de Q-gram

La distance Q-gram est calculé par le biais de *N-gram* [24][25]. L'objectif est d'observer combien y a-t-il de *N-gram* commun dans les deux chaînes de caractères à comparer [26]. Ce concept remonte à Shannon (1948) [27] et il a été développé par Ukkonen (1992) [26].

Prenons les mots "abcde" et "abdcde", sachant que le *N-gram* est égal à 2, nous obtiendrons le tableau suivant :

	ab	bc	cd	de	dc	bd
Ch 1	**1**	1	**1**	**1**	0	0
Ch 2	**1**	0	**1**	**1**	1	1

Tableau 1 – Exemple des *bigrams* communs

Comme nous pouvons voir les *bigrams* communs dans le tableau [1], la distance entre "abcde" et "abdcde" est donc égale à 3 bigrams communs. Selon cette approche, plus la distance est grande, plus les deux chaînes de caractères se ressemblent.

1.3.3 Distance cosinus

La distance cosinus est obtenue en mesurant l'angle cosinus entre deux vecteurs projetés sur un espace multidimensionnel. Elle s'applique donc pas à des chaînes. Le score se variera entre 0 et 1. Plus la distance s'approche à 1, plus les deux vecteurs sont similaires. La formule cosinus est montrée dans l'équation [1]. Soit deux vecteurs A et B, l'angle θ s'obtient par le produit scalaire et la norme des vecteurs.

$$\cos(\theta) = \frac{A \cdot B}{||A|| \cdot ||B||} \tag{1}$$

Elle peut être utilisée dans le but d'obtenir la similarité entre deux mots mais il serait également possible de l'utiliser avec les *plongement des mots* pour une phrase ou un document. Nous l'expliquerons plus en détail dans les chapitres suivants.

1.3.4 Distance de Jaccard

L'indice de Jaccard [28] est une des méthodes la plus populaire et utilisée pour le calcul de similarité. Elle s'appuie sur les ensembles contrairement aux vecteurs que nous avons pu voir dans la section 1.3.3.

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{2}$$

L'indice dont la formule montrée dans l'équation [2] est obtenu par la différence symétrique et le nombre d'éléments de l'union de ces deux ensembles.

$$J_\delta(A,B) = 1 - J(A,B) = 1 - \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cup B| - |A \cap B|}{|A \cup B|} \tag{3}$$

La distance de Jaccard dont la formule est montrée dans l'équation [3] est obtenue par la soustraction de l'indice de Jaccard à 1 puisque quand les deux éléments sont similaires, la similarité va augmenter mais la distance devrait diminuer. En ce sens, la distance est l'opposé de la similitude et c'est pourquoi la distance de Jaccard est obtenue en soustrayant l'indice de Jaccard à 1.

1.4 Représentation des documents dans un corpus

1.4.1 Approche Sac de Mots

L'approche Sac de Mots (en anglais, *Bag of Words*) est utilisée pour plusieurs domaines comme la vision par ordinateur, le traitement automatique des langues, la recherche d'information [29].

Il s'agit d'une vectorisation du contenu textuel. Pour ce faire, l'encodage *one-hot* est utilisé (en français, vecteur binaire). Dans cet encodage, chaque document est représenté sous forme d'un vecteur de taille n, qui est la taille du vocabulaire du document. Le modèle prend en entrée une donnée textuelle et il la découpe en mots. Ensuite, il construit un vocabulaire avec les mots *uniques* sans prendre en compte les mots doublons, ni l'ordre des mots. Les vecteurs contiennent principalement des zéros et sont aussi appelés "creux" (en anglais, *sparse*) sauf lorsqu'un mot apparaît, le modèle compte l'occurrence du mot.

Phrase 1	L'exemple est très difficile à comprendre
Phrase 2	Mathias a du mal à comprendre le problème difficile

Tableau 2 – Exemple de deux phrases

En prenant les deux exemples montrés dans le tableau [2], il est possible de construire un vocabulaire qui ne contient pas de doublons ni de mots vides (en anglais, *stop-words*).

Vocabulaire complet
exemple, est, très, difficile, comprendre, Mathias, a, mal, problème

Tableau 3 – Exemple du vocabulaire complet

Comme nous pouvons le voir dans le tableau [3], le vocabulaire complet possèdent 9 mots uniques. Afin de pouvoir vectoriser une de ces phrases, l'objectif est d'indiquer la présence du mot avec la représentation booléenne c'est-à-dire le "1" pour la présence et le "0" pour l'absence.

L'exemple est très difficile à comprendre	**[1,1,1,1,1,0,0,0,0]**
Mathias a du mal à comprendre le problème difficile	**[0,0,0,1,1,1,1,1,1]**

Tableau 4 – Exemple de vectorisation des phrases

Comme nous pouvons le voir dans le tableau [4], les deux phrases sont présentées par les vecteurs ayant la taille n qui est celle du vocabulaire.

1.4.2 Approche TF-IDF

La représentation en TF-IDF [15] est une technique de pondération qui permet de mettre l'accent sur les termes *discriminants* en fonction du document dans lequel ils se trouvent. L'objectif de cette technique est de représenter les données textuelles en vecteur. **TF** signifique la fréquence du terme (en anglais, *term frequency*) et **IDF** signifie la fréquence inverse du document (en anglais, *inverse document frequency*).

$$TF_{x,y} = \frac{\text{Nombre d'occurrences de } \mathbf{x} \text{ dans le document } \mathbf{y}}{\text{Nombre de termes dans } \mathbf{y}} \tag{4}$$

$$IDF(x) = \log(\frac{\text{Nombre de documents}}{\text{Documents contenant } \mathbf{x}}) \tag{5}$$

Afin de pouvoir mieux expliquer cette approche, il sera utile de la comparer avec celle de *Sac de mots* que nous avons pu voir dans la section 1.4.1. Prenons les trois exemples typiques [1] :

1. This movie is very scary and long
2. This movie is not scary and is slow
3. This movie is spooky and good

	1 This	**2 movie**	**3 is**	**4 very**	**5 scary**	**6 and**	**7 long**	**8 not**	**9 slow**	**10 spooky**	**11 good**	**Longueur de la phrase**
Phrase 1	1	1	1	1	1	1	1	0	0	0	0	7
Phrase 2	1	1	2	0	0	1	1	0	1	0	0	8
Phrase 3	1	1	1	0	0	0	1	0	0	1	1	6

Tableau 5 – Exemple des représentations vectorielles avec l'approche Sac de Mots

Avec l'approche Sac de Mots, les trois phrases seront présentées comme nous pouvons le voir dans le tableau [5].

Quant au TF-IDF, comme indiqué juste au-dessus, l'objectif est de mettre l'accent sur les termes importants. Pour concrétiser l'exemple, prenons la deuxième phrase qui est *This movie is not scary and is slow* :

Pour le même vocabulaire, nous allons suivre les étapes suivantes pour calculer le **TF** :

— Le nombre de termes dans le document : 8

— Pour le mot " *This*", le calcul sera le suivant : (combien de fois apparait-il le terme *This*) / (le nombre de tous les termes dans le document) = 1/8

En appliquant la formule de TF [4] pour tous les termes, nous allons obtenir le tableau suivant :

1. L'exemple est inspiré de ce lien. Le lien a été consulté le 15/05/2020

Mots	Sac de Mots Phrase 1	Sac de mots Phrase 2	Sac de mots Phrase 3	TF Phrase 1	TF Phrase 2	TF Phrase 3
This	1	1	1	1/7	1/8	1/6
movie	1	1	1	1/7	1/8	1/6
is	1	2	1	1/7	1/4	1/6
very	1	0	0	1/7	0	0
scary	1	1	0	1/7	1/8	0
and	1	1	1	1/7	1/8	1/6
long	1	0	0	1/7	0	0
not	0	1	0	0	1/8	0
slow	0	1	0	0	1/8	0
spooky	0	0	1	0	0	1/6
good	0	0	1	0	0	1/6

Tableau 6 – Exemple montrant la différence des représentations vectorielles entre l'approche Sac de Mots et le calcul de TF

Quant au **IDF**, le calcul sera le suivant :

— Pour le mot " *This*" : log(nombre des documents / nombre des documents contenant le terme *This*) : log(3/3) = log(1) = 0

Après avoir appliqué la formule d'**IDF** [5] pour tous les termes, nous allons calculer la valeur TF-IDF pour chaque terme. Pour ce faire, il suffira de multiplier les valeurs de **TF** et celles d'**IDF**.

Mots	TF Phrase 1	TF Phrase 2	TF Phrase 3	IDF	TF-IDF Phrase 1	TF-IDF Phrase 2	TF-IDF Phrase 3
This	1/7	1/8	1/6	0	0	0	0
movie	1/7	1/8	1/6	0	0	0	0
is	1/7	1/4	1/6	0	0	0	0
very	1/7	0	0	0.48	0.068	0	0
scary	1/7	1/8	0	0.18	0.025	0.022	0
and	1/7	1/8	1/6	0	0	0	0
long	1/7	0	0	0.48	0.068	0	0
not	0	1/8	0	0.48	0	0.060	0
slow	0	1/8	0	0.48	0	0.060	0
spooky	0	0	1/6	0.48	0	0	0.080
good	0	0	1/6	0.48	0	0	0.080

Tableau 7 – Exemple montrant les représentations vectorielles de TF-IDF

Comme nous pouvons le voir dans le tableau [7], le TF-IDF nous a permis de cibler les mots importants pour un document donné. Le calcul TF vise à mettre un poids en observant la fréquence du terme. Par conséquent, le calcul IDF vise à donner un poids plus important aux termes. Les poids TF-IDF sont obtenus par la multiplication de ces deux valeurs différentes. Comme nous pouvons le voir dans le tableau, les mots qui sont présents dans tous les documents n'ont pas de poids. Cela permet également de repérer les mots vides comme, dans l'exemple ci-dessus, *is* et *and*. C'est la raison pour laquelle que les mots *very*, *scary*, *long*, *not*, *slow*, *spooky*, *good* ont eu des poids plus importants que d'autres.

1.4.3 Approche Analyse sémantique latente

L'approche Analyse Sémantique Latente (LSA, en anglais *Latent Semantic Analysis*) est basée sur l'intuition que les mots apparaissant dans des contextes similaires ont tendance à avoir des significations similaires [30]. Dans un premier temps, la LSA pro-

duit des matrices de la cooccurence des mots. Dans un deuxième temps, l'algorithme d'analyse sémantique latente passera par un procédé qui est la **décomposition en valeurs singulières** (SVD, en anglais *Singular Value Decomposition*) que nous expliquerons ultérieurement.

c1	*Human* machine *interface* for ABC *computer* applications
c2	A *survey* of *user* opinion of *computer system response time*
c3	The *EPS user interface* management *system*
c4	*System* and *human system* engineering testing of *EPS*
c5	Relation of *user* perceived *response time* to error measurement
m1	The generation of random, binary, ordered *trees*
m2	The intersection *graph* of paths in *trees*
m3	*Graph minors* IV : Widths of *trees* and well-quasi-ordering
m4	*Graph minors* : A *survey*

Tableau 8 – Exemple de "Titles of Some Technical Memos" (Deerwester et al., 1990)[30]

La LSA est un concept mathématique. Elle n'utilise pas de dictionnaires ni de bases de connaissances linguistiques. Elle prend en entrée une donnée textuelle et la découpe en mot.

Les exemples habituels de Deerwester sont montrés dans le tableau [8]. Les mots apparaissant plus d'une fois dans les documents différents ont été sélectionnés et mis en italique. Ceci permet d'extraire les *mot importants* selon l'approche. La première étape consistera à représenter le texte comme une matrice dans laquelle chaque ligne représente un mot unique et chaque colonne représente un *document*. L'exemple de la matrice est montré dans le tableau [9].

	c1	c2	c3	c4	c5	m1	m2	m3	m4
human	1	0	0	1	0	0	0	0	0
interface	1	0	1	0	0	0	0	0	0
computer	1	1	0	0	0	0	0	0	0
user	0	1	1	0	1	0	0	0	0
system	0	1	1	2	0	0	0	0	0
response	0	1	0	0	1	0	0	0	0
time	0	1	0	0	1	0	0	0	0
EPS	0	0	1	1	0	0	0	0	0
survey	0	1	0	0	0	0	0	0	1
trees	0	0	0	0	0	1	1	1	0
graph	0	0	0	0	0	0	1	1	1
minors	0	0	0	0	0	0	0	1	1

Tableau 9 – Exemple de matrice (Deerwester et al., 1990)[30]

La technique de la décomposition en valeurs singulières (SVD) consiste à réduire la taille de la matrice obtenue et aussi à condenser l'information présente dans cette matrice [31].

$$M = U\Sigma V^* \tag{6}$$

Comme nous pouvons le voir dans l'équation [6], la valeur SVD est obtenue par la factorisation des matrices. La matrice **U** et **V** sont orthogonales. La matrice $\mathbf{\Sigma}$ sera diagonale et ne pourrait pas avoir des valeurs négatives. La valeur SVD correspondra donc à celle-ci car dans la matrice $\mathbf{\Sigma}$, il n'y aura que des valeurs singulières. Celle-ci pourrait donc être utilisée pour mesurer la similarité cosinus [32].

1.5 Similarité basée sur les ressources externes

1.5.1 WordNet

WordNet est une base de données lexicale en ligne dont la conception est inspirée des théories psycholinguistiques de la mémoire lexicale humaine [1].

La similarité est calculée par utilisation de la structure de ce réseau sous forme de graphe, les nœuds étant des mots / termes et les relations entre ces mots sont sémantiques (synonymie, hyponymie, etc.) [33]. La structure de Wordnet est montrée dans la figure [1]. En ce qui concerne la terminologie, les groupes de mots synonymes sont appelés sous le nom de *synset* dans WordNet.

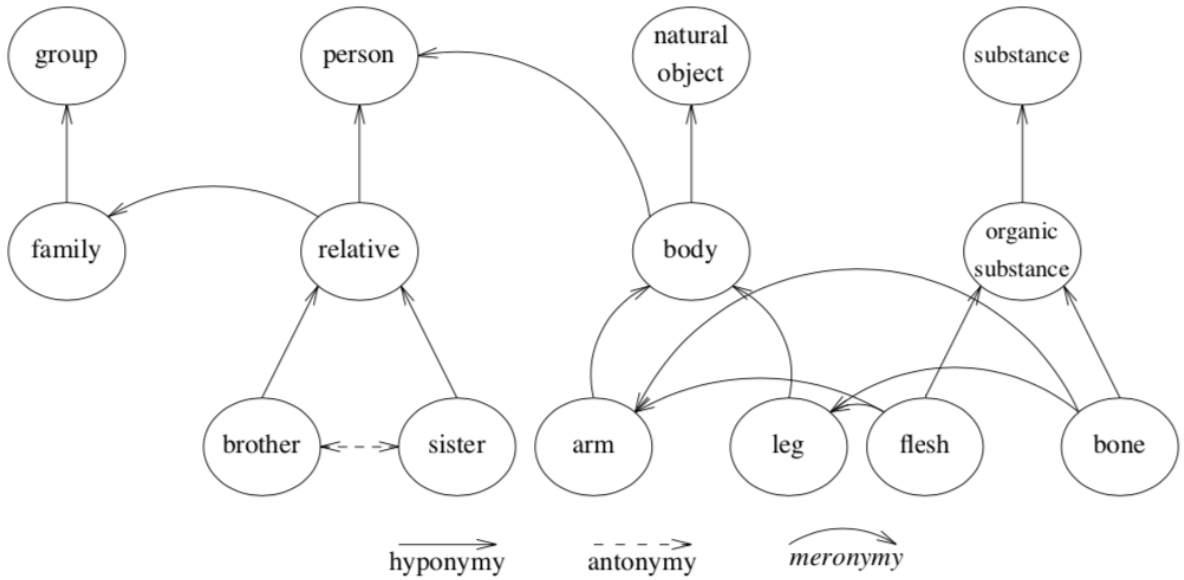


FIGURE 1 – Représentation de la structure sémantique de WordNet dans la langue anglaise [1].

Parmi les méthodes de la relation sémantique et ainsi le calcul de la similarité, Resnik [34], Lin [35] et Jiang et Conrath[36] se sont basés sur le contenu de l'information. Les méthodes de Leacock et Chodorow [37], Wu et Palmer [38] sont basées sur la distance entre les branches du treillis. Panchenko [39] a montré que la méthode Resnik a les meilleures performances parmi les mesures basées sur la connaissance.

1.5.2 Sémantique par le biais du Web

Considérer le Web comme un corpus de texte est devenu récemment un sujet de recherche important [40]. Pour la détection de la similarité sémantique, Matsuo et al. [41] propose donc d'utiliser le Web avec l'objectif de mesurer la similarité entre des noms de personnes. L'objectif, pour lui, est de mesurer l'association entre deux noms de personnes à l'aide du coefficient de chevauchement, calculé en fonction du nombre de pages Web visitées pour chaque nom [40].

Sahami et al. [42] propose d'utiliser des *snippets* pour pouvoir mesurer la similarité sémantique entre deux requêtes envoyées aux moteurs de recherche. Un snippet est une petite fenêtre qui contient la donnée textuelle extraite par le moteur de recherche pour une requête donnée. Cette approche dépend donc du moteur de recherche utilisé.

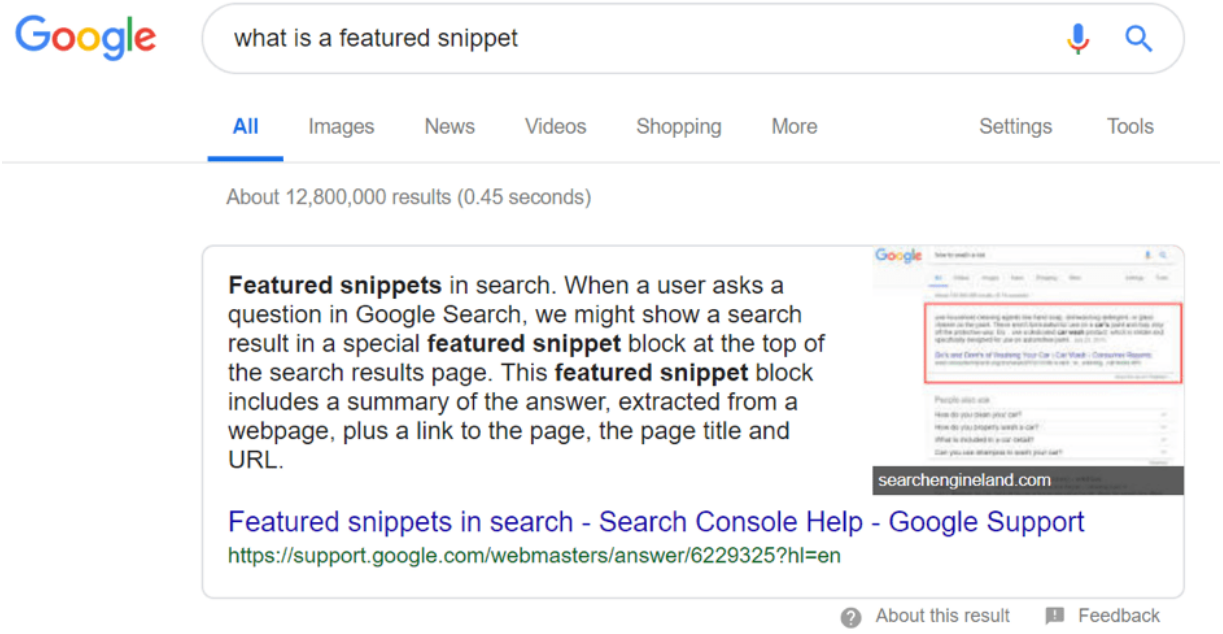


FIGURE 2 – Exemple d'un snippet retourné par le moteur de recherche Google pour une requête donnée.

L'objectif est de collecter les snippets retournés par le moteur de recherche pour chaque requête. Pour la représentation de ces données textuelles, l'auteur préfère les vectoriser en appliquant l'approche **TF-IDF**. Ensuite, les vecteurs sont normalisés et leurs centroïdes sont calculés. La similarité entre deux vecteurs est définie par le calcul du produit scalaire

entre deux centroïdes. L'auteur ne passe pas par un système taxonomique comme WordNet ou la mesure de la similarité cosinus.

Chen et al. [43] propose un système double contenant les deux approches citées juste au dessus, celle pour compter le nombre de pages pour une requête donnée et celle pour extraire le snippet. Pour deux mots **P** et **Q**, ils collectent des snippets pour chaque mot à partir d'un moteur de recherche Web. Ils comptent le nombre d'occurrences du mot **P** parmi les n premiers snippets pour le mot **Q** et le nombre d'occurrences du mot **Q** parmi les n premiers extraits pour le mot **P** [40].

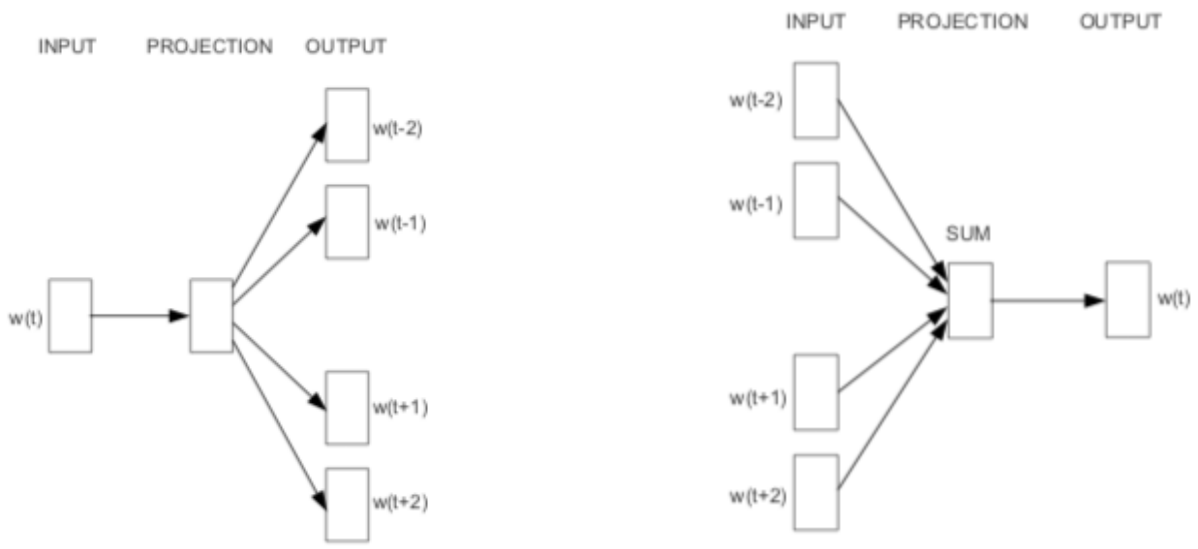
1.6 Modèles des réseaux neuronaux

1.6.1 Word2vec

Word2Vec est un des modèles de réseaux de neurones à deux couches qui sont utilisés pour produire des plongements de mots (en anglais, *word embeddings*). Il existe deux modèles pour word2Vec. Ce sont : sac de mots continu (CBOW en anglais, *Continuous Bag of Words*) et Skip-gram.

Le premier travail dans lequel l'évaluation de plongement des mots a été abordée (cependant, il n'y avait pas de plongement des mots, donc les concepts similaires étaient appelés "*Distribution semantics*") a été réalisé dans [Griffiths et al., 2007][44], avant même que la sémantique distributionnelle ne soit devenue populaire [45].

Mikolov et al.[2] introduit les modèles *Skip-gram* et *Continuous Bag of Words* qui sont deux méthodes efficaces pour mieux obtenir des représentations vectorielles de mots à partir de grandes quantités de données textuelles non structurées.



(a) Skip-gram.

(b) CBOW.

FIGURE 3 – Différence entre deux modèles [2].

La différence entre ces deux méthodes se joue seulement sur l'entrée et la sortie. Dans le modèle **CBOW**, comme nous pouvons le voir dans la figure [3b], nous prenons en entrée les mots voisins pour un mot donné. Les mots voisins sont choisis pour un N-gram donné. (dans la figure, le n-gram est 2). Ce nombre N permet de parcourir la phrase. L'objectif est de regarder les mots voisins pour deviner le mot centré. Comme nous pouvons le voir dans l'équation 7, le **CBOW** nous permet d'obtenir la probabilité d'avoir le mot w_i en prenant en compte le contexte dans lequel il apparaît. Ceci est important du point de vue sémantique.

$$P(w_i|w_{i-c}, w_{i-c+1}, ...w_{i-1}, w_{i+1}, ..., w_{i+c-1}, w_{i+c}) \tag{7}$$

Quant à l'approche de **Skip-gram**, le modèle prend un seul mot en entrée. Ensuite, en fonction du nombre de *N-gram* donné par l'utilisateur, les mots voisins sont prédits [figure 3a]. L'équation [8] est basée sur la probabilité d'avoir un tel contexte sachant que nous avons un tel mot :

$$P(w_{i-c}, w_{i-c+1}, ..., w_{i-1}, w_{i+1}, ..., w_{i+c-1}, w_{i+c}|w_i) \tag{8}$$

Ce processus est appliqué à toutes les séquences (phrases) du corpus fourni en entrée.

Une fois le modèle appris et la représentation par plongements calculée sur un corpus brut (non étiqueté), il est alors possible de l'utiliser comme espace mathématique pour représenter les mots. Cette technique s'appelle en anglais, *mapping*.

Afin de pouvoir visualiser le modèle Skip-gram, il est utile de montrer le processus de celui-ci. L'exemple est présenté dans la figure [4].

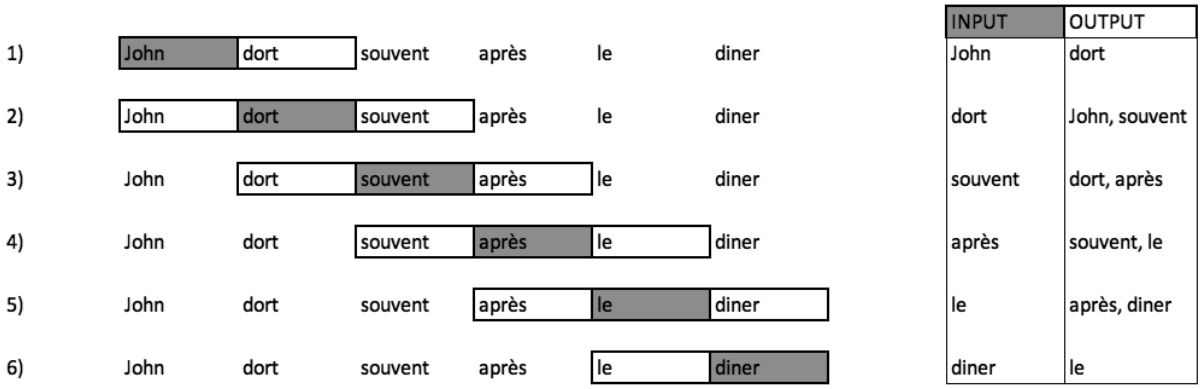


FIGURE 4 – Exemple de Skip-gram. Le paramètre de *n-gram* est 2.

1.6.2 Réseau de neurones siamois

Le modèle de réseau de neurones siamois (en anglais, *Siamese network*) a été introduit par [Bromley et al., 1993][46] pour la vérification des signatures. Celui-ci a été ensuite utilisé dans plusieurs domaines différents comme la reconnaissance faciale [47], la recherche d'image [48] et encore la similarité textuelle [5].

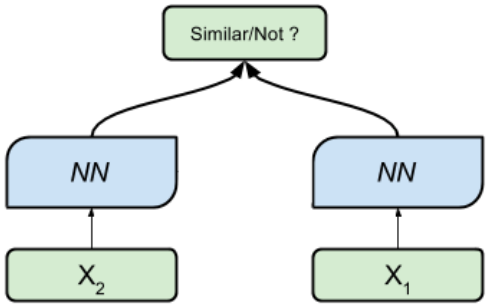


FIGURE 5 – Exemple de la structure de réseau de neurones siamois.

La structure du modèle de réseau de neurones siamois [2] est présentée dans la figure [5]. Comme nous pouvons le voir, le réseau siamois se compose de deux réseaux de neurones symétriques partageant les mêmes poids. Ils sont donc identiques. Cette architecture est conçue pour pouvoir prendre deux entrées différentes qui seront traitées séparément par le même réseau de neurones. L'objectif de celui-ci est de comparer les deux entrées pour savoir si elles sont similaires ou non [49]. Nous expliquerons en détail ultérieurement le modèle conçu pour la similarité textuelle.

1.6.3 Deep Structured Semantic Models (DSSM)

Le modèle *Deep Structured Semantic Models* (DSSM) est proposé pour l'extraction d'information [50]. La première couche du réseau sert à convertir le mot en une représentation vectorielle, connu sous le nom de *plongement de mots* comme nous l'avons vu avec le modèle **word2vec** dans la section [1.6]. Ils sont, soit entraînés séparément un par un dans un corpus, soit entraînés comme un ensemble avec d'autres paramètres du réseau [51] [52].

Le DSSM utilise trois types de réseaux de neurones dans le domaine de la similarité des données textuelles [53] qui sont *Réseau neuronal à propagation avant*, *Réseau neuronal convolutif*, *Le réseau neuronal récurrent avec LSTM*. Malgré les différents modèles proposés, il faudrait souligner que le DSSM conserve sa propre structure. En fonction du modèle du réseau neuronal, seules les couches d'apprentissage changeront.

1.6.3.1 Le DSSM basé sur un réseau neuronal à propagation avant

Le réseau neuronal à propagation avant (dites en anglais *Feed Forward Neural Network*) est une méthode très populaire [54]. Il existe deux modèles. Le premier appelé *Perceptron Simple* ne possède que deux couches, celle en entrée et celle en sortie. Le deuxième appelé *Perceptron Multicouche* contiendrait au moins trois couches essentielles qui sont *l'entrée*, *couches cachées* et *la sortie*.

2. Le lien a été consulté le 01/09/2020

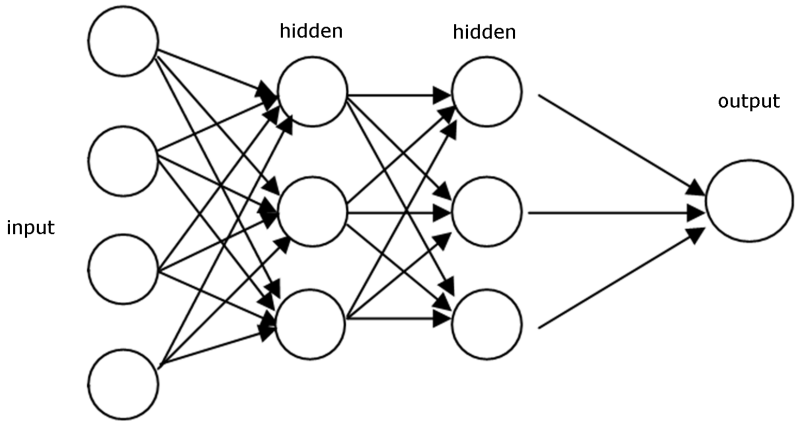


FIGURE 6 – Ex. Réseau neuronal à propagation avant avec deux couches cachées.

Dans ce modèle, le terme **propagation avant** vient du fait que l'information circule vers l'avant, c'est-à-dire, de l'entrée à la sortie sans avoir un retour en arrière contrairement au modèle du réseau de neurones récurrents, il n'y aura pas de répétition sur les neurones. Le modèle donne un score grâce à une fonction d'évaluation des erreurs (en anglais, *loss function*) en comparant les données initiales avec la sortie obtenue. Ensuite, le modèle change les poids des neurones afin d'obtenir un meilleur résultat grâce à l'algorithme de la rétropropagation (en anglais, *backpropagation algorithm*). Cet algorithme de la rétropropagation traverse le réseau de la sortie à l'entrée à chaque *epoch*.

Le modèle DSSM basé sur le réseau neuronal à propagation prend en entrée une donnée textuelle et la découpe en trigrammes de caractères. Ensuite, il la convertit en vecteur et ces vecteurs passent dans le réseau neuronal à propagation avant. Un vecteur sémantique est donc obtenu à la sortie pour chaque texte qui a été représenté sous forme de trigramme [53].

1.6.3.2 DSSM Convolutif (C-DSSM)

Le modèle DSSM est proposé avec une structure de **réseau neuronal convolutif** à cause de l'hypothèse que le réseau de neurones à propagation avant n'est pas fait pour des données séquentielles c'est-à-dire des textes [55]. Le modèle DSSM conçu avec le réseau neuronal convolutif, dites le **C-DSSM** prend en entrée une donnée textuelle et la coupe

en N-gram de mots. Comme tous les autres modèles du réseau neuronal convolutif, ce processus se fait grâce à une petite fenêtre qui glisse et qui parcourt le texte. Pour le C-DSSM, cette fenêtre présentera un trigramme.

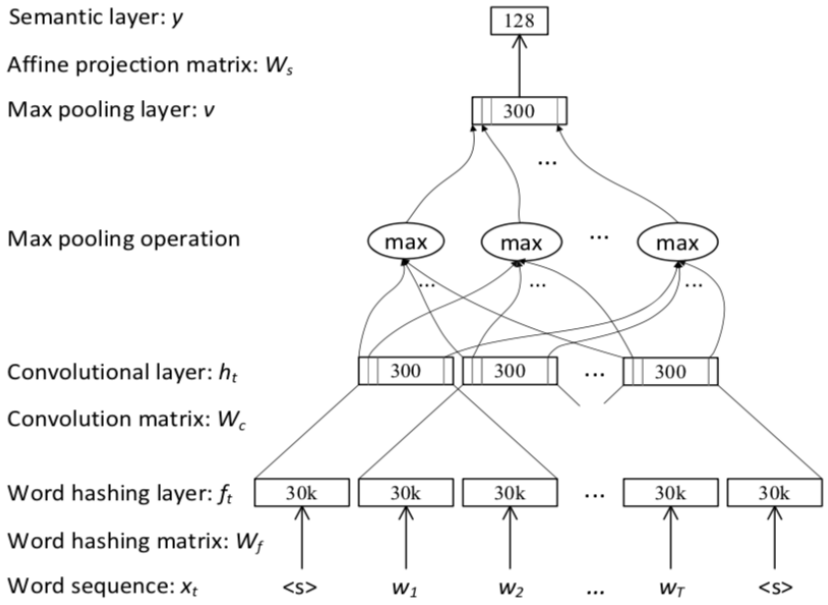


FIGURE 7 – Structure du modèle C-DSSM.

L'architecture du C-DSSM [56] montrée dans la figure [7], elle contient une couche de *hashage de mot* qui convertit dans un premier temps, chaque mot en vecteur. Dans un deuxième temps, ces vecteurs sont convertis en vecteurs de trigramme afin de pouvoir intensifier l'information. Ensuite, la couche de convolution récupérera l'information du contexte pour chaque mot et la couche de max-pooling formera un vecteur global du texte. Finalement, la couche sémantique donnera un vecteur sémantique global de l'entrée textuelle.

1.6.3.3 Réseau neuronal réccurent à mémoire court et long terme

Le réseau récurrent à mémoire court et long terme (en anglais, *Long short-term memory*) est une nouvelle architecture du réseau de neurones récurrents associée à un algorithme d'apprentissage basé sur un gradient approprié [57].

Contrairement au réseau neuronal à propagation avant, les réseaux de neurones récurrents

possèdent des boucles sur chaque neurones. Cela a pour but de faire circuler et enrichir l'information dans le réseau.

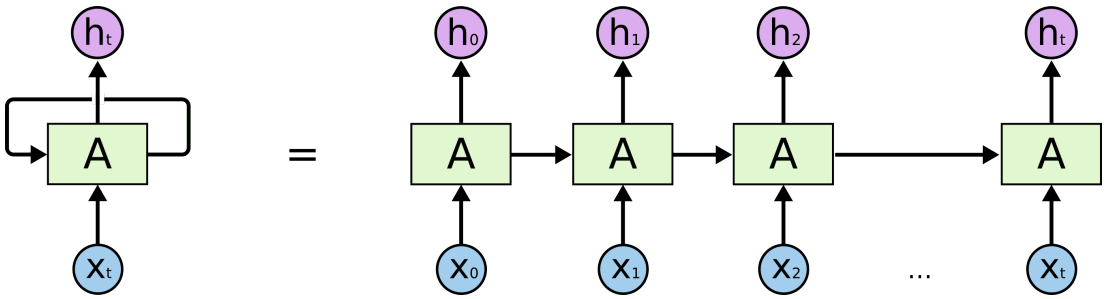


FIGURE 8 – Neurone récurrent [3].

Comme nous pouvons le voir dans la figure [8], la boucle sur le neurone signifie la communication de l'information. Cela montre que le réseau est capable d'utiliser des informations des entrées ou sorties précédentes pour le calcul d'une unité particulière. C'est la raison pour laquelle les réseaux de neurones récurrents (RNN) fonctionnent mieux que les réseaux à propagation avant ou convolutifs [57].

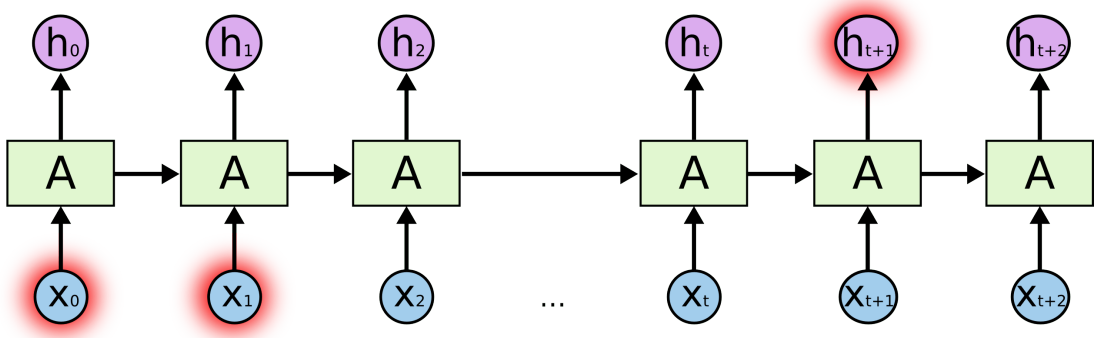


FIGURE 9 – Mémoire à long terme [3].

Pour pouvoir concrétiser le processus de *Mémoire à long terme*, il nous parait utile de montrer un exemple. Admettons que nous essayons de faire prédire au modèle le dernier mot d'un texte. Il faudrait donc que le réseau puisse apprendre à utiliser les anciennes informations. S'il s'agit d'un texte court, ou plus précisément, quand l'information recherchée est proche du mot donnant le contexte, les RNN pourront apprendre à le faire.

En revanche, comme nous pouvons le voir dans la figure [9], à partir du moment où l'information recherchée h_{t+1} dont son contexte X_0 et X_1 possèdent une véritable distance entre eux, les RNN ne pourront pas apprendre à revenir en arrière.

Ce phénomène est appelé en français *Mémoire à long terme* ou *Dépendance à long terme*. Ce problème de *mémoire* a été cité pour la première fois dans (Hochreiter 1991)[58] et (Bengio 1994)[59]. La différence principale entre le modèle de réseaux de neurones réccurent classiques et celui à mémoire court et long terme est basée sur l'approche *forget gate*. Cela est utilisé pour dire au modèle quelle information à garder en multipliant avec **1**. Si la valeur de *forget gate* est **0**, l'information sera supprimée.

C'est pour cette raison que le LSTM a été proposé par *Hochreiter* et *Schmidhuber* en 1997 [57] afin de pouvoir résoudre le problème de mémoire à long terme. Il faut souligner que le LSTM n'a pas pour but d'essayer d'apprendre à mémoriser l'information à long terme, au contraire, il est conçu pour cela [3].

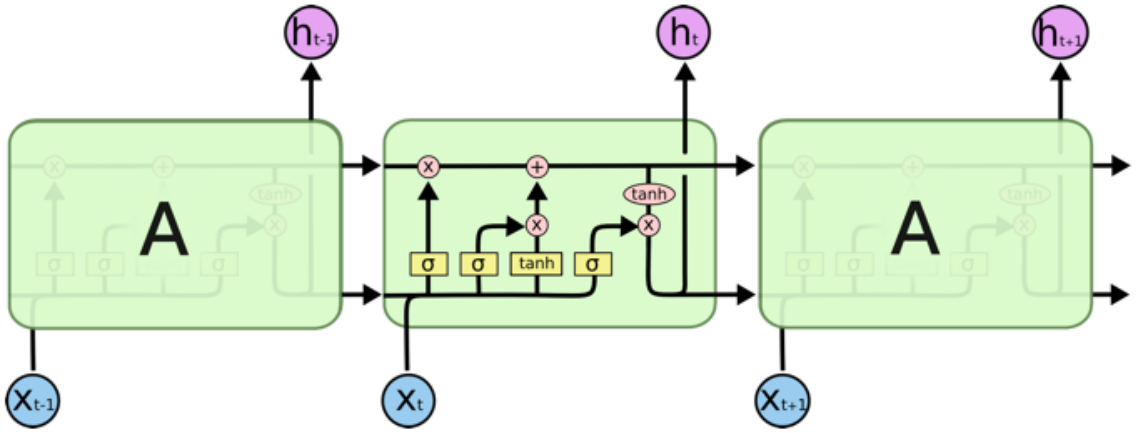


FIGURE 10 – Structure LSTM [3].

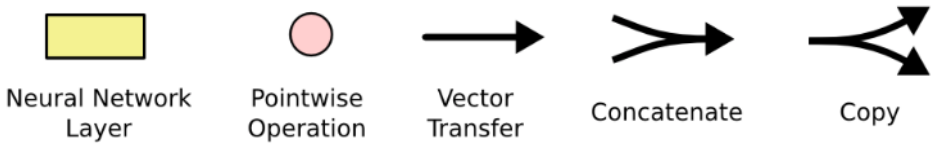


FIGURE 11 – Explication des noeuds du modèle LSTM [3].

Comme nous pouvons le voir dans la figure [10], le système est assez complexe. Les

significations sont montrées dans la figure [11]. Nous avons, tout d'abord, les boîtes jaunes qui représentent des couches de réseaux neuronaux. Ensuite, les cercles roses indiquent des opérations comme l'addition ou la multiplication des vecteurs. Il est également possible de faire une concaténation ou une duplication des vecteurs. Le LSTM apprendra à prendre une décision sur quelle information à garder ou ignorer.

Dans l'architecture du LSTM, nous avons une structure des *portes* (en anglais, *gates*) qui permettent de filtrer l'information. Chaque porte indique une opération à faire et la décision à prendre est contrôlée grâce à celles-ci. Le réseau pourrait rajouter une information ou en supprimer une ou encore, concaténer des informations en passant par une de ces portes. Ce processus se fait à l'aide de la fonction *sigmoid*. Elle retournera une valeur entre 0 et 1. Par exemple, le "1" signifie que le réseau laissera tout passer mais en cas de "0", le réseau ne laissera rien passer. Grâce à celle-ci, le LSTM pourra contrôler ses cellules.

1.6.4 Bidirectional Encoder Representations from Transformers (BERT)

Suite à l'introduction d'un nouveau modèle de réseau neuronal Transformer [60] en 2017, donnant des résultats très pertinents dans le domaine du traitement automatique des langues, BERT (**B**idirectional **E**ncoder **R**epresentations from **T**ransformers) a été proposé par (Devlin et al. 2018) [4].

BERT est conçu pour entraîner les représentations bidirectionelles à partir des données textuelles non-étiquetées en faisant attention au contexte gauche et droite en même temps dans toutes les couches du réseau avec son système d'attention [4].

Avant BERT, il existait deux modèles pour avoir des représentations des langages pré-entraînées. Ce sont, tout d'abord, l'approche basée sur les caractéristiques (en anglais, *feature-based*) comme ELMo [61] et l'approche basé sur le fine-tuning (en français, *re-apprentissage*) comme OpenAI GPT (Generative Pre-trained Transformer) [62]. La différence entre les représentations vectorielles de **word2vec** et celles de **BERT/ELMo** est basée sur la prise en compte du contexte. Dans word2vec, chaque mot possède un seul

vecteur mais en réalité, un mot peut avoir des sens différents en fonction du contexte. Quant à BERT ou ELMo, les représentations vectorielles seront dynamiques. Cela veut dire que les vecteurs d'un mot changeront en fonction du contexte dans lequel le mot apparaît.

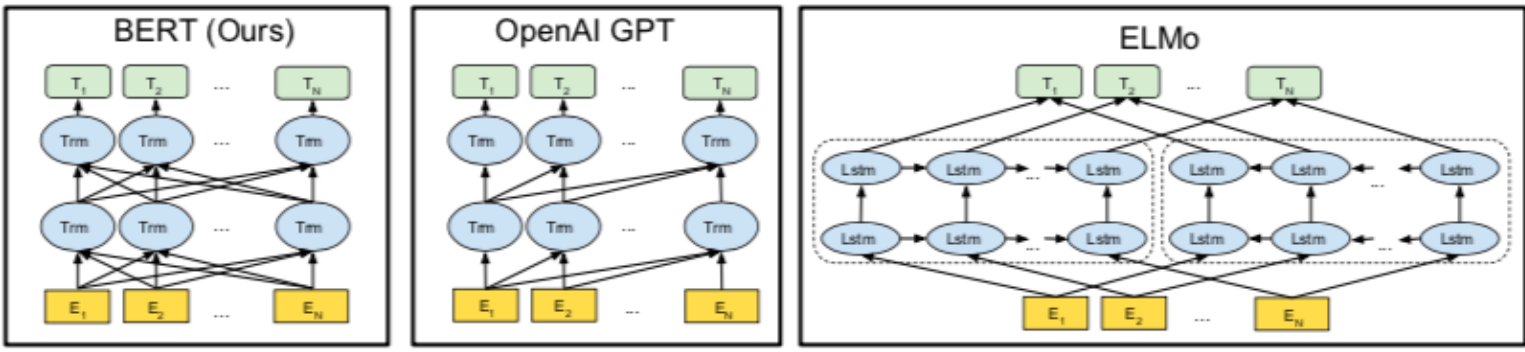


FIGURE 12 – Différence entre les architectures de BERT, ELMo et OpenAI [4]

Comme nous pouvons le voir dans la Figure [12], OpenAI GPT utilise, par exemple, un modèle de langage unidirectionnel pour obtenir des représentations générales d'un langage [4]. Le problème du modèle unidirectionnel consiste à utiliser une architecture allant seulement dans un seul sens c'est-à-dire de gauche à droite ou inversement. Cela signifie que chaque token ne peut faire attention qu'aux tokens suivants ou précédents dans les couches d'auto-attention de Transformer. En revanche, dans le modèle ELMo, en faisant une concaténation de deux lectures dans un sens unique c'est-à-dire premièrement allant de gauche à droite et deuxièmement de droite à gauche, les contextes gauches et droites sont obtenus.

Contrairement à ces modèles, BERT résout définitivement cette contrainte unidirectionnelle avec sa structure bidirectionelle et en visant un objectif de pré-entraînement du modèle de langage masqué (MLM, en anglais *Masked language model*) inspiré par (Taylor, 1953)[63]. Grâce à cette architecture, BERT n'utilise pas les modèles traditionnels comme celui *de gauche à droite* ou celui *de droite à gauche*. Le modèle bidirectionnel permet à chaque mot de "se voir lui-même" en observant le contexte gauche et droite en même temps. De cette manière, il prédit le mot cible en observant son contexte. Ce processus se

fait à l'aide du système d'auto-attention que nous expliquerons en détail ultérieurement.

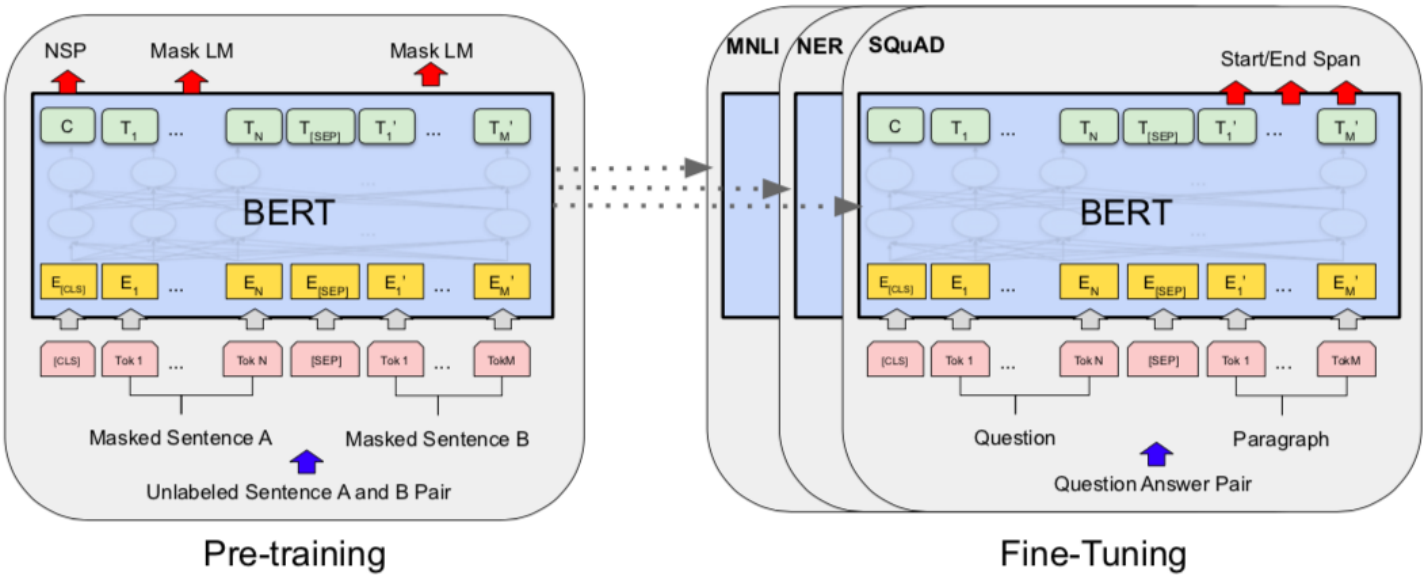


FIGURE 13 – Différence entre le pré-entraînement et le fine-tuning sur une tache de Réponse/Question (en anglais, *Question/Answer*) [4]

BERT utilise l'architecture du réseau Transformers [60] qui se compose de plusieurs "têtes" (en anglais, *heads*) c'est-à-dire, un réseau neuronal entièrement connecté entre chaque neurones (en anglais, *fully-connected*) avec un système d'auto-attention [64].

Dans cette architecture, il y a douze couches d'Encodeur. Chaque encodeur est composé de deux sous-couches : une couche d'auto-attention multi-tête, une couche de FFN (Réseau à propagation avant, en anglais, *Feed-Forward Network*) que nous expliquerons en détail ultérieurement.

Dans le modèle BERT, il existe deux étapes importantes. Ce sont le pré-entraînement et le fine-tuning. La structure de deux processus est montrée dans la figure [13]. Le pré-entraînement utilise deux méthodes de l'apprentissage non-supervisé. La première consiste à masquer, de manière aléatoire, un mot dans une phrase et de le faire prédire au modèle. La deuxième consiste à prendre deux phrases et à faire prédire au modèle si la deuxième phrase est la suivante de la première. Pendant le pré-entraînement, le modèle est entraîné sur des données non-etiquetées.

L'objectif du fine-tuning est d'entraîner des données étiquetées, pour une tache

spécifique, à partir d'un modèle déjà pré-entraîné [4]. Pendant le fine-tuning, une ou plusieurs couches *fully connected* sont rajoutées sur la dernière couche d'encodeur.

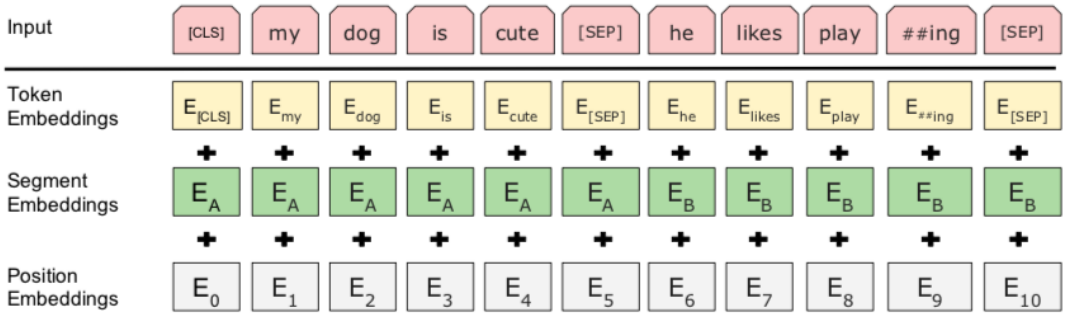


FIGURE 14 – Représentation de l'entrée dans BERT [4].

BERT [4] utilise le WordPiece [65] avec 30.000 mots de vocabulaire. En ce qui concerne la tokenisation, il existe quelques tokens spéciaux. Chaque séquence en entrée devrait commencer par le token spécial **[CLS]**. Celui-ci permet également d'avoir une représentation de la phrase pour faire une classification. Les paires de phrase doivent être mises dans une seule séquence grâce à un autre token spécial **[SEP]** comme nous pouvons le voir dans la figure [13].

Grâce à ce processus, BERT peut comprendre quel token appartient à quelle séquence. Il s'interesse également à la position de chaque token pour prendre en compte l'ordre de la phrase. Ces deux informations sont gardées sous forme de plongement. Afin de construire une représentation générale de l'entrée, BERT fait un processus en additionnant ces informations. Il additionne les plongements des mots de chaque token et les deux autres plongements qui contiennent des informations sur la séquence et la position. Cette structure est illustrée dans la figure [14].

Tenney et al. (2019)[66] ont montré que grâce à cette structure, BERT est capable d'encoder plusieurs informations sur les entités nommées, les relations, les rôles sémantiques.

Chapitre 2

Corpus de travail

2.1 Présentation du corpus

2.1.1 Présentation générale

Les données utilisées pour la réalisation de ce travail proviennent des bases de données de différentes entreprises. Chaque base de données appartient à une société bénéficiant du logiciel **Yoomap**. Les données utilisées possèdent des informations sur les entreprises et les différentes startups ainsi que sur les idées d'innovation. Plus précisément, nous avons utilisé les descriptifs courts. Ceux-ci donnent de l'information soit sur les secteurs d'activité des entreprises ou des startups, soit sur des produits proposés par celles-ci. Quant aux idées d'innovations, ce sont des textes courts de quelques lignes qui proposent une idée d'innovation pour un secteur donné.

Pour des raisons de confidentialité, nous ne pourrons pas partager les bases de données entièrement mais il sera utile de partager certains exemples afin de montrer la structure des bases de données. Les exemples de celle-ci seront présentés dans la figure [15].

2.1.2 Architecture de la base de donnée

En ce qui concerne la structure de la base de données, les champs **id** et **title** sont obligatoires mais le reste ne l'est pas. Le champs **status** est utilisé pour pouvoir trier

les entreprises. Certains status sont associés à certaines entreprises pour ne pas être pris en compte par notre algorithme. Yoomap préfère laisser le choix aux entreprises pour ses outils intelligents. Pour ce genre de cas, nous avons donc utilisé ces status pour les exclure. Comme nous pouvons le voir dans la figure [15], le champ **1_startup_description** contient des descriptifs courts concernant l'entreprise ou la startup. L'objectif est d'avoir une définition sur leurs secteurs d'activité ou sur les produits qu'elles proposent. Les champs **startup_thematic** et **startup_subcategories** permettent d'avoir une information en plus comme des *mot-clés* et de pouvoir classer les entreprises dans le logiciel Yoomap.

id	title	status	1_startup_description	startup_thematic	startup_subcategories
20612	10-Vins	statut1	D-Vine is the first wine-by-the-glass connected sommelier that aerates and brings the wine in your bottles to the ideal serving temperature in less than a minute. Live a new experience with this connected sommelier, tested and approved by enthusiasts, connoisseurs and professionals.	09. RETAIL & DIGITAL IN STORE	Product Interaction

FIGURE 15 – Exemple d'une plateforme contenant les informations sur des entreprises et des startups.

Il existe également un autre type de base de données contenant des idées d'innovation. L'exemple de celle-ci avec les champs est présenté dans la figure [16].

id	title	status	idee_phrase	idee_resume	thematic_idee	porteur_depot_unite
834736	Attention ça tombe!!	emergencepublic	Vigilance autour d'une charge qui peut tomber et faire son chemin au delà de la zone de sécutité.	Le but de ce materièl est de matérialiser le cone de sécurité (hauteur de la charge = rayon du cercle au sol) afin que les personnes passant à proximité d'une charge puisse visualiser cette zone de sécurité. Mais il s'agit aussi de montrer les divers comportements d'une charge qui tombe en fonction de sa forme , de sa hauteur , de son poids mais aussi des caractéristiques du sol qu 'elle va heurter. Les divers matériels mis en oeuvre doivent permettre aux participant d'apréhender les risques encourus dans le cône de levage mais aussi de la vigilange nécéssaire au delà en cas de rupture d'un élingage ou d'une fausse manoeuvre .	Sécurité des personnes	CNPE BLAYAIS

FIGURE 16 – Deuxième exemple d'une base de donnée contenant des idées d'innovation.

Comme nous pouvons le voir dans la figure [16], pour cette base de données, il y a sept champs qui contiennent des informations différentes. Les champs **id** et **title** sont obligatoires mais le reste ne l'est pas. Il arrive parfois que le client ne renseigne pas tous les

champs. Les champs **idee_phrase** et **idee_resume** portent des informations cruciales pour notre tâche c'est-à-dire les idées d'innovations. Le champs **idee_phrase** contient une seule phrase qui donne le concept de l'idée proprosée et le champs **idee_resume** contient un résumé de celle-ci avec plus de détails notamment l'objectif et le secteur d'activité visés. Les champs **thematic_idee** apporte une information complémentaire pour la catégorisation. Le dernier champs **porteur_depot_unite** indique l'organisation qui propose l'idée.

2.2 Utilisation de la base de donnée

Comme nous avons pu le voir dans la section précédente 2.1.2, nous avons différents types de bases de données avec des informations différentes.

Dans le cadre de ce travail, au lieu de se baser uniquement sur les descriptifs des entreprises ou des résumés des idées proposées, nous avons décidé d'utiliser tous les champs textuels de la base de données en ayant pour but d'avoir plusieurs d'informations concernant une entreprise ou une idée. Le champs **status** existe bel et bien dans toutes les bases de données pour pouvoir exclure les entreprises ou les idées pour celles qui ne souhaitent pas être pris en compte par notre algorithme. Après avoir fait notre tri, pour toutes les entreprises et les idées qui restent, nous avons concaténé tous les champs sauf ceux **id** et **status** afin de pouvoir obtenir un texte enrichi contenant des informations variées.

Chapitre 3

Méthodes et architecture du système

3.1 Constitution du corpus de référence

L'objectif principal de notre travail est de créer un système qui mesure la similarité, celui-ci sera utilisé comme système de recommandation. Suite aux expériences réalisées par l'ancienne équipe sur la mesure de similarité avec le calcul cosinus en utilisant les plongements des mots des anciens modèles, les résultats obtenus n'ont pas été jugés pertinents sachant que le système de recommandation déployé fonctionne déjà avec une architecture TF-IDF. Notre deuxième objectif est donc d'apporter une nouveauté et de mettre en application une nouvelle approche. De ce fait, nous avons décidé d'entraîner un modèle de réseau de neurones pour une tâche de classification. Pour ce faire, nous avons cherché à constituer un corpus de référence afin de pouvoir faire notre apprentissage supervisé. Nous avons donc récupéré les données textuelles depuis les bases de données.

3.1.1 Annotation manuelle

Yoomap possède de nombreux clients et de nombreuses bases de données différentes qui contiennent des informations concernant les entreprises ou les idées d'innovation. Pour pouvoir couvrir tous les secteurs possibles, nous avons choisi 13 plateformes, donc 13 bases de données différentes. Pour des raisons de confidentialité, nous ne pourrons pas partager les noms des clients. Les plateformes utilisées seront donc numérotées.

Afin de pouvoir faire un apprentissage supervisé, il fallait procéder à l'annotation manuelle qui est l'étape la plus importante et la plus coûteuse. Pour ce faire, nous avons récupéré 100 entreprises ou idées pour chacune de ces 13 plateformes. Ces échantillons récupérés ont été donnés à l'ancien système de recommandation (modèle version 1 déjà déployé) qui fonctionne avec l'architecture de **TF-IDF**. Le système nous a retourné trois suggestions pour chaque phrase en entrée. De cette manière, nous avons pu constituer notre corpus à annoter. Nous avons fait ce choix puisque c'était une manière rapide pour obtenir des paires de phrases pour commencer à faire l'annotation.

Le tableau qui montrent des informations statistiques est présenté dans le tableau [10].

	Nombre de paires de phrases récupérées	**Nombre total des phrases existantes dans la base de données**
Client 1	300	2342
Client 2	300	1473
Client 3	300	8773
Client 4	300	3652
Client 5	300	185
Client 6	300	6214
Client 7	300	514
Client 8	300	457
Client 9	300	493
Client 10	300	197
Client 11	189	63
Client 12	300	901
Client 13	300	1117

Tableau 10 – Quelques statistiques concernant les plateformes utilisées

Après avoir observé les paires de phrase créées, nous avons effectué certains nettoyages pour les doublons et les paires que l'on a jugées inefficaces. Suite à cela, nous avons obtenu **3410 paires de phrase** à annoter. Notre corpus est un corpus multilingue qui contient des données en anglais et en français. En ce qui concerne le prétraitement des textes, nous avons décidé de préserver les données brutes à l'exception de quelques nettoyages en raison

du choix de notre modèle que nous expliquerons ultérieurement. Pour cette étape, nous avons choisi de :

— supprimer l'élément **&** qui était présent plusieurs fois dans nos données
— supprimer des adresses mails et des sites web, des hashtags en utilisant des regex.
— supprimer les sauts de ligne.

3.1.1.1 Guide d'annotation

Ce guide d'annotation a été inspiré de celui du DEFT 2018 [1]. Celui-ci n'est pas fait exactement pour la même tâche de classification mais nous l'avons adapté à notre tâche car il nous semblait important d'expliquer le processus de nos annotations plus en détail avec des exemples concrets. Les descriptifs des entreprises et les idées innovantes ont été annotées en deux catégories : **similaire** et **non-similaire**. La classe binaire peut apparaître insuffisante quand nous parlons de la similarité sémantique. Nous avons préféré avoir seulement deux classes pour deux raisons :

— La facilité pour le test **Kappa**
— Le fait que ce soit plus adaptable pour le machine learning

Les paires de phrase peuvent être multilingues ou unilingues. Lors de l'annotation, on s'est mis à la place d'un client. L'objectif est de trouver les entreprises qui travaillent dans le même secteur et/ou qui proposent des produits similaires ou identiques avec ou sans lexique commun afin de pouvoir afficher une entreprise similaire pour une entreprise dont son profil est consulté par un client.

Similaire

La similarité que nous cherchons à mesurer peut se baser à la fois sur le lexique commun et sur le lexique synonyme pour le concept du produit ou de l'idée proposé. Nous présenterons certains exemples pour concrétiser nos données textuelles.

1. https://perso.limsi.fr/pap/DEFT2018/annotation_guidelines/index.html

– Si les deux entreprises/idées sont dans le même domaine et qu'ils produisent des éléments/produits tout en gardant le même esprit du concept et la même utilité, alors cette paire de phrase est **similaire**.

Premier exemple :

Label	**Phrase 1**	**Phrase 2**
1	L'expert d'assurance qui vous simplifie la vie. Une photo de vos contrats vous suffit pour enfin comprendre et optimiser vos contrats d'assurance ! Autres Technologies IA / Automatisation Auto	Prenez contrôle de vos assurances via notre robo-advisor. Agrégateur de contrats d'assurance FinTech AssurTech Auto MRH Prévoyance Santé

Deuxième exemple :

Label	**Phrase 1**	**Phrase 2**
1	Transpoco provides GPS vehicle tracking solutions for vehicle owners who want increased control with their vehicle. IoT Auto	Le mini-traceur GPS Inetis intelligent est une innovation 100% française. Le plus petit et le plus performant des traceurs gps disponible sur le marché. IoT Auto MRH

– Si les deux entreprises/idées sont dans le même domaine et qu'ils proposent des produits avec une idée différente du même concept, alors la paire de phrase est **similaire**. L'objectif principal est de trouver une thématique similaire entre les deux entreprises contrairement au premier cas similaire (le sujet de l'invention identique). Grâce à cela, le modèle pourrait faire attention à différents vocabulaires utilisés pour une thématique similaire.

Premier exemple :

Label	Phrase 1	Phrase 2
1	Human Ressources use AI tool to improve quantity and quality of CVs. Use AI tool to improve engagement of candidates. Use AI tools to evaluate candidate's competency in order to reduce manual work and bias. Artificial Intelligence	A video interview management platform. A tool that uses Artificial Intelligence to analyze word choice, tone, and facial movement of job applicants who do video interviews. It does not replace the final face-to-face interview. The idea is that the AI helps highlight the top performers so that recruiters can dive in and spend time with the most promising candidates. Using artificial intelligence to screen job applicants is a glimpse at the future of recruiting. Recruitment Artificial Intelligence

Deuxième exemple :

Label	Phrase 1	Phrase 2
1	D'un point de vue technique, Stimul comprend la mise à disposition d'un objet connecté d'activité physique et suivi du rythme cardiaque ainsi que celle d'un cursus d'éducation thérapeutique de 12 mois structuré par l'intervention à distance d'un éducateur thérapeutique dans une démarche d'accompagnement santé vers la ré-insertion sociale et professionnelle. Digital, Health Sector, M2M industrial internet, IOT, Machine Learning Artificial Intelligence Data and Analytics Digital & Information Technology Hardware Heathtech & Biotechnology Industry & Manufacturing Software	KAP CODE est un objet connecté à destination des patients asthmatiques adaptable à tous types d'inhalateurs. Ce dispositif permet de recueillir toutes les informations liées à l'utilisation des traitements inhalés tout en restituant ces données sous une forme communautaire. Health Sector, M2M industrial internet, IOT, Machine Learning, Smart Home Artificial Intelligence Building & Cities Data and Analytics Digital & Information Technology Hardware Heathtech & Biotechnology Industry & Manufacturing Software

Non-similaire

– Si les deux entreprises/idées ne sont pas dans le même domaine et qu'ils proposent des produits complètement différents , alors la paire de phrase est **non-similaire**.

Exemple :

Label	Phrase 1	Phrase 2
0	Notre produit combine notre algorithme neuromorphique propriétaire intégré à une application pour smartphone et un t-shirt intelligent permet d'établir, de suivre et de prédire l'évolution de l'état de santé du patient.	Une solution de sacs cycliste/piéton, luxe : convertibles, intelligent et fabriqués en France par des compagnons du devoir. Les sacs se clipsent sur tous les vélos, s'éclairent pour protéger le cycliste et lui permet de devenir piéton avec panache en moins de deux secondes.

– Si les entreprises/idées proposent des éléments/produits complètement différents en étant dans le même domaine, la paire de phrase est **non-similaire**. Ces produits différents ne peuvent pas avoir une thématique ou utilité similaire ni un concept de l'idée identique.

Exemple :

Label	Phrase 1	Phrase 2
0	Utilisez la puissance des conversations et de l'intelligence artificielle Proposez vos services sous forme de messagerie grâce aux chatbots. Digital Innovation Relation Client	En quelques mots, Vidata permet d'améliorer les programmes de relation client, en enrichissant automatiquement vos push et vos sites web avec des vidéos personnalisées à partir de datas client. Client, communications

3.1.2 Évaluation

Au cours de l'étape d'annotation des 3410 paires de phrase constituant notre corpus, il a fallu les évaluer afin de s'assurer de sa fiabilité. Les annotations ont été faites par deux annotateurs. Pour cela, nous avons décidé de calculer l'**accord inter-annotateur** entre nous grâce à la mésure du **Kappa de Cohen**. Le script utilisé pour ce calcul est présenté dans l'annexe [1].

Résultat du test Kappa
0.66

Le résultat obtenu concerne un échantillon de 150 paires de phrase. D'après la table d'interprétation du **Kappa de Cohen** [67], nous avons obtenu un **accord fort** (entre 0.61 et 0.80). Ce résultat montre une certaine cohérence.

3.2 Modèle informatique : Choix des algorithmes

Un des objectifs de ce travail est de trouver une nouvelle approche optimale pour le calcul des similarités entre textes courts comme présenté précédemment. Celle-ci doit être adaptée à notre corpus.

3.2.1 Classification de textes avec BERT

Nous avons d'abord décidé d'utiliser le modèle **BERT** (Bidirectional Encoder Representations from Transformers) que nous avons expliqué dans la section [1.6.4]. Pour ce faire, nous avons choisi une méthode de **re-apprentissage** (en anglais, *fine-tuning*). Entraîner un modèle *from scratch* avec BERT nécessite un corpus gigantesque et des matériels puissants pour l'apprentissage comme des **GPUs**.

Le fine-tuning nous permet de prendre un modèle déjà pré-entraîné et de le re-entraîner plus finement pour une tâche donnée. Pour notre tâche, nous avons décidé d'utiliser un modèle **multilingue** entraîné par Google avec les données de Wikipedia. [2] Pour cela, nous avons utilisé la librairie *transformers* de *huggingface* [3] et celle *PyTorch* [4]. Lors du fine-tuning, le modèle utilise les paramètres appris sur les données initiales en apprenant les nouvelles données que nous lui fournissons.

Nous ne détaillerons pas le fonctionnement du modèle **BERT** que nous avons déjà expliqué dans la section [1.6.4]. Cependant, il nous semble important de concrétiser les entrées et les sorties du modèle avec des exemples.

Comme indiqué plus haut, pour le pré-traitement de nos données, nous avions préféré garder l'originalité des textes en effectuant seulement quelques nettoyages. La raison vient

2. https://github.com/google-research/bert/blob/master/multilingual.md
3. https://huggingface.co/transformers/
4. https://pytorch.org/docs/stable/index.html

du fait que le modèle BERT possède un système d'attention et qu'il est capable d'encoder plusieurs informations sur les relations entre mots voire les rôles sémantiques [66].

Prenons les deux exemples suivants :

Phrase 1	J'ai mangé une orange.
Phrase 2	La couleur de la voiture est orange.

Avec BERT, nous ne pouvons pas utiliser n'importe quel tokenizer puisqu'il en a déjà un et celui-ci fonctionne avec la technique WordPiece. Cette technique a pour but de renforcer l'apprentissage du modèle. Imaginons que dans notre corpus, il y a plusieurs mots comme **walking**, **walked**, **walkers**, **walks**. Les modèles qui n'utilisent pas la technique WordPiece considèrent ces mots comme des mots différents. L'objectif du WordPiece est de couper le mot en sous-mots comme **walk** et **##ing** ou **walk** et **##ed**. De cette manière, le modèle pourrait apprendre plus sur le mot **walk**.

La visualisation de la tokenisation pour les deux phrases données en haut est la suivante :

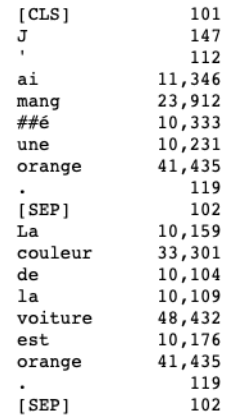


FIGURE 17 – Visualisation de la tokenization de BERT

Comme nous pouvons le voir dans la figure [17], notre paire de phrase a été tokenizée. Les numéros que nous voyons dans la figure [17] sont les **ids** puisque dans le vocabulaire de BERT, chaque token possède un id. De cette manière, BERT est capable de répérer de quel mot il s'agit.

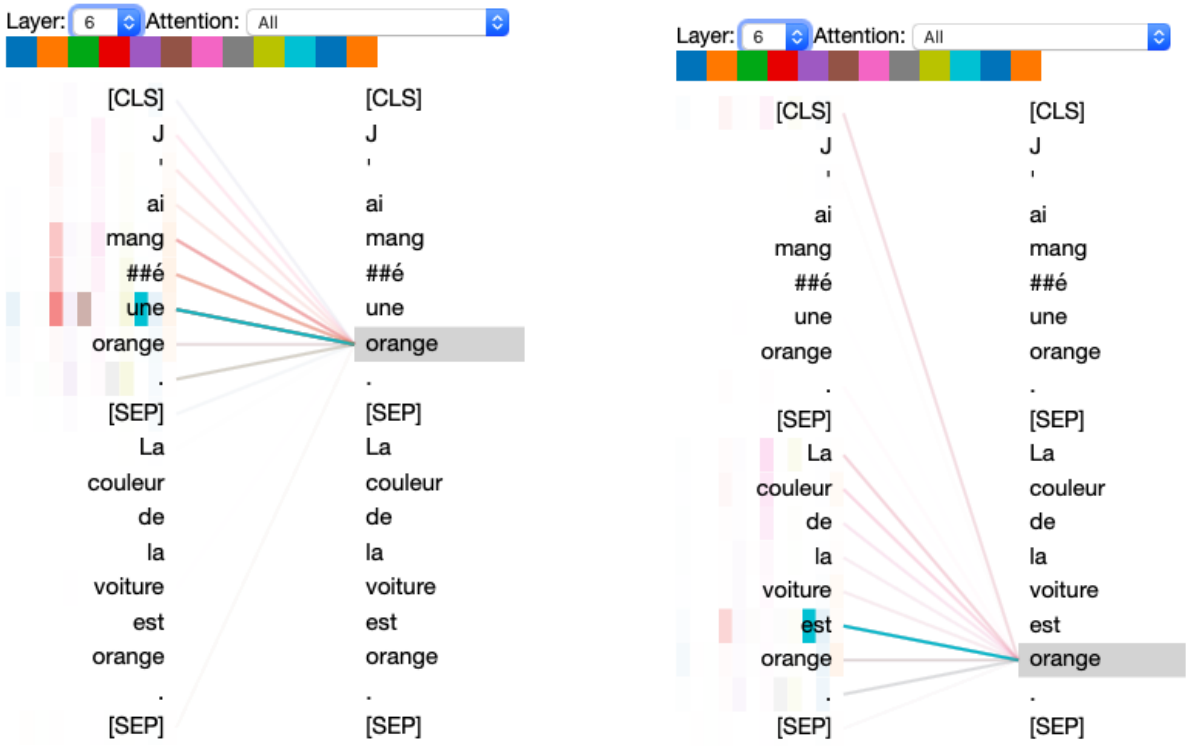
```
Input ids : tensor([[  101,   147,   112, 11346, 23912, 10333, 10231, 41435,   119,   102,
         10159, 33301, 10104, 10109, 48432, 10176, 41435,   119,   102]])

Segment ids : tensor([[0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1]])

Attention Mask : tensor([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]])
```

FIGURE 18 – Les plongements initiaux de BERT

Comme nous pouvons le voir dans la figure [18], nous avons trois plongements en entrée. Le premier **Input ids** concerne les ids des tokens. Le deuxième **Segment ids** porte l'information sur la position des tokens. Dans ce tensor, les **0** signifient la première phrase et les **1** signifient la deuxième phrase. Grâce à cela, BERT est capable de savoir quel token appartient à quelle phrase. Le dernier plongement **Attention Mask** porte l'information sur l'attention de BERT. Dans ce tensor, les **1** indiquent au modèle de quel token à prendre en compte. Il nous arrive de mettre des **0** pour le cas où nous souhaitons que les tensors aient la même taille. Nous appelons ce processus en anglais *padding*. Lorsque nous rajoutons les **0**, le modèle ne fait aucun traitement pour ceux-ci et il les ignore. Finalement, notre entrée pour une paire de phrase concerne ces trois plongements et une étiquette pour indiquer si la paire est similaire ou non-similaire (pour nous, c'est 1 ou 0).

Système d'attention de BERT

Afin de pouvoir concrétiser le fameux système d'atttention, nous avons décidé de le visualiser grâce à l'outil de Bertviz [68]. Le script utilisé est présenté dans l'annexe [2].



(a) Premier exemple du système d'attention.

(b) Deuxième exemple du système d'attention.

FIGURE 19 – Exemples du système d'attention de BERT.

Comme nous pouvons le voir dans les figures [19a], pour le mot **orange**, l'attention est mis sur le participe passé du verbe **manger** et sur le déterminant du mot **orange**. Nous observons également que le mot qui indique le fruit orange ne fait aucun lien avec l'autre mot orange qui indique la couleur.

Dans le deuxième exemple [19b], nous observons que pour le mot **orange**, l'attention est mis sur le verbe **être** et le mot **couleur**. Nous pouvons donc voir que BERT est capable de comprendre que ces deux mots **orange** ne signifient pas la même chose. Comme indiqué plus haut dans la section [1.6.4], BERT possède une architecture **bidirectionelle**. Cela signifie que le modèle, pour un mot, sera capable de regarder son contexte de gauche et celui de droite en même temps. Pour chacune des 12 couches, les poids d'attention changeront. Cela veut dire que le modèle essayera de faire attention à d'autres choses pour chaque couche. C'est donc cette architecture qui rend BERT si pertinent et puissant [68]. D'autres exemples sur le système d'attention concernant nos données seront présentés dans

la partie [4.4].

3.2.2 Similarité cosinus avec les plongements de BERT

Nous avons essayé de mesurer la similarité en calculant le cosinus entre les plongements BERT. Pour cela, nous avons suivi deux méthodes. La première consiste à utiliser les plongements du token spécial **[CLS]** qui importe une information générale sur la phrase [69]. La deuxième consiste à prendre tous les plongements de chaque token et prendre leur moyenne afin d'obtenir une représentation générale pour une phrase donnée [70].

Pour ce faire, nous nous sommes basés sur le papier *Sentence-BERT* [5] (appelé aussi S-BERT), une approche qui regroupe ces deux méthodes en rajoutant la structure du réseau de neurones siamois (en anglais, *Siamese network*).

L'architecture siamois nous permet de mesurer une similarité à partir de deux entrées indépendantes. L'approche S-BERT propose d'utiliser cette structure avec BERT [5].

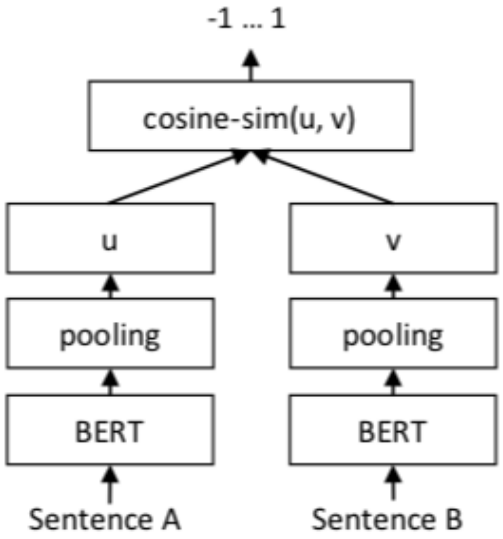


FIGURE 20 – S-BERT architecture pour l'inférence, par exemple, obtenir des scores de similarité avec le cosinus [5].

Comme nous pouvons le voir dans la figure [20], nous avons deux textes différents en entrée qui sont traités séparément par le même modèle BERT. L'objectif est d'obtenir des

plongements et de calculer la similarité cosinus pour savoir si deux textes en entrée sont similaires ou non [5].

3.2.3 Similarité cosinus avec les plongements d'Universal Sentence Encoder

Après avoir effectué des vectorisations avec S-BERT, nous avons voulu tester le modèle pré-entraîné d'Universal Sentence Encoder [5]. Pour cela, nous avons utilisé la librairie de *TensorFlow Hub* de Google [6]

Le modèle Universal Sentence Encoder[71] utilise l'architecture de **Transformers** [60] et le système d'attention pour pouvoir mieux capturer le contexte des mots pour une phrase donnée. C'est la raison pour laquelle nous avons voulu tester ce modèle afin de pouvoir observer si les plongements produits seraient efficaces ou non.

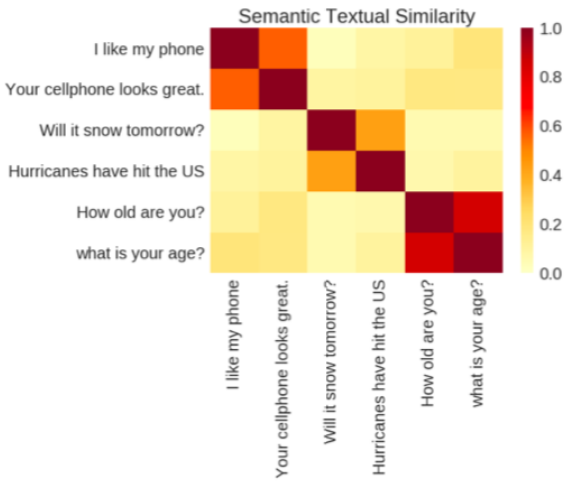


FIGURE 21 – Examples des scores de similarités obtenus grâce au modèle Universal Sentence Encoder

Il existe un autre modèle d'Universal Sentence Encoder avec une architecture **CNN** mais nous avons préféré privilégier celui de Transformers. Celui-ci possède plusieurs avantages qui sont :

5. https://tfhub.dev/google/universal-sentence-encoder/2
6. https://tfhub.dev

– La rapidité du traitement
– L'utilisation simple
– La présence du système d'attention
– L'existence d'un modèle multilingue [7] qui concerne 16 langues [72].

Les plongements obtenus grâce au modèle **Universal Sentence Encoder** peuvent être utilisés pour une tache de classification de textes ou la mesure de similarité textuelle [72] [71].

7. https://tfhub.dev/google/universal-sentence-encoder-multilingual/3

Chapitre 4

Résultats et discussion

4.1 Re-apprentissage du modèle BERT : Classification de textes

Après le fine-tuning de notre modèle BERT, nous avons obtenu la matrice de confusion ci-dessous, figure [22]. Comme nous pouvons le voir, la classe **similaire** contient 283 paires et celle **non-similaire** 213. Nous constatons que les 147 éléments sur 213 et 226 éléments sur 283 ont été bien classés. Les résultats des mesures de précision, rappel et f-mesure pour chaque classe sont présentés dans le tableau [11].

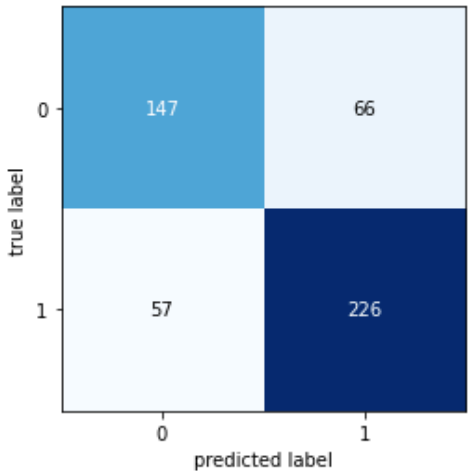


FIGURE 22 – Matrice de confusion du modèle BERT ré-entraîné.

	Précision	**Rappel**	**F-Mesure**
not similar(0)	0.72	0.69	0.71
similar(1)	0.77	0.80	0.79
accuracy			**0.75**
moyenne macro	0.75	0.74	0.75
moyenne pondérée	0.75	0.75	0.75

Tableau 11 – Mesures de précision, rappel et f-mesure par classe pour la classification de BERT ré-entraîné.

Comme nous pouvons le voir dans le tableau [11], les résultats obtenus sont satisfaisants. Les meilleurs résultats sont obtenus dans la classe **similaire** avec une f-mesure de **0.79**. Quant à l'autre classe **non-similaire**, nous avons une f-mesure de **0.71**. Comme indiqué plus haut avec les nombres de données indiqués, notre corpus de test n'est pas équilibré au niveau du nombre des données pour chaque classe. Par conséquent, cela ne posera pas problème puisque ça peut être considéré comme une réalité du corpus lorsque nous parlons de la similarité sémantique.

La précision nous indique la proportion des éléments classés correctement parmi l'ensemble des éléments proposés. Le rappel indique la proportion des éléments classés correctement parmi les éléments pertinents. La f-mesure présente la moyenne harmonique des deux premières mesures (*f-mesure* et *rappel*). Quant à la macro-moyenne, elle calculera la précision et le rappel indépendamment pour chaque classe, puis prendra la moyenne. (*Si le nombre des classes est supérieur à 1, la macro-moyenne peut être calculée.*)

Afin de comparer les résultats obtenus par notre modèle, nous avons décidé d'évaluer le modèle multilingue pré-entraîné par Google (*bert-multilingual-cased*) sur notre corpus de test. La matrice de confusion obtenue est présentée dans la figure [23]. Les résultats des mesures de précision, rappel et f-mesure pour chaque classe sont présentés dans le tableau [12].

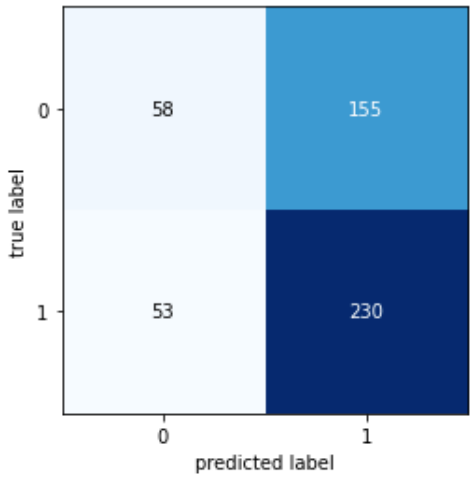


FIGURE 23 – Matrice de confusion du modèle BERT standard (bert-multilingual-cased).

	Précision	**Rappel**	**F-Mesure**
not similar(0)	0.52	0.27	0.36
similar(1)	0.60	0.81	0.69
accuracy			**0.58**
moyenne macro	0.56	0.54	0.52
moyenne pondérée	0.57	0.58	0.55

Tableau 12 – Mesures de précision, rappel et f-mesure par classe pour la classification de BERT standard (bert-multilingual-cased)

Comme nous pouvons le voir dans le tableau [12], les résultats ne sont pas satisfaisants. Nous constatons que pour ce modèle qui n'a pas eu le processus du fine-tuning, les 55 éléments sur 213 et 230 éléments sur 283 ont été bien classés. Nous pouvons dire que le modèle a tendance à prédire souvent **1**. La f-mesure de 58% que nous avons obtenue nous montre que le modèle n'est pas capable de détecter une telle similarité. Si nous avions construit un modèle qui prédit seulement le 1, nous aurions obtenu une f-mesure de 57%. Le modèle multilingue standard ne fait que 1% de différence. Cela montre et prouve que notre modèle a pu acquérir une certaine logique pendant son re-apprentissage.

Courbes d'apprentissages

Nous nous sommes intéressés beaucoup aux courbes d'apprentissage du modèle puisque le processus du fine-tuning peut subir rapidement un surapprentissage. Comparé aux apprentissages standards, le nombre d'*époque* est conseillé entre 1 et 4 quand il s'agit du fine-tuning. [1] Pour notre tâche, nous avons donc préféré réaliser le fine-tuning pour 2 époques.

La courbe indiquant la précision sur le corpus d'entraînement et de validation est montrée dans la figure [24].

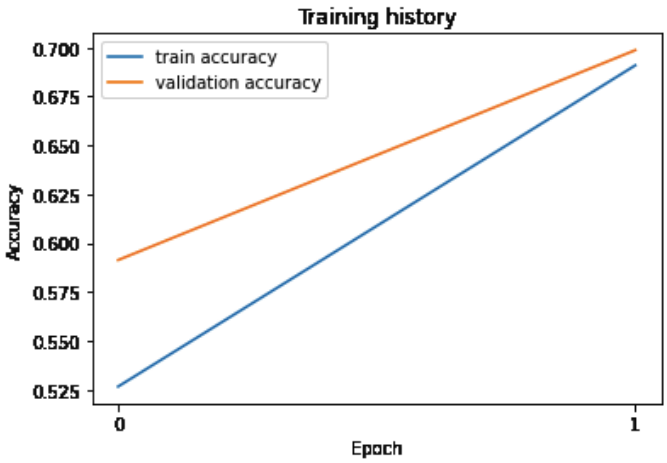


FIGURE 24 – Courbe indiquant la précision sur le corpus d'entraînement et de validation.

Pendant l'apprentissage, le modèle calcule la valeur *loss*. Celle-ci est utilisée pour évaluer la mesure dans laquelle les sorties vraies sont correctement prédites par le modèle. La courbe indiquant la fonction de *loss* est montrée dans la figure [25].

1. https://github.com/google-research/bert

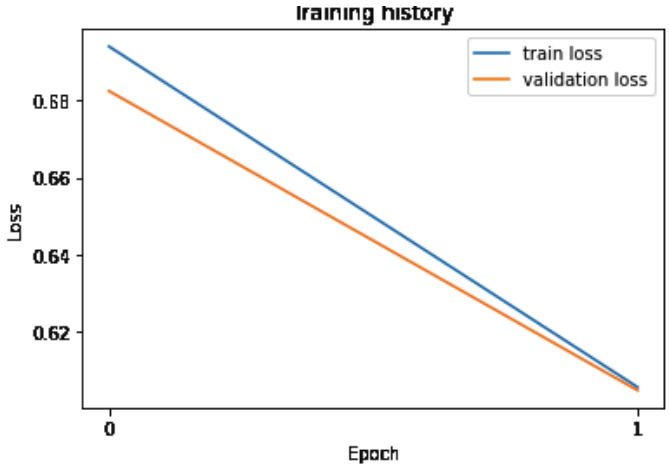


FIGURE 25 – Courbe indiquant les valeurs *loss*

Réglages des hyperparamètres du modèle

Les hyperparamètres du fine-tuning et les autres réglages du modèle utilisés sont indiqués dans le tableau [13]. Pour le processus de notre fine-tuning, nous avons décidé d'utiliser le modèle multilingue (**bert-multilingue-cased** [2]). Le paramètre de la longueur maximale concerne la longueur de l'entrée après la tokenisation. BERT accepte jusque 512 tokens en entrée pour une paire de phrase. Ce paramètre joue également un rôle important sur la performance de l'entraînement. Pour notre tâche, cette longueur était suffisante mais s'il s'agit de documents ou de textes longs, BERT ne sera pas un modèle idéal à cause de cette contrainte.

Nom du modèle	**Longueur maximale**	**Taux d' apprentisage**	**Taille de corpus de Train/Val/Test**	**Batch Size**	**Epoch**	**Valeur de dropout**	**F-Mesure**
Bert-multilingue-cased	512	**2e-5**	0.7/0.15/0.15	8	2	0.1	**%75**
Bert-multilingue-cased	512	5e-5	0.7/0.15/0.15	12	2	0.1	%71
Bert-multilingue-cased	512	3e-5	0.7/0.15/0.15	12	2	0.1	%69

Tableau 13 – Tableau des expériences effectuées avec les hyperparamètres

Il existe quelques hyperparamètres importants pour le fine-tuning de BERT. Nous avons réalisé notre re-apprentissage en trois fois. Comme nous pouvons le voir dans le

2. https://github.com/google-research/bert/blob/master/multilingual.md

tableau [13], le taux d'apprentissage **2e-5** donne la meilleure f-mesure. Ces taux utilisés sont conseillés par les auteurs de BERT [4]. En revanche, ce taux d'apprentissage augmente lors de l'entraînement de manière linéaire. Nous appelons cette technique en anglais, *linear learning rate warmup*. Pour diviser notre corpus, nous avons décidé de constituer un corpus d'entraînement sur 70% de nos données et les 30% qui reste ont été divisés en moitié pour le corpus de validation et celui de test.

Pour notre modèle, nous avons utilisé la taille de batch de **8**, le **AdamW** comme *optimizer* avec un taux d'epsilon 1e-6 (par defaut) et la valeur de dropout de **0.1**. Celle-ci est utilisée pour pouvoir éviter au maximum le surapprentisage.

Nous avons effectué notre fine-tuning avec un seul GPU (**GeForce GTX 1050 Ti with Max-Q Design**). Avec celui-ci, l'entraînement a duré **1 heure 37 minutes**.

4.2 Similarité cosinus avec les plongements de BERT

En ce qui concerne la similarité cosinus avec les plongements de BERT, les résultats obtenus sont moins intéressants. Concernant ce modèle, nous n'avons pas pu l'évaluer sur le corpus de test que nous avons utilisé pour le fine-tuning de BERT car la similarité cosinus que nous obtenons est un score qui varie entre 0 et 1. Le fait de mettre un seuil comme 0.70 ou 0.75, pour prédire 1 au dessus de ce seuil ou 0 pour en dessous de celui-ci, nous semblait aléatoire.

Nous avons donc décidé d'évaluer le modèle sur un autre corpus de 900 entreprises. S-BERT nous permet d'utiliser un modèle BERT qui a déjà eu un proccesus du fine-tuning mais il était tout à fait possible d'utiliser le modèle pré-entraîné par Google comme par exemple celui multilingue standard (bert-multilingual-cased). Nous avons préféré utiliser notre propre modèle ré-entraîné. Le corpus que nous avons constitué pour l'évaluation de ce modèle n'a pas été utilisé pendant le fine-tuning. Nous avons été très vigilants sur ce point puisque nous évaluerons aussi notre modèle ré-entraîné sur ce corpus afin de pouvoir comparer les résultats.

Après avoir obtenu les plongements pour le corpus entier, nous avons choisi 100 phrases de manière aléatoire parmi 900. Pour chacune de ces 100 phrases, nous avons récupéré 3

suggestions les plus similaires obtenues par le calcul cosinus. Nous avons donc eu 3 paires de phrases différentes pour une seule phrase donnée. Ensuite, pour évaluer la pertinence, nous avons annoté ces paires de phrases obtenues à deux annotateurs. Le modèle est donc dépendant de ces suggestions. Il pourrait y en avoir de meilleurs suggestions en dehors de ces trois.

Nom du modèle	**Accuracy**	**Kappa**
S-BERT (CLS TOKEN)	%28	0.68
S-BERT (Moyenne des tokens)	%36	0.72

Tableau 14 – Résultats obtenus de la similarité cosinus avec S-BERT.

D'après la table d'interprétation du Kappa de Cohen, nous avons obtenu un **accord fort** (entre 0.61 et 0.80) pour les deux annotations. Ces scores obtenus concernent un échantillon de 150 paires de phrase pour chaque modèle. Le script utilisé pour ce calcul est présenté dans l'annexe [1].

Comme nous pouvons le voir dans le tableau [14], nous avons pu expérimenter que le token spécial **[CLS]** n'est pas efficace pour obtenir des plongements sémantiques pour une phrase donnée. Quant à l'autre modèle qui consiste à prendre la moyenne de tous les plongements de chaque token, il donne aussi des résultats non-pertinent mais il s'agit d'une augmentation de 8% de précision comparé au modèle précedent.

4.3 Similarité cosinus avec les plongements d'Universal Sentence Encoder

Les derniers plongements testés pour mesurer la similarité cosinus est ceux obtenus par Universal Sentence Encoder. Les résultats obtenus sont meilleurs que ceux obtenus avec S-BERT.

Afin de pouvoir évaluer le modèle, nous nous sommes basés sur le même corpus que nous avons utilisé pour l'évaluation du modèle S-BERT. Pour que cela soit cohérent, et que nous puissions comparer les deux approches, nous avons pris les mêmes 100 phrases

et nous avons obtenu 3 suggestions pour chacune de ces 100 phrases grâce à deux modèles différents d'Universal Sentence Encoder, celui standard et celui multilingue. Ensuite, nous avons annoté les suggestions obtenues comme l'expérience précédente.

Nom du modèle	**Accuracy**	**Kappa**
Universal Sentence Encoder	%47	0.68
Universal Sentence Encoder Multilingue	%59	0.70

Tableau 15 – Résultats obtenus de la similarité cosinus avec Universal Sentence Encoder.

D'après la table d'interprétation du Kappa de Cohen, nous avons également obtenu un **accord fort** (entre 0.61 et 0.80) pour ces deux annotations. Ces scores obtenus concernent un échantillon de 150 paires de phrase pour chaque modèle. Le script utilisé pour ce calcul est présenté dans l'annexe [1].

Comme nous pouvons le voir dans le tableau [15], le modèle multilingue d'Universal Sentence Encoder s'en sort mieux que le modèle standard d'Universal Sentence Encoder car celui-ci est basé seulement sur l'anglais. Pour cette partie de l'évaluation, notre corpus était aussi multilingue mais nous avons voulu tester tout de même le modèle standard qui n'a au final pas donné de résultats pertinents avec une précision de 47%.

Pour la dernière partie de cette évaluation, comme indiqué plus haut, nous avons évalué notre propre modèle BERT ré-entraîné sur ce corpus. Nous avons suivi la même méthodologie en prenant les mêmes 100 phrases et nous les avons donné au modèle pour pouvoir récupérer 3 suggestions pour chacune de ces 100 phrases. Ensuite, nous avons annoté ces suggestions obtenues. Avec ce modèle, nous avons obtenu 77% de précision et un score de Kappa de 74%. Ces résultats obtenus sont satisfaisants et seront discuté dans la partie [4.4] en présentant un tableau récapitulatif de toutes les expériences réalisées.

4.4 Récapitulatif des résultats et discussion générale

Les résultats obtenus pour chacune des expériences sont présentés dans le tableau [16].

Nom du modèle	**Précision**	**Kappa**
S-BERT (CLS TOKEN)	%28	0.68
S-BERT (Moyenne des tokens)	%36	0.72
Universal Sentence Encoder	%47	0.68
Universal Sentence Encoder Multilingue	%59	0.70
BERT Multilingue ré-entraîné (fine-tuned)	**%77**	0.74

Tableau 16 – Tableau récapitulatif des résultats obtenus

Chaque expérience réalisée nous amène à tirer certaines conclusions. En ce qui concerne le score de Kappa pour le modèle BERT ré-entraîné, nous avons obtenu un **accord fort** (entre 0.61 et 0.80). Tous ces scores de Kappa calculés pour chaque modèle nous montrent une certaine cohérence dans nos annotations.

Comme nous pouvons le voir dans le tableau [16], les résultats obtenus avec le modèle S-BERT nous montre que le fait d'utiliser le vecteur du token **[CLS]** pour mesurer la similarité entre deux phrases n'apporte aucune aide pour détecter une telle similarité. Contrairement à celui-ci, le fait d'utiliser la moyenne de tous les plongements de chaque token donne des meilleurs résultats avec une augmentation de 8%.

Les résultats obtenus avec le modèle Universal Sentence Encoder nous montre que son architecture est capable d'encoder plus d'information comparé au modèle S-BERT puisque même le modèle non-multilingue d'Universal Sentence Encoder nous a donné une précision de 47%. Quant à celui multilingue, il nous montre qu'il est encore capable de détecter une telle similarité entre les phrases avec une précision de 59%.

Concernant notre modèle ré-entraîné, les résultats obtenus sont satisfaisants. La première raison est le fait de ré-entraîner un modèle qui est déjà pré-entraîné et qui contient déjà beaucoup d'information sur les langages. Le modèle arrive donc à mieux classer les paires de phrases. Par conséquent, les plongements obtenus de ce modèle ne sont pas tellement efficaces. Cela prouve, en quelque sorte, notre hypothèse de départ. Quand il s'agit des textes

courts, la similarité sémantique attendue n'est pas souvent détectée par les plongements [73] [10].

Le modèle BERT ré-entraîné possède également quelques inconvénients. L'un de ces inconvénients est lié à la performance. En effet, nous étions très curieux de voir les résultats de S-BERT à cause d'un problème de performance que nous avons rencontré lors de l'exécution de notre modèle ré-entraîné. Comme indiqué plus haut, Yoomap possède de nombreux clients et de nombreuses bases de données. Afin de mettre en place le système de recommandation, il nous fallait exécuter notre modèle sur toutes les bases de données et enregistrer les suggestions en sortie pour chaque entreprise ou idée d'innovation. De cette manière, lorsqu'un profil d'une entreprise est consulté, il était possible d'appeler les suggestions en se basant sur les ids des entreprises pour pouvoir les afficher dans le logiciel de Yoomap. Il s'agit donc de milliers de descriptifs et d'idées d'innovation à traiter.

Lorsque nous les faisions prédire au modèle, nous avons pu remarquer que BERT était très lent pour nous retourner les prédictions. Avec un seul GPU, nous avions eu une estimation d'environ 8 mois de traitement pour toutes les bases de données. Suite à l'autorisation de l'entreprise, nous avons donc décidé de louer un cloud contenant **4 GPU Tesla V100 16GB** pour faire notre expérience. Pour cette infrastructure, nous avons utilisé une autre structure en parallelisant notre modèle. La parallelisation nous permet de diviser notre donnée en entrée en nombre des GPU. Admettons que nous avons une taille de batch de 8, le modèle envoie donc 2 paires de phrases à chacun des 4 GPUs. L'objectif est d'exécuter les GPUs en même temps et de réduire le temps de traitement. Pour cette expérience, nous avons choisi 4 plateformes parmi 13 plateformes. Les résultats indiquant les différents temps de traitement sont présentés dans le tableau [17].

	GeForce GTX 1050 Ti with Max-Q Design x1	**TESLA V100 x4**	**Nombre de données à traiter**
Client 4	199 heures	15 heures	3652
Client 12	18 heures	1 heure	901
Client 11	5 minutes	46 secondes	63
Client 1	119 heures	6 heures	2342

Tableau 17 – Tableau indiquant les temps de prédictions estimées avec les GPUs

Comme nous pouvons le voir dans le tableau [17], le temps de traitement avec un seul GPU **GeForce GTX 1050 Ti** n'est pas envisageable. En revanche, les 4 GPUs **Tesla V100** nous ont beaucoup aidé pour réduire le temps de traitement mais le modèle reste quand même lent et coûteux sachant que le système doit être mis à jour régulièrement puisqu'il y a souvent des rajouts ou des suppressions dans les bases de données.

La raison pour laquelle nous étions curieux concernant les résultats du modèle S-BERT est qu'avec celui-ci, après avoir fait le fine-tuning, il restait seulement calculer la similarité cosinus avec les plongements obtenus sans avoir un problème de performance. Nous n'avons hélas pas obtenu des résultats pertinents.

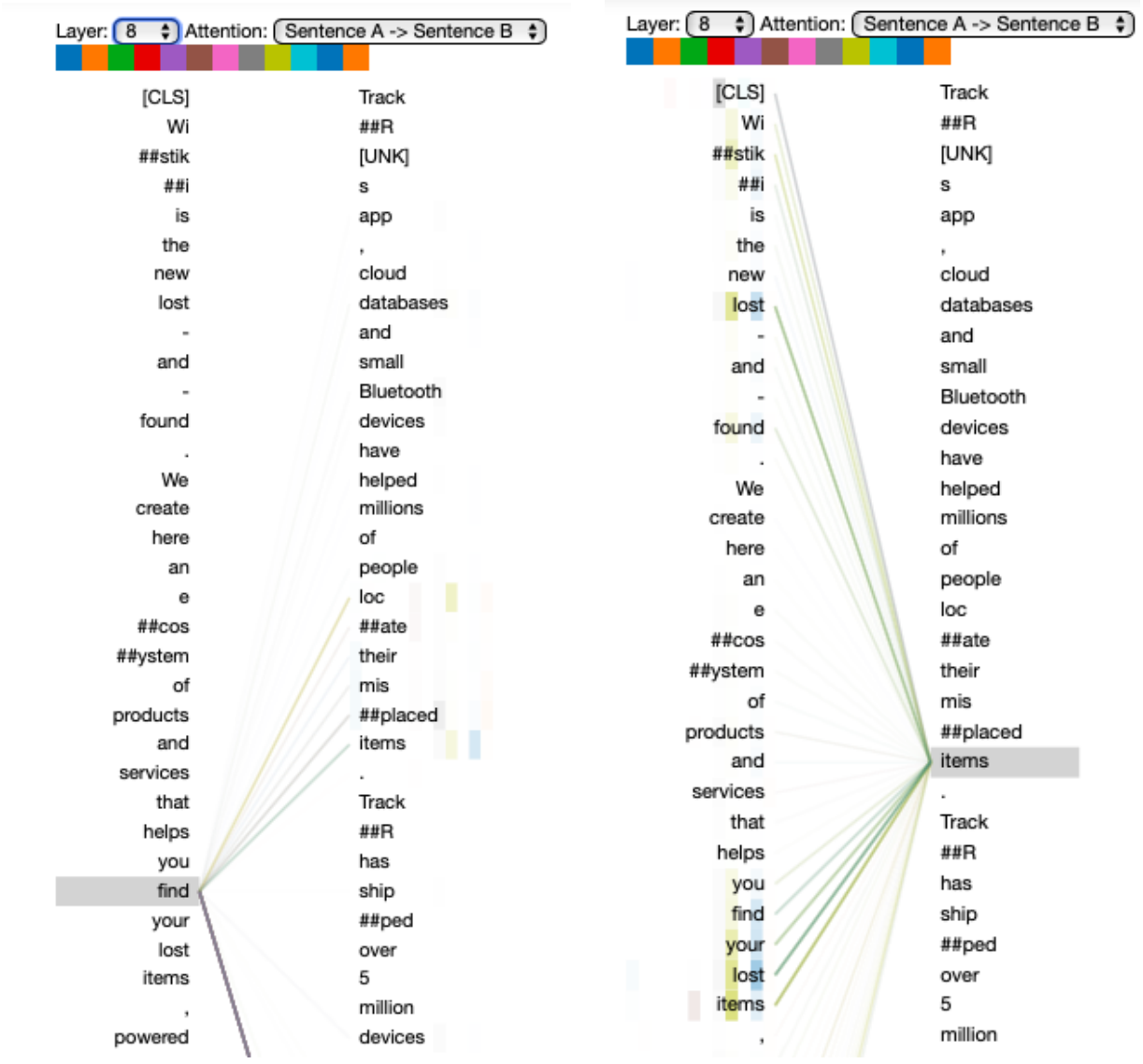
En ce qui concerne la puissance du modèle BERT ré-entraîné comparé aux autres modèles qui sont basés sur la similarité cosinus, il sera utile de montrer quelques exemples pour visualiser le système d'attention de BERT sur un exemple de nos données.

Phrase Initiale	Wistiki is the new lost-and-found. We create here an ecosystem of products and services that **helps you find your lost items**, powered by a strong community. Our 1st product is a **small connected device** also called Wistiki that locates and makes your lost items ring using your smartphone or tablet.
Première suggestion	TrackR's app, cloud databases and **small Bluetooth devices** have helped millions of people **locate their misplaced items**. TrackR has shipped over 5 million devices and is sold in over 5,000 stores worldwide.

Tableau 18 – Exemple d'une suggestion pour une entreprise donnée au modèle

Comme nous pouvons le voir dans le tableau [18], la phrase initiale est donnée au modèle et la suggestion retournée est la première parmi les trois. La similarité détectée a été indiquée en gras dans le tableau.

Concernant ces deux phrases, nous avons décidé de visualiser le système d'attention de notre modèle. Les résultats obtenus sont présentés dans les figures [26a] et [26b]. Le script utilisé pour la visualisation est présenté dans l'annexe [2].



(a) Premier exemple du système d'attention.

(b) Deuxième exemple du système d'attention.

FIGURE 26 – Exemples du système d'attention de BERT sur nos données.

Comme nous pouvons le voir dans la figure [26a], le système d'attention, pour le mot "**find**" a mis ses poids sur l'ensemble des mots "**locate their misplaced items**". Nous pouvons donc dire que BERT a pu faire un lien entre les mot **find** et **locate**. Dans le deuxième exemple [26b], nous observons que le mot "**items**" met ses poids sur des mots différents dans la phrase. Ce sont d'abord le nom de l'entreprise "**Wistiki**" et le mot "**lost**" et sur l'ensemble des mots "**find your lost items**".

Ces résultats prouvent que notre modèle BERT ré-entraîné est capable de détecter la similarité même entre les mots synonymes (*find* et *locate*) [66] puisque le modèle intègre des informations sémantiques sur les langues naturelles et en faisant le fine-tuning, nous lui avons ajouté des nouvelles données. C'est la raison pour laquelle, celui-ci a obtenu les meilleurs résultats parmi cinq modèles testés. Son système d'attention est à la fois un point très fort pour la sémantique des textes courts et en même temps un point très coûteux pour le traitement des corpus possédant des milliers de données.

Les résultats obtenus grâce au modèle Universal Sentence Encoder multilingue peuvent être considérés moyennement satisfaisants. En revanche, les suggestions obtenues ne sont pas similaires avec celles du modèle BERT ré-entraîné. Cela est tout à fait normal. Il ne serait pas logique de s'attendre à avoir les mêmes suggestions alors que d'un côté il s'agit d'un modèle qui a eu un apprentissage supervisé et de l'autre côté il s'agit d'un modèle pré-entraîné par Google qui produisent des plongements.

Les résultats obtenus grâce au modèle S-BERT ne sont vraiment pas satisfaisants. En observant les suggestions retournées par celui-ci, nous avons remarqué un point commun avec le modèle Universal Sentence Encoder. Comme indiqué plus haut dans la section [2.1.2], nous avions décidé de concaténer plusieurs champs de la base de donnée pour avoir un texte enrichi. Parmi ces champs, il y'en a un qui contient, en quelque sorte, les noms des catégories. Nous avons pu observer que ces deux modèles se sont basés plus souvent sur ces mots communs pour détecter une similarité entre deux phrases. Comme nous l'avions expliqué dans notre guide d'annotation, le fait que les deux entreprises soient dans le même domaine n'est pas suffisant pour dire que les descriptifs sont similaires. La précision que ce modèle a obtenue est due également à notre guide d'annotation strict. Ce point nous a amené à penser que certaines parties de notre travail auraient pu être envisagées

différemment. Ce problème des plongements était notre problématique principale parce qu'il est difficile de détecter une telle similarité et de faire un lien sémantique entre les mots quand une phrase possède peu de mots.

L'interprétation en profondeur des résultats nous a permis de mieux comprendre la problématique de similarité. Le modèle BERT ré-entraîné aurait pu avoir encore des meilleurs résultats si nous pouvions rajouter encore des données annotées sur notre corpus de référence.

Le fait d'annoter des paires multilingues nous a amené à vérifier si notre modèle serait capable de faire un lien entre mots multilingues. L'exemple est présenté avec une visualisation du système d'attention dans l'annexe. [4.4]

Finalement, nous pouvons dire que les résultats du modèle BERT ré-entraîné sont, malgré les différents points soulignés en haut, convaincants.

Conclusion et perspectives

Ce travail de recherche a permis de mettre en avant la capacité de BERT pour la mesure de similarité entre des textes courts. Nous avons cherché à construire un modèle qui serait utilisé comme un système de recommandation basé sur le contenu textuel pour un logiciel Web.

Nous avons d'abord présenté différentes approches existantes de cette problématique. Ensuite, nous avons présenté notre méthodologie et nos choix de l'algorithme. Notre choix a donc été de faire un re-apprentissage supervisé (en anglais, *fine-tuning*) sur un modèle pré-entraîné de BERT pour une tâche de classification.

Ensuite, nous nous sommes penchés sur l'annotation afin de pouvoir construire un corpus de référence. Pour ce faire, nous avons d'abord expliqué comment nous avons extrait les données textuelles à annoter depuis les bases de données. En ce qui concerne l'évaluation de notre annotation en similarité, nous avons calculé l'accord inter-annotateur qui était entre 0.61 et 0.80 pour toutes les expériences réalisées. Cela nous montre une certaine cohérence. En revanche, le score aurait pu être meilleur si nous avions utilisé 4 ou 5 degrés de similarité au lieu des classes binaires dans notre guide d'annotation. Les désaccords que nous avons eus entre nous est dû également à notre guide d'annotation strict.

L'utilisation de BERT nous a permis d'observer sa pertinence convaincante pour la détection de la similarité entre des textes courts grâce au système d'attention de Transformers. Cette pertinence nous a amené à faire d'autres expériences. Nous étions curieux d'observer si le système d'attention a un impact sur les plongements obtenus du modèle re-entraîné. Nous avons également voulu tester le modèle Universal Sentence Encoder possédant aussi un système d'attention de Transformers. Pour ces deux derniers modèles,

nous avons utilisé les plongements des mots afin de calculer la similarité cosinus. Les anciens modèles proposant ce calcul avec les plongements des mots ne possèdent pas un système d'attention. Il nous semblait donc important d'expérimenter la pertinence de ceux-ci. En revanche, nous n'avons hélas pas obtenu des résultats satisfaisants.

Une des conclusions principales tirée des expériences réalisées est que la similarité sémantique attendue n'est pas souvent détectée par les plongements quand il s'agit des textes courts. Cela n'est pas facile non plus pour des humains.

Finalement, nous pouvons dire que BERT est un modèle idéal qui arrive à détecter la similarité même entre des textes courts. En revanche, les prédictions sont lentes malgré les multiples GPUs puissants utilisés puisque BERT est composé de centaines de millions de paramètres. Cela est un point coûteux s'il s'agit de milliers de données à traiter. Est-il vraiment souhaitable de mettre en place des modèles très lourds s'ils apportent peu en performances ?

Perspectives

Les perspectives de ce travail sont multiples et auraient pour objectif d'augmenter les performances de classification de notre modèle. Pour cela, il serait utile de mieux observer les résultats obtenus, surtout les données mal classées afin de pouvoir mieux comprendre pourquoi le modèle s'est comporté ainsi. Il serait également intéressant de rajouter d'autres données annotées dans notre corpus de réference pour pouvoir observer si le modèle peut atteindre 80% ou 90% de précision.

En ce qui concerne la performance de l'inférence du modèle, une des pistes serait des techniques de compressions comme *quantification* qui vise à réduire la taille des poids ou *pruning* qui vise à supprimer des neurones [74].

Finalement, ce travail de recherche a permis de mettre en évidence la difficulté de similarité entre textes courts. Malgré tout, les résultats obtenus montrent une pertinence même si ceux-ci restent améliorables.

Bibliographie

[1] G. A. Miller, R. Beckwith, C. Fellbaum, D. Gross, and K. J. Miller, *Introduction to WordNet : An online lexical database*. International journal of lexicography 3, 1991.

[2] T. Mikolov, K. Chen, G. Corrado, and J. Dean, *Efficient Estimation of Word Representations in Vector Space*. ICLR Workshop, 2013.

[3] C. Olah, "*Understanding Long Short-Term Memory Networks*." `http://colah.github.io/posts/2015-08-Understanding-LSTMs`. Consulté le 16-04-20.

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

[5] N. Reimers and I. Gurevych, "*Sentence-BERT : Sentence Embeddings using Siamese BERT-Networks*," *arXiv:1908.10084 [cs.CL]*, 2019.

[6] D. Metzler, S. Dumais, and C. Meek, "Similarity measures for short segments of text," in *Advances in Information Retrieval* (G. Amati, C. Carpineto, and G. Romano, eds.), (Berlin, Heidelberg), pp. 16–27, Springer Berlin Heidelberg, 2007.

[7] Y. Lin, J. Jiang, and S. Lee, "A similarity measure for text classification and clustering," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 7, pp. 1575–1590, 2014.

[8] P. Foltz, "Latent semantic analysis for text-based research," *Behavior Research Methods*, vol. 28, pp. 197–202, 02 1996.

[9] M.Mohler and a. R. R.Bunescu, "Learning to grade short answer questions using semantic similarity measures and dependency graph alignments," *Proc. 49th Annu. Meeting Assoc. Comput. Linguistics, Hum. Lang. Technol*, 2011.

[10] H. Pu, G. Fei, H. Zhao, G. Hu, C. Jiao, and Z. Xu, "Short text similarity calculation using semantic information," in *2017 3rd International Conference on Big Data Computing and Communications (BIGCOM)*, pp. 144–150, 2017.

[11] X. Hu, N. Sun, C. Zhang, and T.-S. Chua, "Exploiting internal and external semantics for the clustering of short texts using world knowledge," in *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, CIKM '09, (New York, NY, USA), p. 919–928, Association for Computing Machinery, 2009.

[12] M. Balabanović and Y. Shoham, "Fab : Content-based, collaborative recommendation," *Commun. ACM*, vol. 40, p. 66–72, Mar. 1997.

[13] N. Wang, "Vers un système de recommandation à partir de traces sémantiques pour l'aide à la prise de décision," 2014.

[14] S. K. Lee, Y. H. Cho, and S. H. Kim, "Collaborative filtering with ordinal scale-based implicit ratings for mobile music recommendations," *Information Sciences*, vol. 180, no. 11, pp. 2142 – 2155, 2010.

[15] G. Salton and C. Buckley, "Term-weighting approaches in automatic text retrieval," *Inf. Process. Manage.*, pp. 513–523, 1988.

[16] E. Greengrass, "Information retrieval : A survey," 2000.

[17] G. Salton, A. Wong, and C. S. Yang, "A vector space model for automatic indexing," *Commun. ACM*, vol. 18, p. 613–620, Nov. 1975.

[18] C. Yves and M. Philippe, "La recherche d'information. de la documentation automatique à la recherche d'information en contexte," *Document numérique, 2007/1 (Vol. 10)*, pp. p. 11–38, 2007.

[19] T. A., *Features of similarity.* Psychological Review, 84(4), 1977.

[20] V. Hatzivassiloglou, J. L. Klavans, and E. Eskin, *Detecting text similarity over Short Passages : Exploring Linguistic Feature Combinations via Machine Learning.* Joint SIGDAT conference on empirical methods in natural language processing and very large corpora, 1999.

[21] W. H. Gomaa and A. A. Fahmy, *A Survey of Text Similarity Approaches.* 2013, nternational Journal of Computer Applications (0975 – 8887), Volume 68– No.13.

[22] W. E. Winkler, *String Comparator Metrics and Enhanced Decision Rules in the Fellegi-Sunter Model of Record Linkage.* Proceedings of the Section on Survey Research Methods, American Statistical Association, 1990.

[23] V. I. Levenshtein, *Binary codes capable of correcting deletions, insertions, and reversals (in russian).* Doklady Akademii Nauk SSSR, 1965.

[24] R. Rousseau, *Jaccard similarity leads to the marczewski-steinhaus topology for information retrieval.* Information Process Management, 1998.

[25] S. Niwattanakul, J. Singthongchai, E. Naenudorn, and S. Wanapu, *Using of jaccard coefficient for keywords similarity.* Proceedings of the International Conference on Computer, 2013.

[26] E. Ukkonen, *Approximate string-matching with q-grams and maximal matches.* Theoretical Computer Science https://doi.org/10.1016/0304-3975(92)90143-4, 1992.

[27] C. Shannon, *A mathematical theory of communications.* The Bell Systems Tech. J. 27, 1948.

[28] P. Jaccard, *Etude comparative de la distribution florale dans une portion des Alpes et du Jura.* Bulletin de la Société Vaudoise des Sciences Naturelles, 1907.

[29] Z. S. Harris, *Distributional Structure.* Word, 10 :2-3, 1954.

[30] S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman", *Indexing by latent semantic analysis.* Journal of the American Society for Information Science, 1990.

[31] T. Kenter and M. de Rijke, "Short text similarity with word embeddings," in *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, CIKM '15, (New York, NY, USA), p. 1411–1420, Association for Computing Machinery, 2015.

[32] E. L. Terra and C. L. A. Clarke, *Frequency estimates for statistical word similarity measures.* NAAC 03 Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1, 2003.

[33] Y. Li, D. McLean, Z. Bandar, J. O'Shea, and K. Crockett, "Sentence similarity based on semantic nets and corpus statistics," *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, pp. 1138–1150, 09 2006.

[34] P. Resnik, *Using information content to evaluate semantic similarity in a taxonomy.* arXiv preprint cmp-lg/9511007, 1995.

[35] D. Lin, *Extracting collocations from text corpora.* First workshop on computational terminology, 1998.

[36] J. J. Jiang and D. W. Conrath, *Semantic similarity based on corpus statistics and lexical taxonomy.* arXiv preprint cmp-lg/9709008, 1997.

[37] C. Leacock and M. Chodorow, *Combining local context and WordNet similarity for word sense identification.* WordNet : An Electronic Lexical Database, 1998.

[38] Z. Wu and M. Palmer, *Verbs semantics and lexical selection.* Proceedings of the 32nd annual meeting on Association for Computational Linguistics. Association for Computational Linguistics, 1994.

[39] A. Panchenko, *Comparison of the baseline knowledge-, corpus-, and web-based similarity measures for semantic relations extraction.* Proceedings of the GEMS 2011 Workshop on GEometrical Models of Natural Language Semantics. Association for Computational Linguistics, 2011.

[40] B. Danushka, M. Yutaka, and I. Mitsuru, "Websim : a web-based semantic similarity measure," 2007.

[41] Y. Matsuo, J. Mori, M. Hamasaki, K. Ishida, T. Nishimura, H. Takeda, K. Hasida, and M. Ishizuka., "Polyphonet : An advanced social network extraction system.," *Proc. of 15th International World Wide Web Conference*, 2006.

[42] M. Sahami and T. Heilman, "A web-based kernel function for measuring the similarity of short text snippets.," *Proc. of 15th International World Wide Web Conference*, 2006.

[43] H.-H. Chen, M.-S. Lin, and Y.-C. Wei, "Novel association measures using web search with double checking," in *Proceedings of the 21st International Conference on Com-*

putational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics, ACL-44, (USA), p. 1009–1016, Association for Computational Linguistics, 2006.

[44] G. T. L., S. M., and T. J. B., *Topics in semantic representation.* Psychological review, 2007.

[45] A. Bakarov, *A Survey of Word Embeddings Evaluation Methods.* arXiv:1801.09536, 2018.

[46] J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah, "Signature verification using a "siamese" time delay neural network," in *Proceedings of the 6th International Conference on Neural Information Processing Systems*, NIPS'93, (San Francisco, CA, USA), p. 737–744, Morgan Kaufmann Publishers Inc., 1993.

[47] S. Chopra, R. Hadsell, and Y. LeCun, "Learning a similarity metric discriminatively, with application to face verification," in *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1 - Volume 01*, CVPR '05, (USA), p. 539–546, IEEE Computer Society, 2005.

[48] A. Gordo, J. Almazán, J. Revaud, and D. Larlus, "End-to-end learning of deep visual representations for image retrieval," *CoRR*, vol. abs/1610.07940, 2016.

[49] D. Yi, Z. Lei, and S. Li, "Deep metric learning for practical person re-identification," *Proceedings - International Conference on Pattern Recognition*, 07 2014.

[50] R. Salakhutdinov and G. Hinton, *Semantic hashing.* Proc. SIGIR Workshop Information Retrieval and Applications of Graphical Models., 2007.

[51] R. Salakhutdinov and G. Hinton, *Discovering Binary Codes for Documents by Learning Deep Generative Models.* Topics in Cognitive Science., 2010.

[52] Y. Bengio, *Learning Deep Architectures for AI.* Foundumental Trends Machine Learning., 2009.

[53] P. Aghahoseini, *Short Text Similarity : A Survey.* ., 2019.

[54] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning representation by back-propagating errors.* Nature 323, 1986.

[55] Y. Shen, X. He, J. Gao, L. Deng, and G. Mesnil, "A latent semantic model with convolutional-pooling structure for information retrieval," in *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, CIKM '14, (New York, NY, USA), p. 101–110, Association for Computing Machinery, 2014.

[56] N. Nikhil and M. M. Srivastava, *Content Based Document Recommender using Deep Learning.* arXiv:1710.08321, 2017.

[57] S. Hochreiter and J. Schmidhuber, *Long Short-Term Memory.* Neural Computation 9(8), 1997.

[58] S. Hochreiter, "Untersuchungen zu dynamischen neuronalen netzen," 1991.

[59] Y. Bengio, P. Simard, and P. Frasconi, *Learning Long-Term Dependencies with Gradient Descent is Difficult.* IEEE TRANSACTIONS NEURAL NETWORS VOL. 5 NO. 2, 1994.

[60] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, *Attention Is All You Need.* arXiv:1706.03762 [cs.CL], 2017.

[61] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, *Deep contextualized word representations.* arXiv:1802.05365, 2018.

[62] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, *Improving language understanding with unsupervised learning.* Technical report, OpenAI, 2018.

[63] W. L. Taylor., *Cloze procedure : A new tool for measuring readability.* Journalism Bulletin, 30(4) :415–433, 1953.

[64] A. Rogers, O. Kovaleva, and A. Rumshisky, *A Primer in BERTology : What we know about how BERT works.* arXiv:2002.12327, 2020.

[65] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, and K. M. et al., *Google's neural machine translation system : Bridging the gap between human and machine translation.* arXiv preprint arXiv:1609.08144, 2016.

[66] I. Tenney, P. Xia, B. Chen, A. Wang, A. Poliak, R. T. McCoy, N. Kim, B. V. Durme, S. R. Bowman, D. Das, and E. Pavlick, *What do you learn from context ? Probing for sentence structure in contextualized word representations.* ICLR 2019, 2019.

[67] J. R. Landis and G. G. Koch, "The measurement of observer agreement for categorical data," *Biometrics*, vol. 33, no. 1, pp. 159–174, 1977.

[68] J. Vig, "A multiscale visualization of attention in the transformer model," *arXiv preprint arXiv:1906.05714*, 2019.

[69] Y. Qiao, C. Xiong, Z. Liu, and Z. Liu, "Understanding the behaviors of bert in ranking," 2019.

[70] A. Conneau, D. Kiela, H. Schwenk, L. Barrault, and A. Bordes, "Supervised learning of universal sentence representations from natural language inference data," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, (Copenhagen, Denmark), pp. 670–680, Association for Computational Linguistics, Sept. 2017.

[71] D. Cer, Y. Yang, S. yi Kong, N. Hua, N. Limtiaco, R. S. John, N. Constant, M. Guajardo-Cespedes, S. Yuan, C. Tar, Y.-H. Sung, B. Strope, and R. Kurzweil, "Universal sentence encoder," 2018.

[72] Y. Yang, D. Cer, A. Ahmad, M. Guo, J. Law, N. Constant, G. H. Abrego, S. Yuan, C. Tar, Y.-H. Sung, B. Strope, and R. Kurzweil, "Multilingual universal sentence encoder for semantic retrieval," 2019.

[73] K. Abdalgader and A. Skabar, *Short-Text Similarity Measurement Using Word Sense Disambiguation and Synonym Expansion*. Springer Berlin Heidelberg, 2010.

[74] R. Cheong and R. Daniel, "transformers.zip: Compressing Transformers with Pruning and Quantization," *Technical report, Stanford University, Stanford*, 2019.

Annexes

Code Listing 1 – Script de Kappa

```
import pandas as pd
from sklearn.metrics import cohen_kappa_score

def calculate_kappa(list1, list2):
  """
  To calculate the coef Kappa.

  :param list1: Labels assigned by the first annotator.
  :param list2: Labels assigned by the second annotator.
  :return: The Kappa result
  """

  return cohen_kappa_score(list1, list2)

def get_labels(df):
  """
  To get the labels from csv file.

  :param df: dataframe containing the labels.
  :return: the list of labels
  """

  labels =df.Pertinence1.values.tolist() +df.Pertinence2.values.tolist() +df.Pertinence3.values.tolist()

  return labels

```

```
csv_of_first_annotator =[PATH_1] # you have to change it
csv_of_second_annotator =[PATH_2] # you have to change it

df_first_annotator =pd.read_csv(csv_of_first_annotator, encoding="utf-8")
df_second_annotator =pd.read_csv(csv_of_second_annotator, encoding="utf-8")

labels_first_annotator =get_labels(df=df_first_annotator)
labels_second_annotator =get_labels(df=df_second_annotator)

result_kappa =calculate_kappa(list1=labels_first_annotator, list2=labels_second_annotator)

print(f"Result of Kappa : {result_kappa}")
```

Code Listing 2 – Script de la visualisation du système d'attention et de vérification de similarité

```
from transformers import BertTokenizer, BertForSequenceClassification
from transformers import BertTokenizer, BertModel
from bertviz import head_view

import torch
import pandas as pd
import numpy as np
import time

path_model =#[PATH_BERT_MODEL]

# for inference
device =torch.device("cuda" if torch.cuda.is_available() else "cpu")
model =BertForSequenceClassification.from_pretrained(path_model)
tokenizer =BertTokenizer.from_pretrained(path_model)
model.to(device)

# for visualisation
do_lower_case =False
model_vis =BertModel.from_pretrained(path_model, output_attentions=True)
tokenizer_vis =BertTokenizer.from_pretrained(path_model,
                                        do_lower_case=do_lower_case)

def check_similarity(sent1, sent2):
  """
  To check the similarity between two sentences.
  :param sent1: First sentence
  :param sent2: Second sentence
  :return: class predicted and probability
  """
    encoded_dict =tokenizer.encode_plus(
                            sent1,
                            sent2,
```

```
                                add_special_tokens=True,
                                max_length=512,
                                pad_to_max_length=False,
                                return_attention_mask=True,
                                truncation=True,
                                return_tensors= "pt")


    input_ids =encoded_dict["input_ids"].to(device)
    segment_ids =encoded_dict["token_type_ids"].to(device)
    attention_mask =encoded_dict["attention_mask"].to(device)

    model.eval()
    with torch.no_grad():
        outputs =model(input_ids,
                       token_type_ids=segment_ids,
                       attention_mask=attention_mask)

    out = outputs[0]
    # Logits to classification label
    out_label =out.detach().cpu().numpy()
    logits_after_softmax =np.argmax(out_label, axis=1).flatten()
    logits_to_probability =torch.sigmoid(out)
    proba =float(logits_to_probability[0][1])
    return logits_after_softmax[0], proba

def call_html():
    import IPython
    display(IPython.core.display.HTML('''
        <script src="/static/components/requirejs/require.js"></script>
        <script>
          requirejs.config({
            paths: {
              base: '/static/base',
              "d3": "https://cdnjs.cloudflare.com/ajax/libs/d3/3.5.8/d3.min",
              jquery: '//ajax.googleapis.com/ajax/libs/jquery/2.0.0/jquery.min',
```

```
            },
          });
        </script>
        '''))

def show_head_view(model, tokenizer, sentence_a, sentence_b=None):
    """
    To visaulise the attention of BERT.
    :param model: BERT model
    :param tokenizer: BERT tokenizer
    :param sentence_a: First sentence
    :param sentence_b: Second sentence
    """
    inputs =tokenizer.encode_plus(sentence_a, sentence_b, return_tensors='pt', add_special_tokens=True)
    input_ids =inputs['input_ids']
    if sentence_b:
        token_type_ids =inputs['token_type_ids']
        attention =model(input_ids, token_type_ids=token_type_ids)[-1]
        sentence_b_start =token_type_ids[0].tolist().index(1)
    else:
        attention =model(input_ids)[-1]
        sentence_b_start =None
    input_id_list =input_ids[0].tolist() # Batch index 0
    tokens =tokenizer.convert_ids_to_tokens(input_id_list)
    call_html()
    head_view(attention, tokens, sentence_b_start)

sentence_a ="J'ai mange une orange."
sentence_b ="La couleur de la voiture est orange."

# to visualise the attention of BERT
show_head_view(model_vis, tokenizer_vis, sentence_a, sentence_b)

# to check the similarity between two sentences in input
result =check_similarity(sentence_a, sentence_b)

```

```
print(
    f"{sentence_a}"
    f"\n\n{sentence_b}"
    f"\n\nSimilarite : {result[0]}"
)
```

Annexe 3

Comme nous pouvons le voir dans la figure [27], BERT est capable de faire un lien entre le mot **tracking** et celui **traceur**.

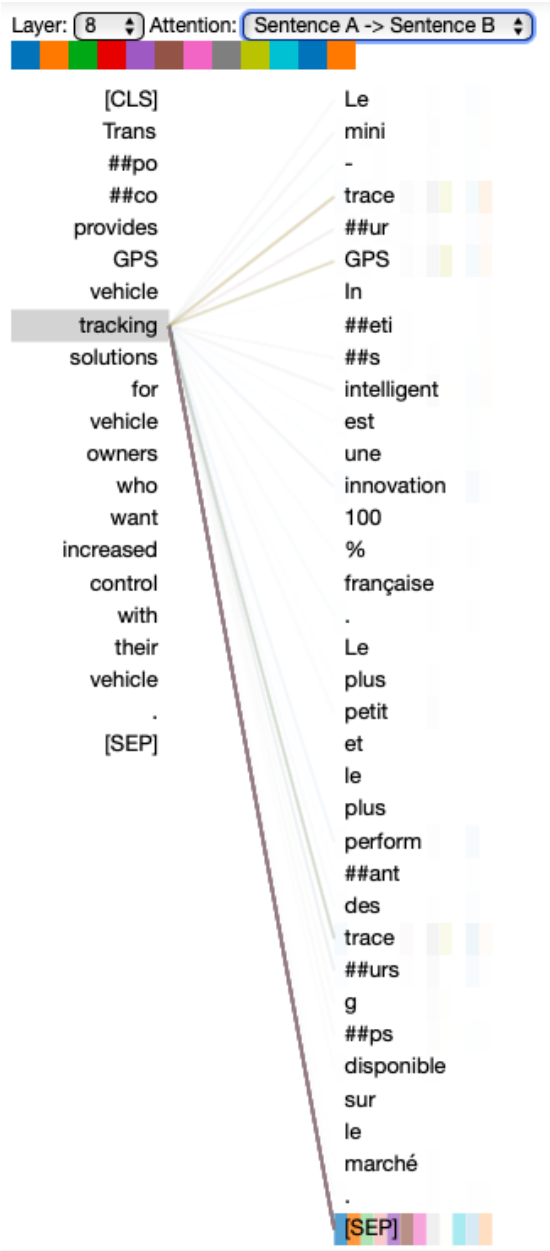


FIGURE 27 – Exemple d'une paire de phrases multilingue avec le système d'attention de BERT.