



الجمهوريـة الـديمقـراطـية الشـعـبـيـة

وزارـة التعليم العـالـي و الـبحـث العـلـمي

جـامـعـة وـهـرـان لـلـعـلـوم وـالـتـكـنـوـلـوـجـيا مـحـمـد بـوـضـيـف

كـلـيـة الـرـيـاضـيـات وـالـاعـلـام الـإـلـيـ

République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur Et de la Recherche Scientifique

Université des Sciences et de la Technologie d'Oran Mohamed BOUDIAF

Faculté des Mathématiques et Informatique

Département : Informatique

Mémoire de fin d'études

*Pour l'obtention du diplôme
de Master*

Deep Learning : natural language process for translating

Domaine : Mathématiques - Informatique

Filière : Informatique

Spécialité : SID

Présenté le : 09/07/2019

Par :

- BENAFFANE Yacine

Jury

Nom et Prénom

Grade

Université

Président NAIT BAHLOUL Sarah

Encadrant BENALLAL Mohamed ANIS

Examinateur BELAID Mohammed Said

Examinateur MOUSSA Hadjer

2018/2019

Remerciements

Aux membres du jury

Je tiens à remercier les membres du Jury d'avoir pris le temps de lire le mémoire, et m'ont fait l'honneur de bien vouloir étudier avec attention le travail : NAIT BAHLOUL Sarah, BELAID Mohammed Said, et MOUSSA Hadjer, et merci d'avoir accepté l'invitation pour participer en tant que Jury.

A mon encadreur

Mr BENALLAL MOHAMED ANIS

Je vous remercie pour votre accueil, vos conseils ainsi que votre patience, et surtout votre confiance en moi. Grâce à vous, j'ai pu réaliser un projet d'une grande envergure, et qui est nouveau dans le domaine de recherche. Veuillez trouver ici, l'expression de ma gratitude.

A mes amis

Je tiens à remercier mes amis pour leur présence, leur soutien inestimable, et leurs encouragements.

Dédicace

A Ma mère, qui a œuvré pour ma réussite, de par son amour, son soutien, tous les sacrifices consentis et ses précieux conseils, pour toute son assistance et sa présence dans ma vie.

Grace à elle, je suis devenu ce que je suis aujourd'hui.

A mon frère et mes sœurs qui n'ont cessé d'être pour moi des exemples de persévérance, de courage et de générosité.

Résumé

by Yacine BENAFFANE

La traduction automatique ne cesse d'évoluer. Elle a permis au monde d'avancer que ce soit dans le domaine du commerce ou du business. Elle a facilité la donne pour de nombreuses tâches, et elle a pu relier des personnes issues de différents pays ayant différentes langues.

L'objectif de ce projet consiste à accomplir la tâche de la traduction automatique de l'anglais vers le français en utilisant une nouvelle architecture de pointe qui est le Transformer. Les questions qui se posent, en quoi cette technologie est si spéciale par rapport aux précédentes ? Pourquoi avoir créer cette architecture ? Les architectures actuelles ne sont-elles pas performantes ? Quelles sont les solutions qu'apporte le Transformer ?

Cette technologie, qu'est le Transformer a suscité la curiosité de nombreux chercheurs, car elle a permis d'atteindre des niveaux de qualité et de performance très haut, surpassant ceux utilisés par Google Traduction ainsi que Facebook. Ceux-ci possédaient des lacunes néfastes et ont été corrigés par le Transformer, tout en offrant de bien meilleurs résultats. Le Transformer se base sur un mécanisme déjà utilisé par les anciennes architectures, mais en meilleur. Celui-ci se nomme l'auto-attention. Cette découverte a ouvert un nouvel état de l'art concernant le traitement du langage naturel, les gens peuvent créer de nouveaux modèles innovants sur le Transformer. Ainsi ce dernier sera le sujet de recherche du traitement du langage naturel en 2019, ou on verra beaucoup d'approches se basant sur celui-ci.

Le résultat obtenu n'a pas atteint l'état de l'art du Transformer, encore moins de Facebook et de Google Traduction, sauf qu'il a été obtenu après une courte durée de formation, alors qu'en utilisant les anciennes approches telles que celle de Google Traduction, le résultat aurait été encore moins meilleur, d'où la puissance du Transformer qui réduit grandement le coût d'entraînement.

Abstract

by Yacine BENAFFANE

Machine translation is constantly evolving. It has allowed the world to advance whether in the field of trade or business. She made it easy for many tasks, and she was able to connect people from different countries with different languages.

The goal of this project is to accomplish the task of automatic translation from English to French using a new advanced architecture that is the Transformer. The questions that arise, why is this technology so special compared to previous ones ? Why did you create this architecture ? Are not the current architectures performing well ? What solutions does the Transformer bring ?

This technology, which is the Transformer has aroused the curiosity of many researchers, because it has achieved levels of quality and performance very high, surpassing those used by Google Translation and Facebook. These had bad deficiencies and were corrected by the Transformer, while offering much better results. The Transformer is based on a mechanism already used by older architectures, but better. This is called self-attention. This discovery has opened a new state of the art regarding natural language processing, people can create new and innovative models on the Transformer. So this last one will be the subject of research of the natural language processing in 2019, where we will see many approaches based on this one.

The result did not reach the state of the art of the Transformer, let alone Facebook and Google Translate, except that it was obtained after a short training period, while using the old approaches such than that of Google Translate, the result would have been even less better, hence the power of the Transformer which greatly reduces the cost of training.

Table des matières

Introduction général	1
1 Chapitre d'état de l'art	2
1.1 Introduction	3
1.2 Traitement automatique du langage naturel	4
1.2.1 Compréhension humaine et informatique de la langue	4
1.2.2 Qu'est-ce que le traitement du langage naturel?	4
1.2.3 Composants du TLN	5
1.2.4 L'importance du TLN	5
1.2.5 L'impact du TLN sur le monde	7
1.2.6 Domaine d'applications du TLN	8
1.2.7 Le fonctionnement du TLN	9
1.2.8 Pourquoi le TLN est difficile?	9
1.2.9 Les difficultés rencontrées pour le TLN	10
Les défis pour comprendre le texte	10
1.2.10 Les techniques utilisées en TLN	12
1.2.11 Construction d'un pipeline du TLN	13
Segmentation des phrases	13
Prétraitement	14
Structuration	17
Analyse	17
Transformation	17
1.2.12 L'importance de la traduction automatique	17
1.2.13 Le pipeline de construction d'un modèle de traduction	18
1.3 La traduction automatique statistique (SMT)	19
1.3.1 Penser aux probabilités	19
1.3.2 Les limites de la traduction automatique statistique	19
1.4 La traduction automatique neuronale	20
1.4.1 Le modèle séquence a séquence	21
1.4.2 L'encodeur	22
1.4.3 Décodeur	23
1.4.4 Une pile d'encodeur et de décodeur	23
1.4.5 L'Attention visuelle	23
1.4.6 Le mécanisme de l'attention	24
1.4.7 L'impact du mécanisme de l'attention	24
1.5 Exemple de but recherché	25
1.5.1 Google neural machine translation GNMT	25
Les réseaux de neurones récurrents	26
L'attention et GNMT	26

	Puissance des LSTM	28
	Avantage	28
1.5.2	Facebook	28
	Pourquoi des réseaux de neurones convolutifs ?	29
	Meilleure traduction avec multi-hop attention et gating	29
	Explication	29
	Avantage	31
1.6	Les Transformers	32
1.6.1	Une architecture dominante	32
1.6.2	Perte d'informations	32
1.7	Conclusion	33
2	Chapitre de conception	34
2.1	Introduction	35
2.2	Comparaison entre les architectures existantes	36
2.2.1	L'association de la linguistique informatique	36
2.2.2	Les architectures actuelles	36
2.2.3	Les Transformers	37
2.2.4	Exactitude et efficacité dans la compréhension du langage	37
2.2.5	Solution du Transformer	38
2.2.6	L'attention avec le Transformer	38
2.2.7	Traitements avec RNN	40
2.2.8	Traitements avec CNN	40
2.2.9	Dépendances à long termes	41
2.2.10	Les difficultés rencontrées avec le modèle Seq2Seq	41
2.2.11	Les solutions apportées du Transformer	42
2.2.12	Comparaison de complexité	42
2.2.13	Comparaison de résultat entre différentes architectures	43
2.3	L'auto-attention	44
2.3.1	L'attention, c'est tout ce dont vous avez besoin	44
2.3.2	Les points importants du papier de recherche	44
2.3.3	L'importance du transformer	44
2.3.4	Le Transformer	46
2.3.5	La couche Encodeur	47
2.3.6	La couche Décodeur	48
	Exemple d'attention masquée	50
2.3.7	Inférence (prédiction)	52
2.3.8	L'auto-attention détaillée	52
	Problème avec une seul attention	53
2.3.9	L'attention multi-tête	55
2.3.10	L'application de l'auto-attention	57
2.3.11	Connexion résiduelle	58
2.3.12	Position-wise Feed-Forward Networks	58
2.3.13	Normalisation	59
2.3.14	Incorporation et Softmax	59
2.3.15	La puissance du Transformer	60
2.3.16	L'encodage des positions	60

Pourquoi sinus et cosinus	62
2.3.17 Analyse et tests	63
2.4 L'architecture du Transformer	64
2.4.1 L'incorporation	64
2.4.2 L'encodage des positions	64
Le motif	65
2.4.3 Le premier encodeur	66
Exemple	66
2.4.4 Le calcul de L'attention personnelle	67
Première étape	67
Deuxième étape	67
Troisième étape	68
Quatrième étape	69
Cinquième étape	69
Sixième étape	70
2.4.5 Calcul matriciel de l'attention personnelle au niveau du mot	70
Attention additive	71
Remarque	71
2.4.6 L'attention multiple	72
2.4.7 Les résidus	74
2.4.8 Le décodeur	75
2.4.9 La couche finale linéaire et Softmax	77
2.4.10 Résumé de la formation	78
2.4.11 La fonction de perte	79
Exemple plus compliquer	80
2.4.12 Type de prédiction	81
Décodage glouton	81
Recherche de faisceau (beam search)	81
2.5 La fonction de perte et l'optimiseur	83
2.5.1 L'entropie croisée	83
2.5.2 Logits	84
2.5.3 Softmax	85
2.5.4 Déivation	85
2.5.5 L'optimiseur Adam	86
Avantage	86
Paramètre d'adam	86
2.6 Le score BLEU	87
2.6.1 Vérification de modèle	87
Le BLEU	87
2.7 Le calcul de BLEU	87
2.7.1 Combinaison du score BLEU	88
2.7.2 Longueur optimale	88
2.7.3 Calcul final du BLEU	88
2.8 Représentation vectorielle	89
2.8.1 One-Hot encodage	89
L'importance De l'encodage à chaud	89
2.8.2 Word Embeddings	90

Comprendre l'incorporation de mot	90
2.9 Normalisation	91
2.9.1 La normalisation des couches	91
2.9.2 Avantages	91
2.10 Conclusion	92
3 Chapitre d'implémentation	93
3.1 Introduction	94
3.2 L'implémentation	95
3.2.1 Le corpus de données et la taille du lot	95
3.2.2 Matériel et le temps d'entraînement	95
3.2.3 Les hyperparamètres	95
3.2.4 Optimiseur	95
3.2.5 L'Abandon résiduel (Residual Dropout)	95
3.2.6 Lissage des étiquettes (Label smoothing)	95
3.2.7 Entrainement	96
3.2.8 Prédiction	96
3.3 Résulat	97
3.3.1 Le BLEU	97
3.3.2 Le BLEU approximative	97
3.3.3 La perplexité	97
3.3.4 La fonction de perte	98
3.3.5 Comparaison de traductions	99
3.3.6 Visualisation de l'attention	100
3.4 Les technologies utilisées	101
3.4.1 Python	101
3.4.2 Tensorflow	101
3.4.3 Tensor2Tensor (T2T)	102
3.5 Conclusion	103
Conclusion général	104
Bibliographie	106

Table des figures

1.1	Traitement automatique du langage naturel	4
1.2	Langage naturel non structuré avec google	6
1.3	Le TLN et l'apprentissage automatique	6
1.4	Nombre de publications contenant la phrase « traitement du langage naturel » dans PubMed au cours de la période 1978-2018. En 2018, PubMed comptait plus de 29 millions de citations dans la littérature biomédicale	7
1.5	Exemple d'application du domaine TLN	8
1.6	Les différents contextes d'une phrase	10
1.7	Des termes différents	11
1.8	Exemple de Coréférence	11
1.9	Le fonctionnement du TLN (Référence : RESPENSABLE, 28 Octobre 2018)	12
1.10	La pipeline du TLN (Référence : SATHIYAKUGAN, 24 Juillet 2018)	13
1.11	Les étapes du pré-traitement (Référence : « Introduction au NLP (Partie II) »)	14
1.12	Tokenisation	15
1.13	Retrait des Stopwords	15
1.14	Stemmatisation	16
1.15	Lemmatisation	16
1.16	Différentes approches de représentation vectorielle (Référence : SHAFFY, 8 Novembre 2017)	17
1.17	Pipeline de construction d'un modèle de traduction	18
1.18	Historique de la traduction automatique	19
1.19	Deep learning vs le TLN Classique (Référence : « Overview of Artificial Intelligence and Natural Language Processing »)	20
1.20	Qualité de traduction avec les différentes méthodes (Référence : « Neural Machine Translation »)	21
1.21	Encodeur et décodeur	22
1.22	Séquence à séquence (Référence : KEITAKURITA, 29 décembre 2017)	22
1.23	Des piles d'encodeurs et décodeurs	23
1.24	Logo de Google translate	25
1.25	GNMT comparaison (Référence : AI, Mardi, 27 September 2016) . .	25
1.26	Encodage et décodage avec GMNT	27
1.27	Résultat d'encodage et décodage avec GMNT	27
1.28	Exemple de traduction produite à partir d'un site de nouveautés (Référence : AI, Mardi, 27 September 2016)	28
1.29	Traduction d'une phrase en anglais avec les CNN partie 1	30
1.30	Traduction d'une phrase en anglais avec les CNN partie 2	30

1.31	Traduction d'une phrase en anglais avec les CNN partie finale	31
2.1	Scores BLEU de modèles individuels sur la traduction standard WMT 2014 anglais-français (Référence : AI, Jeudi, 31 août 2017)	37
2.2	Traduction de l'anglais vers le français (Référence : AI, Jeudi, 31 août 2017)	39
2.3	La distribution de l'attention du mot « it » avec un Transformeur (Référence : AI, Jeudi, 31 août 2017)	39
2.4	Résultat de traduction avec différents systèmes de l'anglais vers le français (WMT 14, newstest2014) (Référence : AI, 12 Juin 2017)	43
2.5	Recherche sur les mécanismes d'attention (Référence : ZHANG, 18 Septembre 2018)	44
2.6	Architecture du Transformer (Référence : CHROMIAK, Mardi, 12 Septembre 2017)	46
2.7	Une pile de six encodeurs et de décodeurs (Référence : ALAMMAR, 20 Février 2017)	47
2.8	Les composants d'une couche d'encodeur (Référence : ALAMMAR, 20 Février 2017)	48
2.9	Les composants d'une couche de décodeur sans l'attention masquée (Référence : ALAMMAR, 20 Février 2017)	48
2.10	La bonne façon de former un décodeur (Référence : KEITAKURITA, 29 décembre 2017)	49
2.11	La mauvaise façon de former un décodeur (Référence : KEITAKURITA, 29 décembre 2017)	50
2.12	Représentation de mot dans le décodeur sans masquage	51
2.13	Représentation de mots dans le décodeur sans attention	51
2.14	Représentation de mots dans le décodeur avec masquage	51
2.15	Produit scalaire pour le calcul de l'attention (Référence : KEITAKURITA, 29 décembre 2017)	53
2.16	Une seule attention (Référence : BRIJESH, 09 Décembre 2018)	53
2.17	Encodage du mot « it » dans l'encodeur n5 (Référence : ALAMMAR, 20 Février 2017)	54
2.18	L'attention multi-têtes consiste en h couches d'attention fonctionnant en parallèle (Référence : KEITAKURITA, 29 décembre 2017)	55
2.19	Multi-Head attention (Référence : BRIJESH, 09 Décembre 2018)	55
2.20	Concentration sur le mot « it » sur 2 têtes (Référence : ALAMMAR, 20 Février 2017)	56
2.21	Concentration sur le mot « it » sur 8 têtes. La référence est plus difficile à déterminer. (Référence : ALAMMAR, 20 Février 2017)	56
2.22	Encodeur auto-attention : tout regarde tout	57
2.23	Encodeur-Décodeur attention	57
2.24	Décodeur auto-attention	58
2.25	Connexion résiduelle (Référence : RICARDOKLEINKLEIN, 16 Novembre 2017(c))	58
2.26	Réseau feedforward du transformer basique	59
2.27	Exemple d'encodage du Transformer	60
2.28	Matrice d'encodage de position (Référence : DENOYES, 10 Avril 2019)	61
2.29	Signaux sinusoïdaux (Référence : DENOYES, 10 Avril 2019)	62

2.30	Une série de test concernant l'architecture Transformer de l'anglais vers l'allemand (Référence : AI, 12 Juin 2017)	63
2.31	Intégration des mots en des vecteurs/tenseurs (en réalité une taille de 512)	64
2.32	Utilisation de l'encodage positionnel	64
2.33	Un exemple réel d'encodage de position avec une incorporation d'une taille de 4	65
2.34	Exemple réel d'un encodage de position	65
2.35	Le premier encodeur	66
2.36	La sortie du premier encodeur	66
2.37	Les différents vecteurs nécessaires pour l'auto-attention	67
2.38	Calcul du score pour le mot « je »	68
2.39	Score divisé par la valeur 8	68
2.40	Softmax appliquée	69
2.41	Multiplication du score Softmax par le vecteur de valeur	69
2.42	Résumé des vecteurs de valeurs pondérées	70
2.43	Calcul des matrices de requêtes, clés et de valeurs	70
2.44	L'attention multiple	72
2.45	Concaténation des têtes d'attention	72
2.46	Les étapes de l'attention multiple	73
2.47	Connexion résiduelle	74
2.48	Visualisation des vecteurs et d'opération couche-norme associées à l'attention propre	74
2.49	Un Transformer composé de deux encodeurs et décodeurs avec la normalisation de couches	75
2.50	Le décodeur lors du premier pas	75
2.51	Le décodeur lors du quatrième pas	76
2.52	Décodage complet	77
2.53	Application de la couche linéaire et Softmax	78
2.54	Exemple de vocabulaire de sortie	78
2.55	Exemple d'encodage à chaud (one-hot)	79
2.56	Sortie d'un modèle non entraîné	79
2.57	Encodage one-hot avec plusieurs mots	80
2.58	Distribution de probabilités avec plusieurs mots	81
2.59	L'entropie croisée	83
2.60	Plage de valeur de l'entropie croisée (Référence : MACHINE LEARNING CHEATSHEET MACHINE LEARNING CHEATSHEET, 2017)	84
2.61	Logit	85
2.62	Deux techniques utilisées pour la représentation vectorielle	89
2.63	One-hot encodage représentation (Référence : SHAFFY, 8 Novembre 2017)	89
2.64	Exemple d'incorporation de mots (Référence : SHAFFY, 8 Novembre 2017)	90
3.1	Code pour entraîner le modèle	96
3.2	Code pour la prédiction	96
3.3	Le BLEU approximative	97
3.4	La perplexité	97

3.5	La fonction de perte	98
3.6	Comparaison de traductions	99
3.7	L'attention visuelle avec 16 têtes dans la couche 5	100
3.8	L'attention visuelle avec 16 têtes dans la couche 5	100
3.9	Python logo	101
3.10	TensorFlow logo	101

Liste des tableaux

2.1	Tableau de complexités (Référence : AI, 12 Juin 2017)	42
-----	---	----

Introduction général

Le traitement automatique du langage naturel ou TALN est un domaine de l'intelligence artificielle qui donne aux machines la capacité de lire, de comprendre et de tirer du sens des langages humains. Tout ce que nous exprimons (verbalement ou par écrit) contient d'énormes quantités d'informations. Le sujet que nous choisissons, notre ton, notre choix de mots, tout ajoute un certain type d'informations pouvant être interprétées et une valeur extraite de celles-ci. En théorie, nous pouvons comprendre et même prédire le comportement humain en utilisant ces informations.

Les ordinateurs travaillent parfaitement avec des données normalisées et structurées telles que des tables de bases de données, sauf que les humains, ne communiquent pas en « données structurées » et ne parlent pas en binaires. Mais cela se fait à l'aide de mots, une forme de données non structurées.

Les ordinateurs ont du mal à travailler avec des données non structurées car il n'existe pas de techniques normalisées pour les traiter. Avec des données non structurées, ces règles sont assez difficiles à définir concrètement. Une personne peut générer des centaines ou des milliers de mots dans une déclaration, chaque phrase avec la complexité correspondante.

Grâce aux avancées dans des disciplines telles que l'apprentissage automatique, une grande révolution est en cours dans ce domaine. De nos jours, il ne s'agit plus d'essayer d'interpréter un texte ou un discours en fonction de ses mots-clés (l'ancienne méthode mécanique), mais de comprendre le sens de ces mots (la méthode cognitive). De cette façon, il est possible de détecter des figures de style comme l'ironie, ou même d'effectuer une analyse de sentiment.

Aujourd'hui, la traduction automatique connaît une très grande avancée, en utilisant l'apprentissage en profondeur, plusieurs applications de traduction qui offrent de très bon résultat existent, tels que Google Translate.

Dans ce cadre s'inscrit le projet de fin d'études qui portera sur la traduction automatique de l'anglais vers le français en utilisant une nouvelle architecture. L'objectif de ce projet est de faire des recherches, et de tester une nouvelle architecture de traduction automatique afin de comprendre son fonctionnement et de comparer les résultats de qualité, ainsi que de performance obtenue par celle-ci.

Le premier chapitre « Etat de l'art », est un chapitre introductif où je présenterai l'importance du TLN, ainsi que son fonctionnement, et les difficultés rencontrées, on verra ensuite les technologies déjà existantes qui ont fait avancer à grande échelle le monde du TLN. Et Enfin l'architecture qui sera la nouvelle tendance de recherche.

Concernant le deuxième chapitre intitulé « Conception », j'étendrai la représentation de l'architecture étudiée, avec une comparaison entre elle et les autres architectures existantes, ainsi que le corpus utilisé, puis on parlera des différentes méthodes telles que la métrique d'évaluation, la fonction de perte et d'autres algorithmes qui seront cruciaux pour le bon fonctionnement de l'architecture.

Enfin, nous nous intéresserons au troisième chapitre « Implémentation ». On montrera les différents hyperparamètres et les optimisations utilisées avec la configuration du modèle. Puis les différentes technologies utilisées seront présentées. Le rapport sera clôturé par une conclusion générale où un résumé sur le travail sera fait avec les connaissances acquises grâce à ce projet.

Chapitre d'état de l'art

1.1 Introduction

Ce chapitre se compose de plusieurs parties. La première partie, a pour objectif de présenter le traitement automatique du langage naturel (TALN), son impact, son processus, ainsi que les difficultés rencontrées. Ensuite vient la deuxième partie, on présentera une ancienne méthode de la traduction automatique et sa limite. Dans le chapitre qui suit, on parlera de la traduction automatique moderne, qui est la traduction automatique neuronale ou neural machine translation (NMT). Ensuite vient le chapitre, où on présentera des différentes architectures existantes, en citant leurs avantages. Puis vient la partie où on parlera brièvement de l'architecture Transformer. Et on terminera ce chapitre avec une conclusion.

1.2 Traitement automatique du langage naturel

1.2.1 Compréhension humaine et informatique de la langue

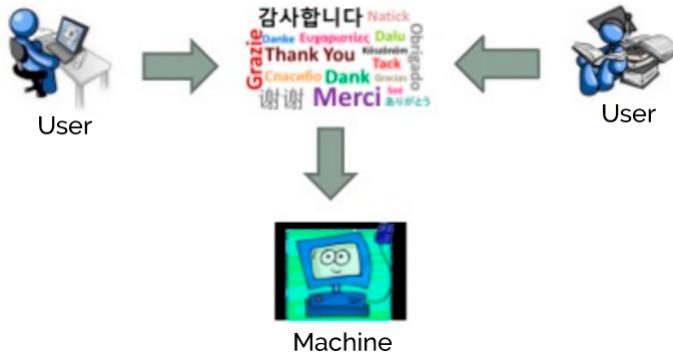


FIGURE 1.1 – Traitement automatique du langage naturel

Les humains écrivent des choses depuis des milliers d'années. Au cours de cette période, le cerveau humain a acquis une énorme quantité d'expérience dans la compréhension du langage naturel. Lors de la lecture de quelque chose d'écrit sur un bout de papier ou dans un article de blog sur Internet, la compréhension et la signification réelle de celui-ci est possible.

Le traitement automatique du langage naturel (TALN) ou natural language process (en anglais, NLP) est un sous-domaine de l'intelligence artificielle qui vise à permettre aux ordinateurs de comprendre et de traiter les langages humains, afin de les rapprocher de la compréhension du langage par l'homme. Les ordinateurs ne possèdent pas encore la même compréhension intuitive du langage naturel que les humains. Ils ne peuvent pas vraiment comprendre ce que la langue essaie vraiment de dire.

Les récents progrès en Machine Learning (ML) ont permis aux ordinateurs de faire beaucoup de choses utiles avec le langage naturel. Le Deep Learning a permis d'écrire des programmes permettant d'exécuter des tâches telles que la traduction linguistique, la compréhension sémantique et la synthèse de texte. Toutes ces choses ajoutent une valeur réelle, facilitant la compréhension et l'exécution de calculs sur de gros blocs de texte.

1.2.2 Qu'est-ce que le traitement du langage naturel ?

Le traitement du langage naturel (TLN) est la capacité d'un programme informatique à comprendre le langage humain tel qu'il est parlé. Le TLN est une composante de l'intelligence artificielle (IA). C'est un ensemble de techniques permettant aux ordinateurs d'extraire une signification structurée et exploitable d'un langage humain naturel et non structuré. Extraire le sens d'une phrase, d'un paragraphe, d'une phrase non structurée (ou d'un livre, d'un site Web, d'un blog, etc.) est une tâche très complexe et potentiellement intensive pour un ordinateur. La signification peut différer en fonction d'un certain nombre de facteurs, tels que les mots utilisés, le contexte, le domaine, etc. Prenons cette phrase comme exemple : **Où aimerais tu aller ensuite ?**

Si on lit ce message sur votre téléphone, sa signification dépendrait du reste de la conversation avec cette personne, du contexte de cette conversation et de la situation générale dans laquelle l'interaction a eu lieu. Cette interprétation est très rapide et facile pour un humain mais difficile pour un ordinateur. Le TLN peut être utilisé pour analyser du texte dans différentes langues. Un moteur de traitement de langage naturel est généralement spécifique à chaque langue, chaque langue ayant une structure particulière de la phrase.

1.2.3 Composants du TLN

Il y a deux composants du TLN :

1. Compréhension du langage naturel (NLU)

NLU ou Natural Language Understanding essaye de comprendre le sens d'un texte donné. La nature et la structure de chaque mot dans le texte doivent être comprises pour NLU. Pour comprendre la structure, NLU tente de résoudre l'ambiguïté suivante en langage naturel :

- **Ambiguïté lexicale** : Les mots ont plusieurs sens.
- **Ambiguïté syntaxique** : Phrase comportant plusieurs arbres d'analyse
- **Ambiguïté sémantique** : Phrase à significations multiples
- **Ambiguïté anaphorique** : Phrase ou mot mentionnés précédemment mais ayant un sens différent.

Ensuite, la signification de chaque mot est comprise en utilisant des lexiques (vocabulaire) et un ensemble de règles grammaticales. Cependant, il existe certains mots différents ayant une signification similaire (synonymes) et des mots ayant plus d'une signification (polysémie).

2. Génération de Langage Naturel (NLG)

C'est le processus de production de phrases et de phrases significatives sous forme de langage naturel à partir d'une représentation interne.

- **Planification du texte** : Cela inclut la récupération du contenu pertinent de la base de connaissances.
- **Planification de la phrase** : Cela comprend le choix des mots nécessaires, la formation de phrases significatives, la définition du ton de la phrase.
- **Réalisation de texte** : Il s'agit de mapper un plan de phrase en une structure de phrase.

1.2.4 L'importance du TLN

1. Grands volumes de données textuelles

Le traitement du langage naturel permet aux ordinateurs de communiquer avec les humains dans leur propre langage et d'échelonner d'autres tâches liées au langage. Par exemple, le TLN permet aux ordinateurs de lire du texte, de l'interpréter, de mesurer le sentiment et de déterminer les parties importantes.

Les machines actuelles peuvent analyser plus de données linguistiques que les humains, sans fatigue et de manière cohérente et impartiale. Compte tenu de l'énorme quantité de données non structurées générées chaque jour, des dossiers médicaux aux médias sociaux, l'automatisation sera essentielle pour analyser efficacement les données de texte et de parole.

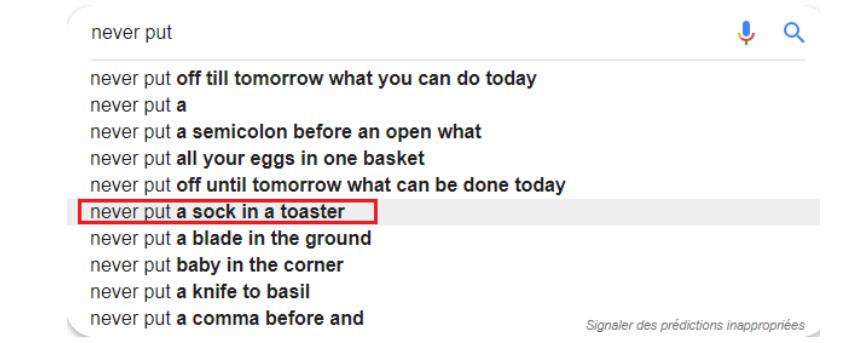


FIGURE 1.2 – Langage naturel non structuré avec google

Parfois, même Google ne sait pas ce que vous recherchez !

2. Structurer une source de données hautement non structurée

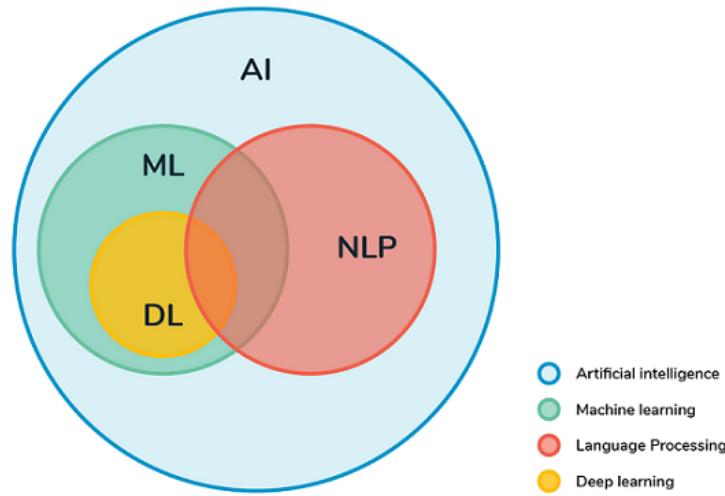


FIGURE 1.3 – Le TLN et l'apprentissage automatique

Le langage humain est incroyablement complexe et diversifié. Nous nous exprimons de manière verbale et par écrit. Il existe non seulement des centaines de langues et de dialectes, mais chaque langue comprend un ensemble unique de règles de grammaire et de syntaxe, de termes et d'argot. Lorsque nous écrivons, nous abrégeons souvent les mots, ou omettons la ponctuation. Lorsque nous parlons, nous avons des accents régionaux et nous marmonnons, empruntons des termes à d'autres langues.

Bien que l'apprentissage supervisé et non supervisé, et en particulier l'apprentissage en profondeur, soit maintenant largement utilisé pour modéliser le langage humain, il existe également un besoin de compréhension syntaxique et

sémantique et d'expertise de domaine qui ne sont pas nécessairement présents dans ces approches d'apprentissage automatique. Le TLN est important car elle aide à résoudre les ambiguïtés linguistiques et ajoute une structure numérique utile aux données de nombreuses applications, telles que la reconnaissance vocale ou l'analyse de texte.

1.2.5 L'impact du TLN sur le monde

Le TLN est particulièrement important dans le secteur de la santé. Cette technologie améliore la prestation des soins, le diagnostic des maladies et la réduction des coûts pendant que les organisations de soins de santé adoptent de plus en plus les dossiers médicaux électroniques. Le fait que la documentation clinique puisse être améliorée signifie que les patients peuvent être mieux compris et bénéficier de meilleurs soins de santé. L'objectif devrait être d'optimiser leur expérience et plusieurs organisations y travaillent déjà.

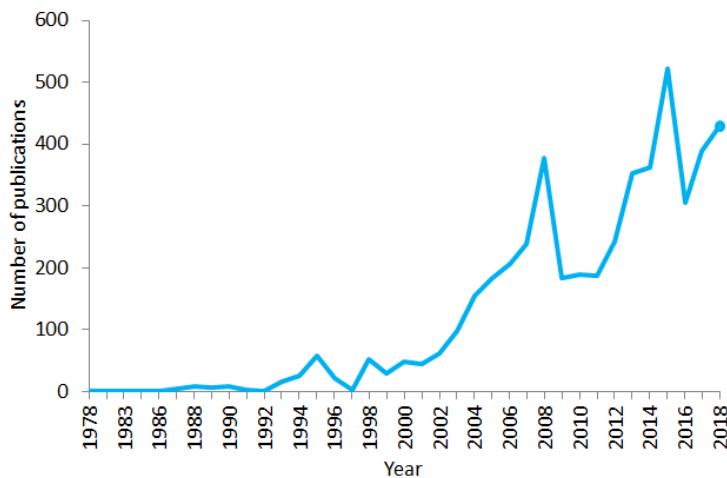


FIGURE 1.4 – Nombre de publications contenant la phrase « traitement du langage naturel » dans PubMed au cours de la période 1978-2018. En 2018, PubMed comptait plus de 29 millions de citations dans la littérature biomédicale

Des entreprises telles que Winterlight Labs (Référence : YSE, 15 Janvier) apportent des améliorations considérables au traitement de la maladie d'Alzheimer en surveillant les troubles cognitifs par la parole et peuvent également soutenir des essais cliniques et des études portant sur un large éventail de troubles du système nerveux central. Suivant une approche similaire, l'Université de Stanford a développé Woebot, un thérapeute de chatbot, dans le but d'aider les personnes souffrant d'anxiété et d'autres troubles.

Microsoft a démontré qu'en analysant un grand nombre de requêtes dans les moteurs de recherche, il était possible d'identifier les internautes atteints d'un cancer du pancréas avant même d'avoir reçu le diagnostic de la maladie. Comment les utilisateurs réagiraient-ils à un tel diagnostic ? Et que se passerait-il si votre test était un faux positif ? (Signifiant que vous pouvez être diagnostiqués avec la maladie même si

vous ne l'avez pas). Cela rappelle le cas de Google Flu Trends qui, en 2009, avait été annoncé comme étant capable de prédire la grippe mais avait ensuite disparu en raison de sa faible précision et de son incapacité à respecter les taux projetés.

1.2.6 Domaine d'applications du TLN

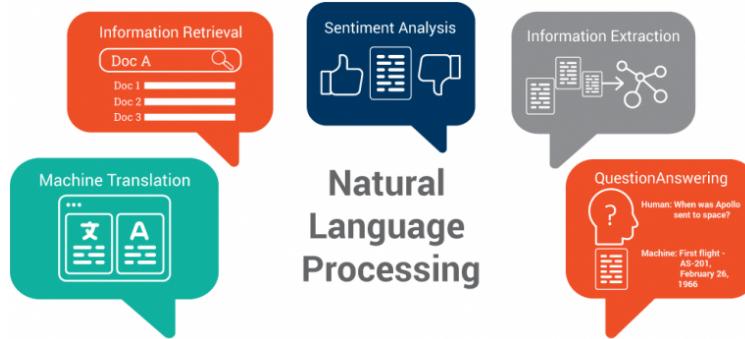


FIGURE 1.5 – Exemple d'application du domaine TLN

Le traitement du langage naturel peut être utilisé avec ces applications courantes suivantes :

- Applications de traduction de langues telles que Google Translate.
- Les logiciels de traitement de texte, tels que Microsoft Word et Grammarly pour vérifier l'exactitude grammaticale des textes.
- Applications de réponse vocale interactive utilisées dans les centres d'appels pour répondre aux demandes de certains utilisateurs.
- Applications de réponse vocale interactive utilisées dans les centres d'appels pour répondre aux demandes de certains utilisateurs.
- Générateur de texte qui égale un niveau de presse.
- Permettre de faire des débats avec des personnes politiques.
- Identifier l'humeur ou les opinions subjectives dans de grandes quantités de texte, y compris l'extraction de sentiments et d'opinions moyennes. Les organisations peuvent maintenant déterminer ce que les clients disent d'un service ou d'un produit en identifiant et en extrayant des informations dans des sources telles que les médias sociaux. Cette analyse des sentiments peut fournir de nombreuses informations sur les choix des clients et leurs facteurs de décision.
- Créer des idées de publicité pour certains produits.
- Capturer avec précision le sens et les thèmes des collections de texte et appliquer des analyses avancées au texte, telles que l'optimisation et les prévisions.
- Un résumé de document basé sur la langue, comprenant la recherche et l'indexation, les alertes de contenu et la détection de duplication.

- Des entreprises comme Google et Microsoft filtrent et classifient les e-mails avec le TLN en analysant le texte des e-mails qui transitent par leurs serveurs et en bloquant le spam avant même d'entrer dans la boîte de réception.
- Le TLN permet la reconnaissance et la prévision des maladies sur la base des dossiers de santé électroniques et du discours du patient. Cette capacité est à l'étude dans des conditions de santé allant des maladies cardiovasculaires à la dépression et même à la schizophrénie. Par exemple, Amazon Comprehend Medical est un service qui utilise le TLN pour extraire des maladies, des médicaments et les résultats de traitements de notes de patients, de rapports d'essais cliniques et d'autres dossiers médicaux électroniques.

1.2.7 Le fonctionnement du TLN

Le traitement du langage naturel comprend de nombreuses techniques différentes d'interprétation du langage humain, allant des méthodes d'apprentissage statistique et automatique aux approches basées sur des règles et des algorithmes. Les modèles d'apprentissage en profondeur nécessitent des quantités énormes de données étiquetées sur lesquelles former et identifier les corrélations pertinentes, et l'assemblage de ce type de données volumineuses est l'un des principaux obstacles du TLN à l'heure actuelle.

Les approches antérieures du TLN impliquaient une approche davantage basée sur des règles, dans laquelle des algorithmes d'apprentissage automatique plus simples indiquaient les mots et les phrases à rechercher dans le texte et donnaient des réponses spécifiques lorsque ces expressions apparaissaient. Mais l'apprentissage en profondeur est une approche plus souple et intuitive dans laquelle les algorithmes apprennent à identifier l'intention de leurs locuteurs à partir de nombreux exemples, un peu comme si l'enfant apprendrait le langage humain.

Les tâches de base du TLN incluent la création de jetons et l'analyse syntaxique, la lemmatisation / extraction, le balisage partiel de la parole, la détection du langage et l'identification des relations sémantiques. Il implique l'application d'algorithmes pour identifier et extraire les règles de langage naturel, de sorte que les données de langage non structurées soient converties en une forme pouvant être comprise par les ordinateurs. Une fois le texte fourni, l'ordinateur utilisera des algorithmes pour extraire la signification associée à chaque phrase et en collecter les données essentielles.

1.2.8 Pourquoi le TLN est difficile ?

Le langage humain est spécial pour plusieurs raisons. Il est spécifiquement construit pour transmettre le sens du locuteur/écrivain. C'est un système complexe, même si les petits enfants peuvent l'apprendre assez rapidement. Une autre chose remarquable à propos du langage humain est qu'il s'agit uniquement de symboles.

Comprendre le langage humain est considéré comme une tâche difficile en raison de sa complexité. Par exemple, il existe un nombre infini de façons différentes d'arranger les mots dans une phrase. De plus, les mots peuvent avoir plusieurs significations et des informations contextuelles sont nécessaires pour interpréter correctement les phrases. Chaque langue est plus ou moins unique, ambiguë, et des phrases

qui possèdent des interprétations très différentes, ce qui est un des défis posés par le TLN.

Une compréhension parfaite de la langue par un ordinateur donnerait une IA capable de traiter toute l'information disponible sur Internet, ce qui entraînerait probablement une intelligence artificielle générale.

1.2.9 Les difficultés rencontrées pour le TLN

Le traitement du langage naturel est considéré comme un problème difficile en informatique. C'est la nature du langage humain qui le rend si difficile.

Les règles qui dictent la transmission d'informations à l'aide de langues naturelles ne sont pas faciles à comprendre pour les ordinateurs. Certaines de ces règles peuvent être de haut niveau et abstraites, par exemple, lorsque quelqu'un utilise une remarque sarcastique pour transmettre des informations. Aussi, certaines de ces règles peuvent être peu élevées, par exemple, en utilisant le caractère "s" pour signifier la pluralité d'éléments.

Pour comprendre le langage humain de manière globale, il faut comprendre à la fois les mots et la manière dont les concepts sont connectés pour transmettre le message souhaité. Alors que les humains peuvent facilement maîtriser un langage, ce sont les ambiguïtés et les caractéristiques imprécises des langages naturels qui rendent le TLN difficile à mettre en œuvre par les machines.

Les défis pour comprendre le texte

Les problèmes cités proviennent de : (Référence : SATHIYAKUGAN, 24 Juillet 2018).

Ambiguïté : Le principal défi du TLN consiste à comprendre et à modéliser des éléments dans un contexte variable. En langage, les mots sont uniques mais peuvent avoir différentes significations en fonction du contexte dans lequel ils sont évalués.

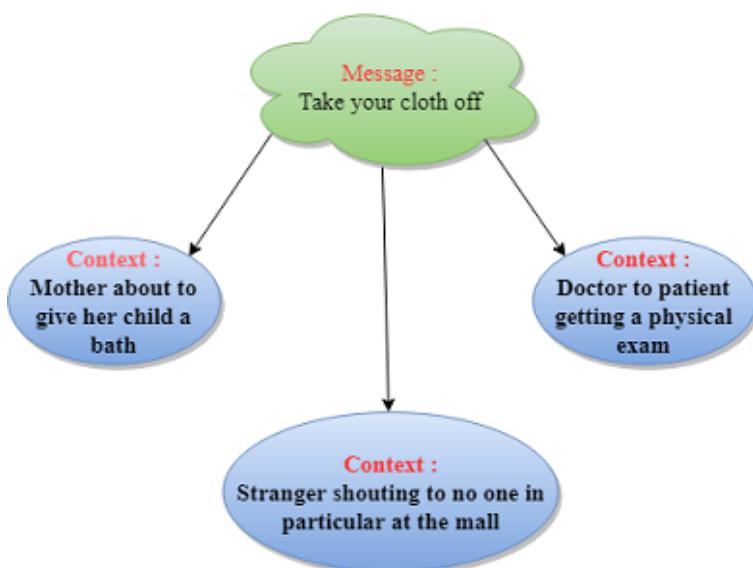


FIGURE 1.6 – Les différents contextes d'une phrase

Synonymie : Un autre phénomène clé du langage naturel est que nous pouvons exprimer la même idée avec des termes différents.

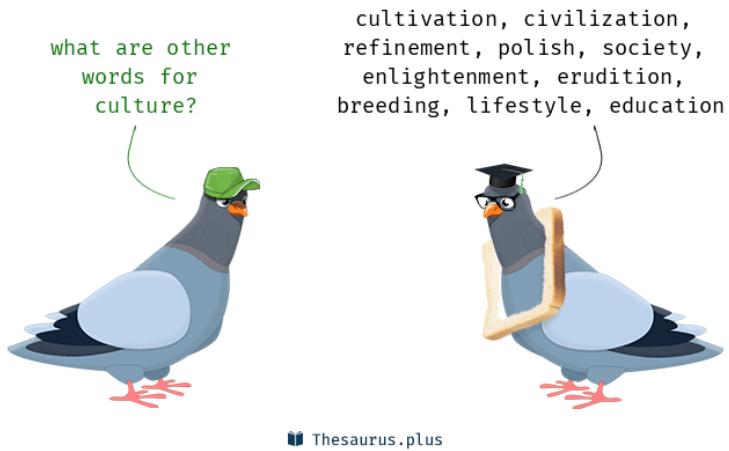


FIGURE 1.7 – Des termes différents

Syntaxe : La structure, qui prend en compte plusieurs règles mais aussi quelques irrégularités dans différents cas.

Coréférence : L'anglais est plein de pronoms, et des mots comme he, she et tout. Ce sont des raccourcis que nous utilisons au lieu d'écrire des noms à chaque fois dans chaque phrase. Les humains peuvent garder une trace de ce que ces mots représentent en fonction du contexte. Mais un modèle TLN ne sait pas ce que les pronoms veulent dire, car il n'examine qu'une phrase à la fois.

“I voted for Nader because he was most aligned with my values,” she said.

FIGURE 1.8 – Exemple de Coréférence

1.2.10 Les techniques utilisées en TLN

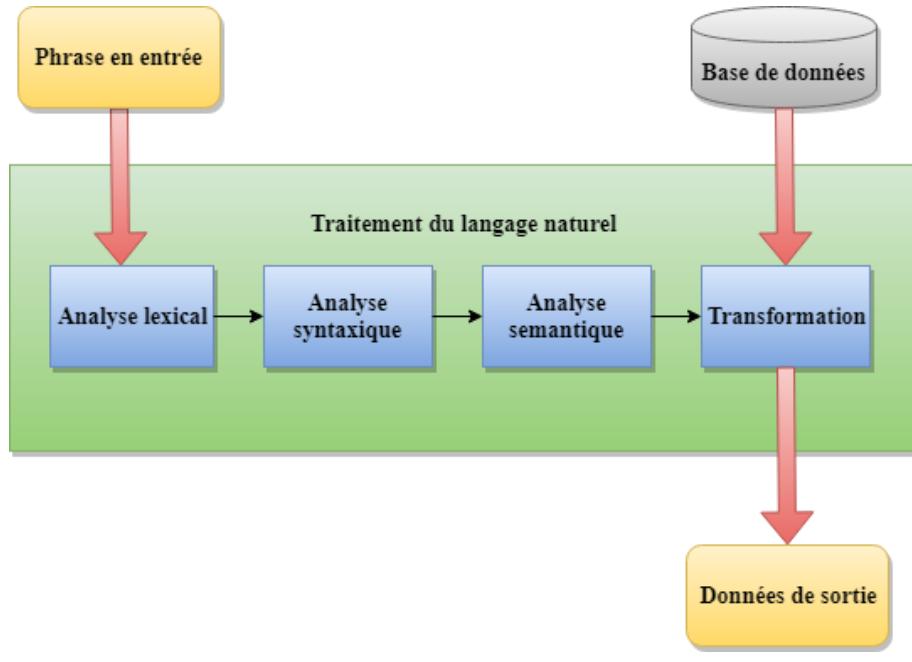


FIGURE 1.9 – Le fonctionnement du TLN (Référence : RESPENSABLE, 28 Octobre 2018)

L’analyse syntaxique (syntaxe) et l’analyse sémantique (sémantique) (Référence : DONGES, 28 Juin 2019) sont les principales techniques utilisées pour le TLN. Elles sont les deux techniques principales qui conduisent à la compréhension du langage naturel. La langue est un ensemble de phrases valides, mais en quoi une phrase est-elle valide ? En fait, la validité se divise en deux choses : la syntaxe et la sémantique. Le terme syntaxe fait référence à la structure grammaticale du texte alors que le terme sémantique fait référence au sens. Cependant, une phrase syntaxiquement correcte ne doit pas nécessairement être sémantiquement correcte. Exemple de phrase : « les vaches s’écoulent suprêmement » est grammaticalement valable (sujet - verbe - adverbe) mais n’a aucun sens.

1. Analyse syntaxique

L’analyse syntaxique est le processus d’analyse d’un langage naturel conforme aux règles d’une grammaire formelle. Les règles grammaticales s’appliquent aux catégories et aux groupes de mots, pas aux mots individuels. L’analyse syntaxique attribue fondamentalement une structure sémantique au texte.

2. Analyse sémantique

En tant qu’humains, la façon de comprendre de ce que quelqu’un a dit est un processus inconscient qui repose sur l’intuition et la connaissance du langage lui-même. Par conséquent, la façon de comprendre le langage est fortement basée sur le sens et le contexte. Puisque les ordinateurs ne peuvent pas compter sur ces techniques, ils ont besoin d’une approche différente. Le mot « sémantique » est un terme linguistique et signifie quelque chose en rapport avec le sens ou la logique.

L'analyse sémantique est le processus de compréhension du sens et de l'interprétation des mots, des signes et de la structure des phrases. Elle fait référence à la signification d'un texte. Cela permet aux ordinateurs de comprendre en partie le langage naturel comme le font les humains, impliquant sens et contexte. L'analyse sémantique est l'un des aspects difficiles du traitement automatique du langage qui n'a pas encore été complètement résolu. Cela implique l'application d'algorithmes informatiques pour comprendre le sens et l'interprétation des mots et la structure des phrases.

1.2.11 Construction d'un pipeline du TLN

Segmentation des phrases

La première étape consiste à diviser le texte en phrases séparées. Il sera beaucoup plus facile d'écrire un programme pour comprendre une phrase que de comprendre un paragraphe entier.

Coder un modèle de segmentation de phrases peut être aussi simple que de scinder des phrases à chaque vue d'un signe de ponctuation. Mais les pipelines du TLN modernes utilisent souvent des techniques plus complexes qui fonctionnent même lorsqu'un document n'est pas formaté correctement.

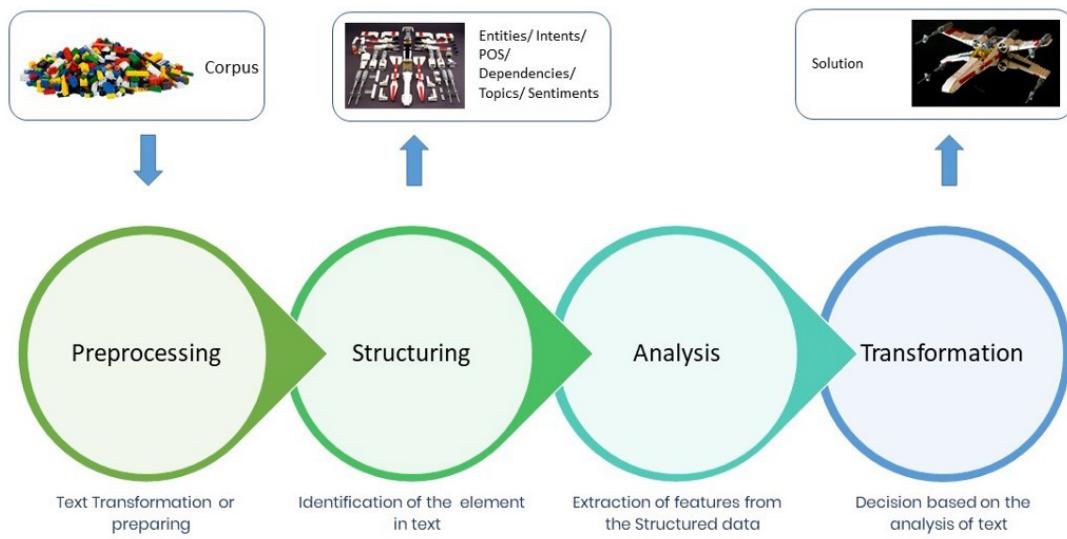


FIGURE 1.10 – La pipeline du TLN (Référence : SATHIYAKUGAN, 24 Juillet 2018)

Le pré-traitement et la structuration doivent être faite sur l'entrée d'un texte (x). Un grand ensemble de données en TLN est utilisé. Celui-ci s'appelle corpus, car il est composé d'un ensemble d'informations textuelles (le pluriel est un corpus). Ce corpus peut être considéré comme un ensemble de pièces de construction dans l'exemple du monde réel.

Prétraitement

Le traitement de données, est appelée en TLN normalisation du texte ou préparation des données.

Les tâches de TLN font très souvent appel au Machine Learning. Un des points clé dans l'efficacité d'un modèle de Machine Learning est la qualité des données utilisées. Or les données textuelles sont très complexes à traiter. La partie de prétraitement des données dans une problématique de TLN est donc longue et fastidieuse.

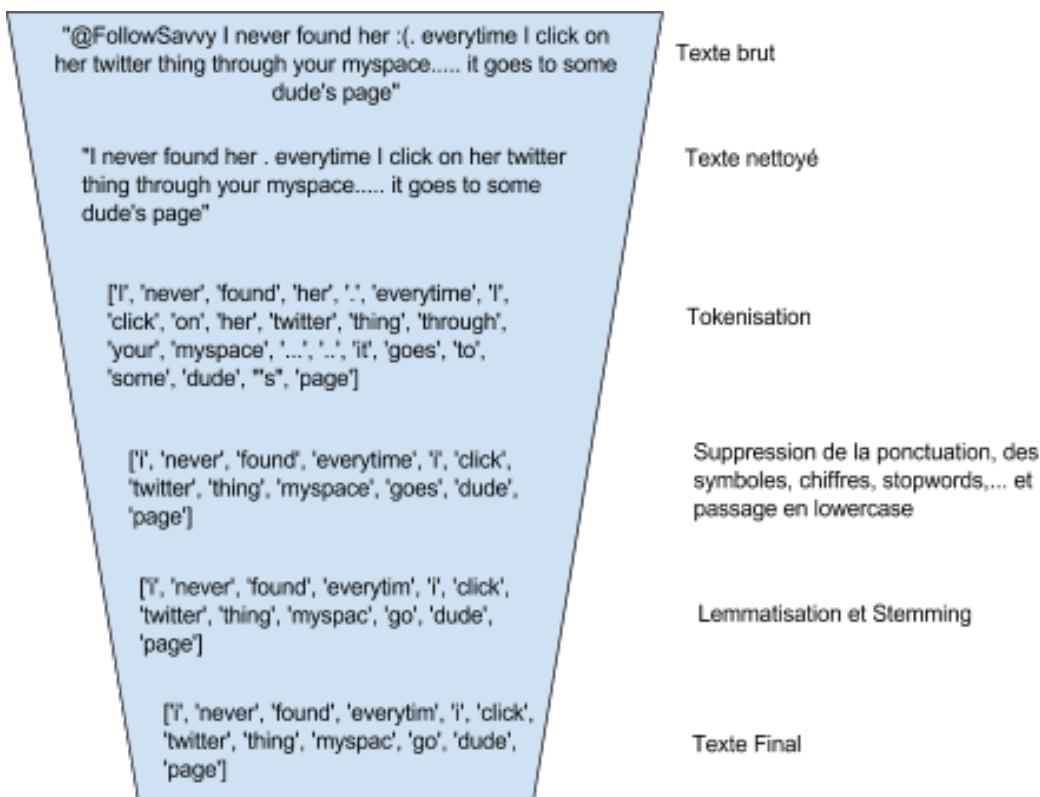


FIGURE 1.11 – Les étapes du pré-traitement (Référence : “Introduction au NLP (Partie II)”)

Elle commence en général par une phase de nettoyage qui varie selon la source des données. Par exemple, dans le cas de données issues de réseaux sociaux ou forums, il peut être nécessaire de supprimer les smileys, emojis, urls et autres spécificités (hashtags, mentions...) mais aussi de gérer les onomatopées et les contractions de mots.

Une fois nettoyées, les données doivent être normalisées : tokenisation, lemmatisation, stemming, suppression des chiffres, ponctuation, symboles et stopwords, passage en minuscule. Toutes ces transformations ne sont pas forcément nécessaires, cela dépend de l'application et il faut parfois tester différentes approches pour déterminer quelles transformations augmenteront l'efficacité du modèle.

1. Normalisation (Référence : JÉRÉMY, 17 Octobre 2018)

(a) Tokenisation

Le processus de tokenisation consiste à diviser chaque phrase en mots ou en jetons distincts. Les mots seront séparés chaque fois qu'il y a un espace entre eux. Les signes de ponctuation sont traités comme des jetons distincts, car la ponctuation a également une signification. Il s'agit donc de décomposer une phrase, et donc un document en tokens. La figure 1.12 présente la tokenisation pour la phrase d'exemple. Le passage en minuscule est aussi intégré.

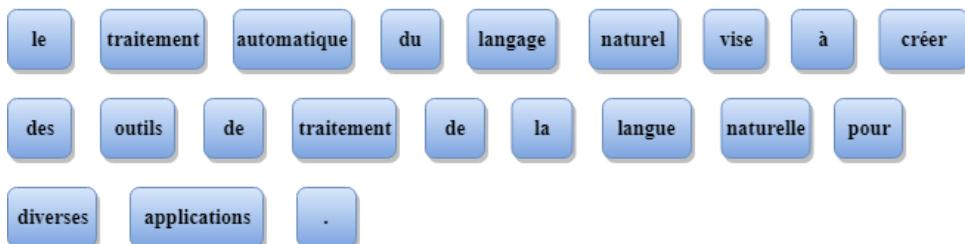


FIGURE 1.12 – Tokenisation

Chaque jetons (token) est examiné pour deviner quelle est sa partie du discours, s'il s'agit d'un nom, d'un verbe, d'un adjectif, etc. Connaître le rôle de chaque mot dans la phrase aidera à comprendre de quoi elle parle.

(b) Retrait des stopwords (mots vides)

Lors de l'utilisation des statistiques sur du texte, ces mots introduisent beaucoup de bruit car ils apparaissent beaucoup plus souvent que d'autres mots. Certains pipelines du TLN les marqueront comme des mots vides, c'est-à-dire des mots peuvent-être filtrer avant toute analyse statistique.

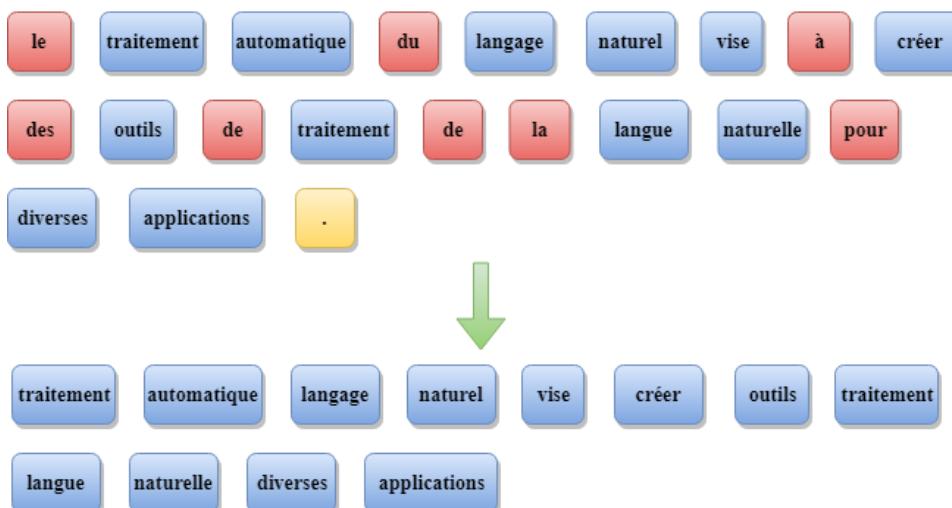


FIGURE 1.13 – Retrait des Stopwords

2. Groupement sémantique

Un mot peut être écrit au pluriel, au singulier ou avec différents accords et les verbes peuvent être conjugués à différents temps et personnes. Les différences grammaticales des mots doivent être réduites en trouvant des formes communes. Pour ce faire, il existe deux méthodes distinctes : la stammatisation et la lemmatisation.

(a) La stammatisation

La stammatisation (ou racinisation) réduit les mots à leur radical ou racine.

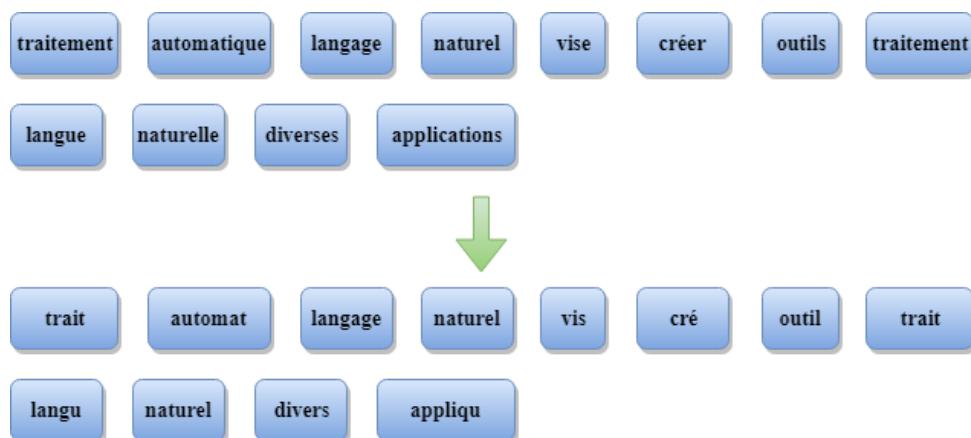


FIGURE 1.14 – Stemmatisation

(b) La Lemmatisation

Lors du travail avec du texte sur un ordinateur, il est utile de connaître la forme de base de chaque mot afin de savoir que les deux phrases parlent du même concept.

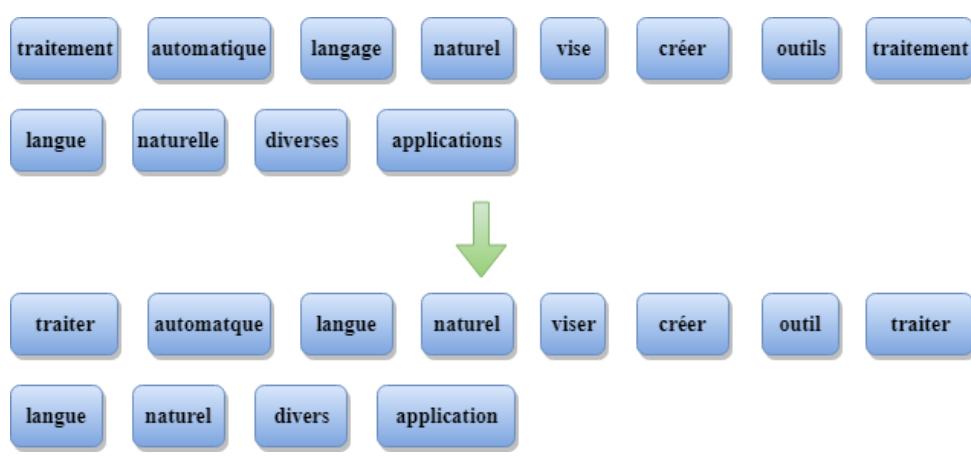


FIGURE 1.15 – Lemmatisation

Structuration

Après le prétraitement, Des groupes sont réalisés. La structuration peut dépendre de l'objectif de la tâche TLN. Nous choisissons quels sont les éléments que nous voulons détecter en fonction de l'objectif. Exemple les actions que l'utilisateur aurait pu consulter de manière positive ou négative

Analyse

Une fois les techniques de structuration appliquée, les informations structurées seront examinées. Il s'agit de la phase d'analyse au cours de laquelle différentes caractéristiques sont extraites afin d'exécuter une tâche. Comme par exemple extraire des informations sur les relations grammaticales des mots grâce à l'arbre d'analyse

Transformation

Le but est de traduire les informations obtenues en une source interprétable par le modèle (réseau de neurone) afin de prendre une décision, de l'observer, d'analyser, comprendre et relier les corrélations.

Les méthodes de Machine Learning nécessitant des données quantitatives et non textuelles, il faut passer par une phase de transformation du texte en chiffres. Les données de texte doivent être converties en représentations vectorielles.



FIGURE 1.16 – Différentes approches de représentation vectorielle
(Référence : SHAFFY, 8 Novembre 2017)

1.2.12 L'importance de la traduction automatique

La capacité de communiquer les uns avec les autres est un élément fondamental de l'être humain. Il existe près de 7 000 langues différentes dans le monde. Alors que le monde devient de plus en plus connecté, la traduction linguistique constitue un pont culturel et économique essentiel entre des peuples de différents pays et groupes ethniques. Parmi les cas d'utilisation les plus évidents :

- **Business** : Commerce international, investissement, contrats, finance.
- **Commerce** : Voyages, achat de biens et services étrangers, support client.
- **Médias** : Accès à l'information via la recherche, partage de l'information via les réseaux sociaux, localisation du contenu et publicité.
- **Éducation** : Partage d'idées, collaboration, traduction de documents de recherche.

- **Gouvernement** : Relations extérieures, négociation.

Pour répondre à ces besoins, les entreprises technologiques investissent massivement dans la traduction automatique. Cet investissement et les progrès récents dans l'apprentissage en profondeur ont permis d'améliorer considérablement la qualité de la traduction. Aujourd'hui, Google et Microsoft peuvent traduire plus de 100 langues différentes et approchent la précision au niveau humain pour nombre d'entre elles.

1.2.13 Le pipeline de construction d'un modèle de traduction

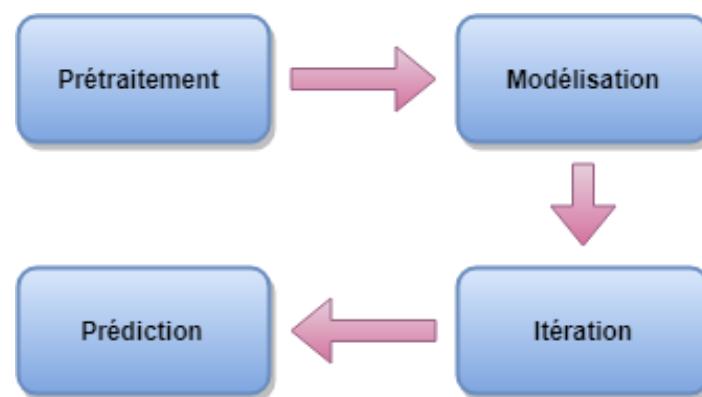


FIGURE 1.17 – Pipeline de construction d'un modèle de traduction

- **Prétraitement** : Chargement et examen des données, nettoyage, création de jetons, remplissage.
- **Modélisation** : Construire, former et tester le modèle.
- **Prédiction** : Générer des traductions spécifiques de langage source vers le langage cible et comparer les traductions de sortie aux traductions de vérité au sol.
- **Itération** Itérer sur le modèle, expérimenter avec différentes architectures.

1.3 La traduction automatique statistique (SMT)

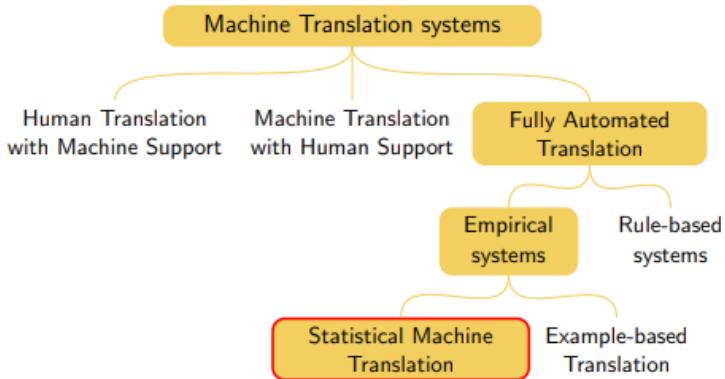


FIGURE 1.18 – Historique de la traduction automatique

La traduction automatique statistique (SMT, statistical machine translation) apprend à traduire en analysant les traductions humaines existantes (connues sous le nom de corpus de texte bilingue). Contrairement à l'approche de la traduction automatique basée sur les règles (RBMT) qui est généralement basée sur des mots. Dans la traduction basée sur les phrases, l'objectif est de réduire les restrictions de la traduction basée sur les mots en traduisant des séquences entières de mots, dont les longueurs peuvent être différentes. Les séquences de mots sont appelées phrases, mais ne sont généralement pas des phrases linguistiques, mais des phrases trouvées à l'aide de méthodes statistiques à partir de corpus de texte bilingues.

L'analyse des corpus textuels bilingues (langues source et cible) et des corpus monolingues (langue cible) génère des modèles statistiques qui transforment le texte d'une langue à une autre en utilisant des pondérations statistiques pour décider de la traduction la plus probable. (Référence : GEITGEY, 22 Août 2016)

1.3.1 Penser aux probabilités

La différence fondamentale avec les systèmes de traduction statistiques est qu'ils n'essayent pas de générer une traduction exacte. Au lieu de cela, ils génèrent des milliers de traductions possibles, puis ils les classent en fonction de leur exactitude. Ils estiment à quel point quelque chose est correcte en se basant sur sa similitude avec les données d'apprentissage.

1.3.2 Les limites de la traduction automatique statistique

Les systèmes de traduction automatique statistique fonctionnent bien, mais leur construction et leur maintenance sont compliquées. Chaque nouvelle paire de langues à traduire nécessite des experts pour peaufiner et ajuster un nouveau pipeline de traduction en plusieurs étapes. Pour la traduction du géorgien en Telegu directement. Le système décrit qu'il doit être traduit en anglais en tant qu'étape intermédiaire, car il n'y a pas suffisamment de traductions du géorgien à Telegu.

1.4 La traduction automatique neuronale

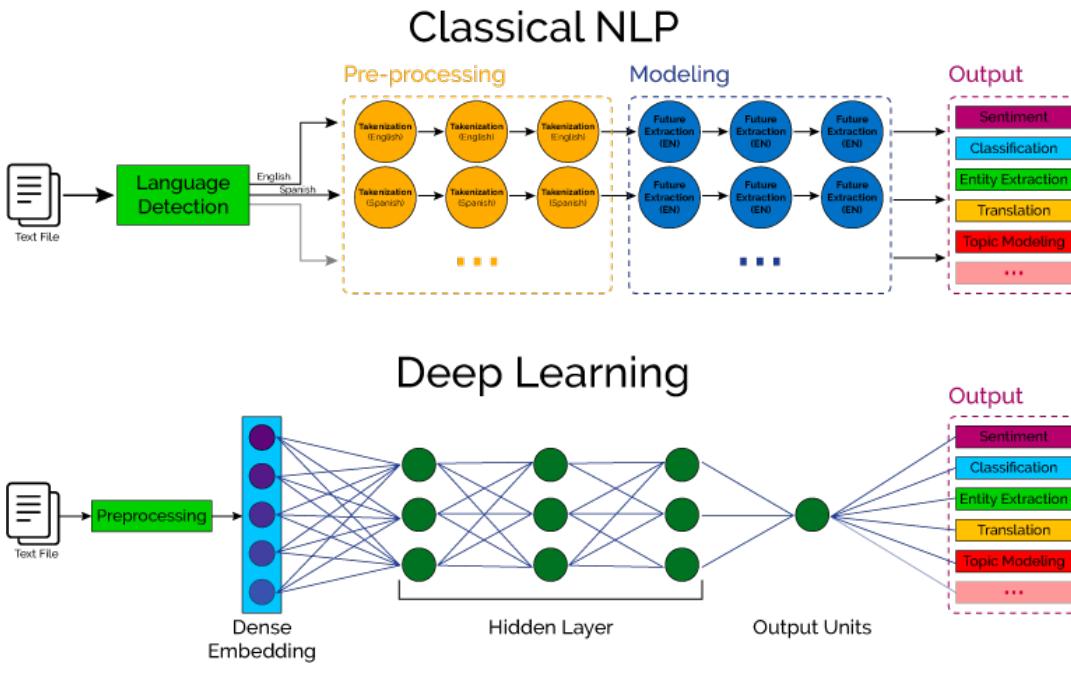


FIGURE 1.19 – Deep learning vs le TLN Classique (Référence : “Overview of Artificial Intelligence and Natural Language Processing”)

Les mécanismes qui permettent aux ordinateurs d'effectuer des traductions automatiques entre les langues humaines (telles que Google Translate) sont connus sous nom de la traduction automatique (MT), la plupart des systèmes de ce type actuels étant basés sur des réseaux de neurones, ces modèles se retrouvent donc sous la balise de Neural Machine Translation, ou NMT. Les résultats les plus récents sont obtenus à l'aide de la classe de réseaux de neurones qui est le Deep Learning. Alors que la traduction automatique basée sur les phrases (PBMT) divise une phrase d'entrée en mots et expressions à traduire de manière largement indépendante. Neural Machine Translation (NMT) considère la phrase d'entrée entière comme une unité de traduction. L'avantage de cette approche est qu'elle nécessite moins de choix de conception technique par rapport aux systèmes de traduction précédents basés sur la phrase.

La traduction automatique neuronale (également Neural MT ou NMT) est une approche relativement récente compte tenu de l'historique de la traduction automatique, mais il est permis de dire qu'elle est désormais très clairement à la pointe de la technologie. C'est une approche radicalement différente du défi de la traduction d'une langue qui utilise des réseaux de neurones profonds et une intelligence artificielle pour former des modèles pour différentes combinaisons de langues et types de contenu.

Comme toutes les approches de la traduction assistée par données, les moteurs Neural MT apprennent à traduire entre les langues sur la base d'un grand nombre d'exemples de traductions précédentes, appelées données d'apprentissage. Cependant, contrairement à ses prédecesseurs, la MT (Machine Translation) se basait sur

des règles et des statistiques. Neural MT est capable de penser d'avantage que le cerveau humain lors de la production de traductions. Plutôt que de traduire mot à mot ou de regarder des séquences de mots plus petites en parallèle, il est possible de placer simultanément des phrases entières, voir d'avantage, dans leur contexte. Les résultats obtenus sont des traductions plus fluides, précises et utilisables que jamais.

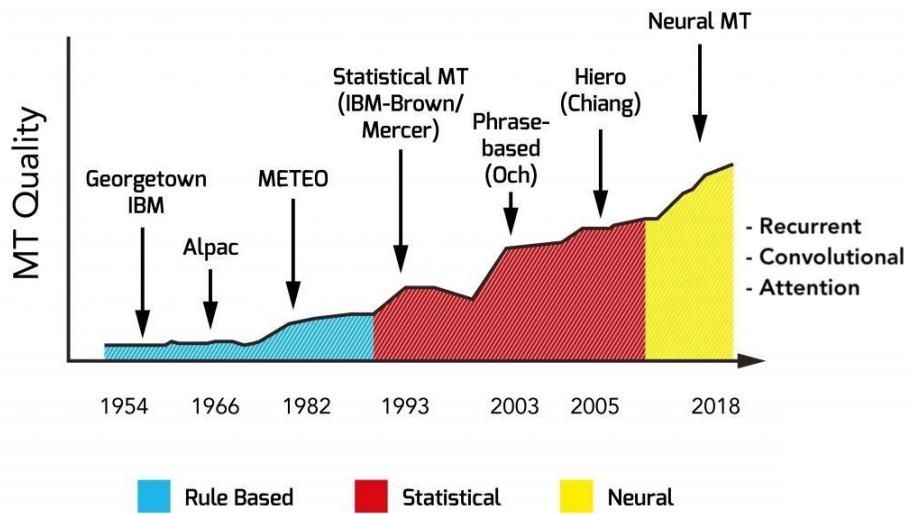


FIGURE 1.20 – Qualité de traduction avec les différentes méthodes
(Référence : “Neural Machine Translation”)

La traduction automatique est essentiellement une tâche dans laquelle une phrase est associée à une autre. Les phrases sont composées de mots, cela revient donc à mapper une séquence sur une autre séquence. Le mappage de séquences en séquence est une structure de tâche omniprésente dans le TLN. Les gens ont développé de nombreuses méthodes pour effectuer un tel mappage : ces méthodes sont appelées méthodes séquence à séquence.

1.4.1 Le modèle séquence à séquence

Séquence-à-séquence (ou Seq2Seq) est un réseau de neurones qui transforme une séquence d'éléments donnée, telle que la séquence de mots d'une phrase, en une autre séquence. Les modèles Seq2Seq sont particulièrement efficaces en traduction, où la séquence de mots d'une langue est transformée en une séquence de mots différents dans une autre langue.

Généralement, les tâches séquence à séquence sont effectuées à l'aide d'un modèle codeur-décodeur. Le codeur prend la séquence d'entrée et la mappe dans un vecteur à n dimensions. Ce vecteur abstrait est introduit dans le décodeur qui le transforme en une séquence de sortie.

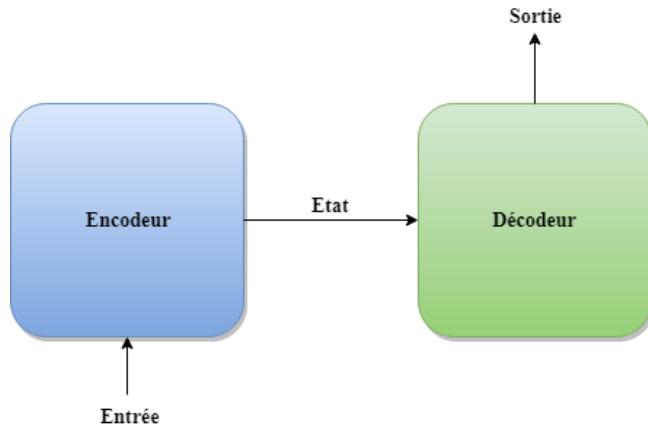


FIGURE 1.21 – Encodeur et décodeur

L'idée de base est que le codeur prend la séquence de mots d'entrée (par exemple, « J'aime les chats plus que les chiens »), la convertit en une représentation intermédiaire, puis transmet cette représentation au décodeur qui produit la séquence de sortie (par exemple : i love cats more than dog).

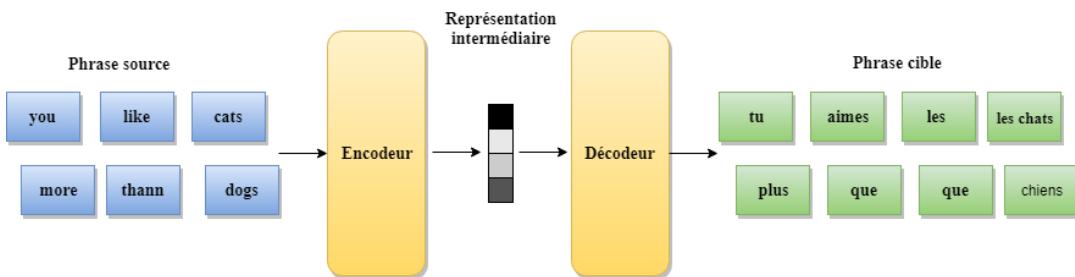


FIGURE 1.22 – Séquence à séquence (Référence : KEITAKURITA, 29 décembre 2017)

L'un des principaux défis de la transduction de séquence consiste à apprendre à représenter à la fois les séquences d'entrée et de sortie de manière invariante aux distorsions séquentielles telles que le rétrécissement, l'étirement et la traduction. Ceci est une limitation sévère car trouver l'alignement est l'aspect le plus difficile de nombreux problèmes de transduction de séquence. En effet, même déterminer la longueur de la séquence de sortie est souvent difficile.

1.4.2 L'encodeur

L'encodeur prend simplement les données d'entrée et s'entraîne dessus, puis passe le dernier état de sa couche en tant qu'état initial à la première couche de la partie décodeur.

1.4.3 Décodeur

Le décodeur prend le dernier état de la dernière couche du codeur et l'utilise comme état initial vers sa première couche. L'entrée du décodeur correspond aux séquences que nous voulons obtenir.

1.4.4 Une pile d'encodeur et de décodeur

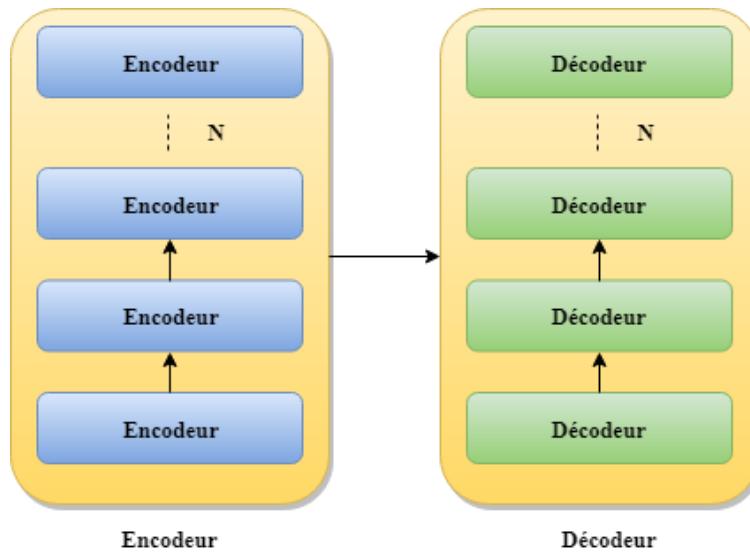


FIGURE 1.23 – Des piles d'encodeurs et décodeurs

Les composants encodeurs et décodeurs en eux même contiennent une pile d'encodeurs et de décodeurs, chaque composant communique avec celui qui lui est inférieure.

1.4.5 L'Attention visuelle

Les différentes méthodes du Deep Learning utilisées, sont inspirées sur fonctionnement du cerveau de l'humain.

L'essence des mécanismes d'attention est qu'ils sont une imitation du mécanisme de la vue humaine. Lorsque le mécanisme de vision humaine détecte un élément, il ne numérise généralement pas la scène entière de bout en bout. Au lieu de cela, il se concentrera toujours sur une portion spécifique en fonction des besoins de la personne. Lorsqu'une personne remarque que l'objet auquel elle veut faire attention apparaît généralement dans une partie particulière d'une scène, elle apprend que, dans le futur, l'objet examinera cette partie et aura tendance à concentrer son attention sur cette zone.

1.4.6 Le mécanisme de l'attention

Les travaux précurseurs de Bahdanau et al (2014) ont introduit des mécanismes d'attention dans les modèles seq2seq. Intuitivement, ce mécanisme permet au décodeur de « regarder en arrière » l'ensemble de la phrase et d'extraire sélectivement les informations dont il a besoin pendant le décodage. Concrètement, les mécanismes d'attention permettent des connexions entre les états cachés(sortie) de chaque codeur et décodeur, de sorte que le mot cible soit prédit sur la base d'une combinaison de vecteurs de contexte, plutôt que simplement de l'état caché précédent du décodeur, ce mécanisme donne au décodeur l'accès à tous les états cachés du codeur. Cela a du sens, seule une petite partie de la phrase source est pertinente pour la traduction d'un mot cible. Le décodeur doit faire une seule prédiction pour le mot suivant, une séquence complète ne peut être envoyée, mais une sorte de vecteur de résumé doit être transmit à la place. Ce qui importe, c'est qu'il demande au décodeur de choisir les états cachés à utiliser et ceux à ignorer en pondérant les états cachés, le décodeur reçoit ensuite une somme pondérée d'états cachés à utiliser pour prédire le mot suivant.

1.4.7 L'impact du mécanisme de l'attention

L'implication des systèmes basés sur l'attention dans le monde de la traduction automatique est énorme. La machine peut maintenant collecter des informations nuancées et traduire des phrases plus naturellement. Les systèmes basés sur l'attention ont surpassé les traducteurs existants, avec moins d'erreurs de tension, d'intention et de référence. C'est une méthode omniprésente dans les modèles d'apprentissage en profondeur modernes. L'attention est un concept qui a permis d'améliorer les performances des applications de traduction automatique neuronale.

1.5 Exemple de but recherché

1.5.1 Google neural machine translation GNMT



FIGURE 1.24 – Logo de Google translate

En novembre, Google Translate a commencé à utiliser la traduction automatique des neurones (NMT) plutôt que ses méthodes statistiques antérieures (SMT) qui était utilisé depuis octobre 2007 avec sa technologie SMT propriétaire interne.

Les chercheurs de Google s’attendaient à ce que la structure continue à apprendre et à s’améliorer. Cela fait partie de l’utilisation de l’apprentissage en profondeur. Cependant, ils ont été surpris de constater que le GNMT pourrait apprendre à effectuer des traductions pour une paire de langues qui n’a jamais été expérimentée. Les chercheurs ont testé le GNMT pour voir si le système avait été formé à la traduction japonais/anglais et à la traduction coréen/anglais s’il était capable d’apprendre à effectuer des traductions japonais/coréen sans avoir à être spécifiquement formé à cette tâche. Le GNMT était en fait capable d’utiliser son expérience des traductions japonais/anglais et coréen/anglais pour développer un système permettant d’effectuer des traductions japonais/coréen. Ceci est une amélioration par rapport à l’ancien système Google Translate en ce sens qu’il a été capable de gérer la “traduction instantanée”.

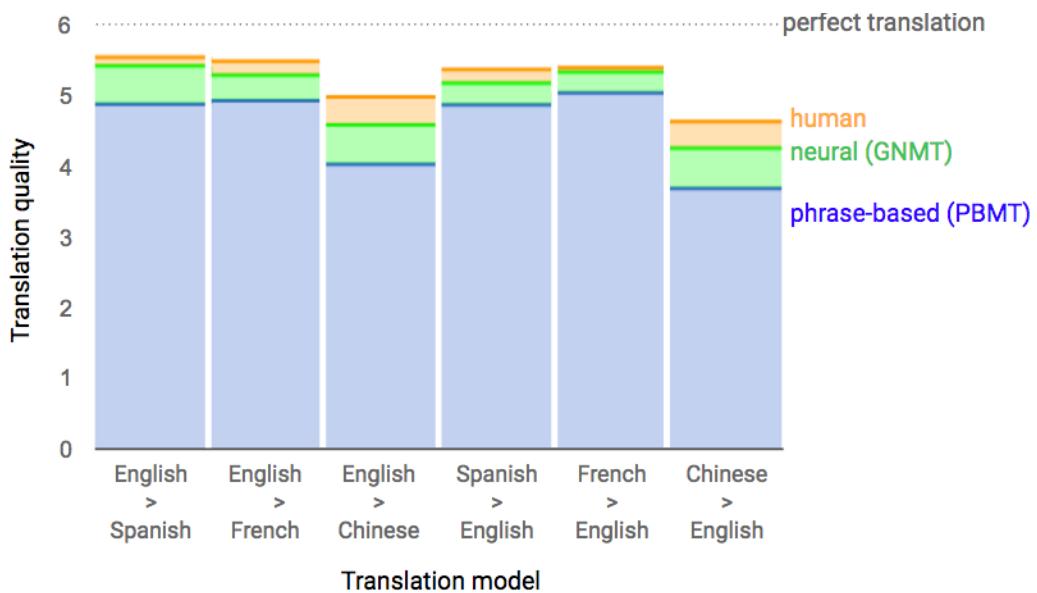


FIGURE 1.25 – GNMT comparaison (Référence : AI, Mardi, 27 September 2016)

La figure 1.25 représente des données provenant d'évaluations côte à côte, dans lesquelles des évaluateurs humains comparent la qualité des traductions pour une phrase source donnée. Les scores vont de 0 à 6.

La traduction automatique n'est en aucun cas résolue. GNMT peut toujours commettre des erreurs importantes qu'un traducteur humain ne ferait jamais, comme laisser tomber des mots et traduire mal les noms propres ou des termes rares, et traduire des phrases isolément plutôt que de tenir compte du contexte du paragraphe ou de la page.

Les réseaux de neurones récurrents

Le document phare de Sutskever et al (2014) a introduit l'apprentissage séquence à séquence (seq2seq), en utilisant des réseaux de neurones récurrents pour les codeurs et les décodeurs.

Les réseaux neuronaux récurrents (RNN) sont un type puissant et robuste de réseaux neuronaux et appartiennent aux algorithmes les plus prometteurs du moment car ils sont les seuls à avoir une mémoire interne. Cependant, les RNN exigent traditionnellement un alignement prédéfini entre les séquences d'entrée et de sortie pour effectuer la transduction. Il existe plusieurs types d'architecture RNN : Vanilla (RNN de base), LSTM et, GRU. C'est l'architecture LSTM qui est la plus utilisée.

Les LSTM permettent aux RNN de se souvenir de leurs intrants sur une longue période de temps. C'est parce que les LSTM contiennent leurs informations dans une mémoire, ce qui ressemble beaucoup à la mémoire d'un ordinateur parce que le LSTM peut lire, écrire et supprimer des informations de sa mémoire. Elle résout le problème des gradients de disparition qui affectent les modèles RNN profonds.

Les RNN constituaient l'architecture la plus utilisée et la plus aboutie à la fois pour le codeur et le décodeur. Les RNN semblaient être nés pour cette tâche, leur nature récurrente correspondait parfaitement à la nature séquentielle du langage. Les humains lisent les phrases de gauche à droite (ou de droite à gauche). Il était donc logique d'utiliser les RNN pour coder et décoder le langage.

Dans les modèles LSTM seq2seq, le décodeur génère une traduction basée uniquement sur le dernier état caché du codeur. La qualité de la traduction est pire pour les phrases plus longues.

Le document note également quelques observations importantes :

- Les modèles peuvent être améliorés en introduisant les phrases source à l'envers. Car cela introduit de nombreuses dépendances à court terme dans les données qui facilitent beaucoup le problème d'optimisation.
- Les modèles peuvent s'améliorer en alimentant une séquence d'entrée deux fois.

L'attention et GNMT

La visualisation suivante montre la progression de GNMT lors de la traduction d'une phrase chinoise en anglais. Le réseau encode les mots chinois sous forme de liste de vecteurs, chaque vecteur représentant la signification de tous les mots lus jusqu'à présent « Encoder ». Une fois que toute la phrase est lue, le décodeur commence et génère la phrase anglaise, mot par mot « Decoder ».

Pour générer le mot traduit à chaque étape, le décodeur fait attention à une distribution pondérée sur les vecteurs chinois codés les plus pertinents pour générer le mot anglais avec l'attention. La transparence du lien bleu représente à quel point le décodeur accorde de l'attention à un mot codé). (Référence : AI, Mardi, 27 September 2016)

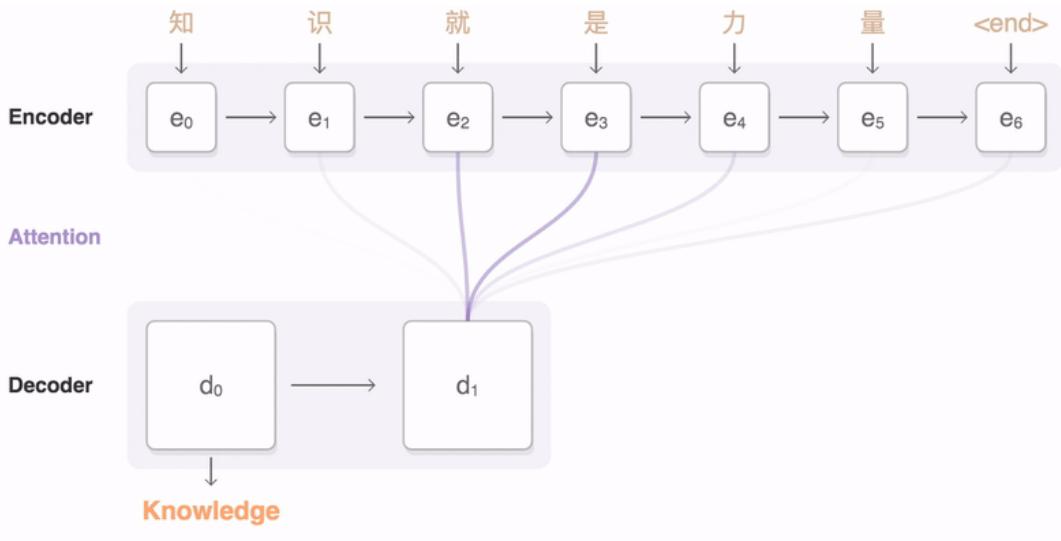


FIGURE 1.26 – Encodage et décodage avec GMNT

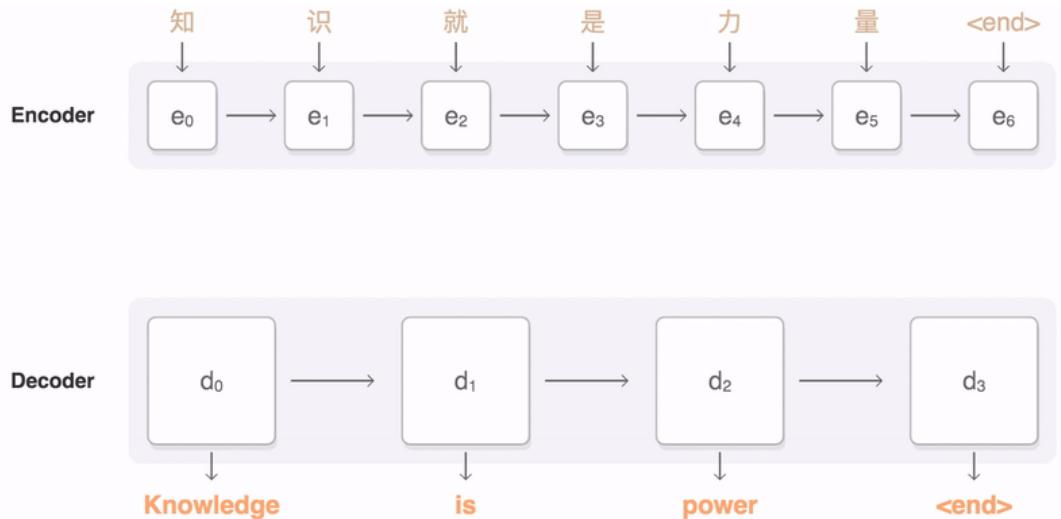


FIGURE 1.27 – Résultat d'encodage et décodage avec GMNT

En utilisant une comparaison côté à côté évaluée par l'homme comme métrique, le système GNMT produit des traductions qui sont considérablement améliorées par rapport au système de production précédent basé sur la phrase. GNMT réduit les erreurs de traduction de plus de 55% à 85% sur plusieurs paires de langues majeures mesurées sur des phrases échantillonées de Wikipedia et de sites Web d'actualités avec l'aide d'évaluateurs humains bilingues.

Il faut remonter à 1948 et à l'épisode dramatique de la « Violencia », guerre civile qui a fait deux cent mille à trois cent mille morts, opposant libéraux et conservateurs, après l'assassinat du leader libéral Jorge Eliécer Gaitan.	Not since 1948 and the dramatic episode of the "Violencia" civil war that two hundred thousand to three hundred thousand dead, between liberals and conservatives, after the assassination of Liberal leader Jorge Eliécer Gaitan.	One must go back to 1948 and to the dramatic episode of the "Violencia", a civil war that left two hundred thousand to three hundred thousand dead, opposing liberals and conservatives, after the assassination of liberal leader Jorge Eliécer Gaitan.	One must go back to 1948, to the dramatic episode of "Violencia", a civil war that left two to three hundred thousand dead, when liberals were fighting conservatives after the assassination of liberal leader Jorge Eliécer Gaitan.
--	--	--	---

FIGURE 1.28 – Exemple de traduction produite à partir d'un site de nouveautés (Référence : AI, Mardi, 27 September 2016)

Le bleu représente une traduction PBMT, le vert GNMT, et l'orange une traduction humaine

Puissance des LSTM

- Cette approche est principalement limitée par la quantité de données d'entraînement disposé et par la quantité de puissance informatique utilisée. Ils fonctionnent déjà aussi bien que les systèmes de traduction automatique statistiques qui ont mis 20 ans à se développer.
- Cela ne dépend pas de la connaissance des règles relatives au langage humain. L'algorithme détermine ses règles lui-même. Cela signifie qu'il n'y est plus besoin d'experts pour ajuster chaque étape le pipeline de traduction.

Avantage

Les RNN sont populaires et réussissent pour des représentations de longueur variable. Ils sont considérés comme le noyau de seq2seq (avec l'attention). Les modèles de déclenchement tels que LSTM ou GRU sont destinés à la propagation d'erreur à longue portée.

1.5.2 Facebook

L'équipe Facebook de recherche sur l'intelligence artificielle (FAIR) a publié des résultats de la recherche en utilisant une nouvelle approche de réseau de neurones convolutionnels (CNN) pour la traduction du langage, qui permet d'obtenir une précision de pointe à une vitesse neuf fois supérieure à celle des systèmes neuronaux récurrents.

Pourquoi des réseaux de neurones convolutifs ?

Développé à l'origine par Yann LeCun il y a plusieurs décennies, les CNN ont connu un grand succès dans plusieurs domaines de l'apprentissage automatique, tels que le traitement d'images. Cependant, les réseaux de neurones récurrents (RNN) sont la technologie actuelle pour les applications de texte et ont été le premier choix en matière de traduction de langue en raison de leur grande précision.

La conception des RNN présente une limite inhérente, qui peut être comprise en examinant la manière dont ils traitent les informations. Les ordinateurs traduisent le texte en lisant une phrase dans une langue et en prédisant une séquence de mots dans une autre langue ayant le même sens. Les RNN fonctionnent dans un ordre strict de gauche à droite ou de droite à gauche, un mot à la fois, le calcul ne peut pas être entièrement parallélisé, car chaque mot doit attendre que le réseau ait terminé avec le mot précédent. En comparaison, les CNN peuvent calculer tous les éléments simultanément, donc plus efficaces sur le plan informatique.

Meilleure traduction avec multi-hop attention et gating

L'attention multi-hop est un élément distinctif de l'architecture. Un mécanisme d'attention ressemble à la façon dont une personne décompose une phrase lorsqu'elle la traduit, au lieu de regarder la phrase une seule fois, puis d'écrire la traduction complète sans regarder en arrière, le réseau prend des « aperçus » répétés de la phrase pour choisir les mots qu'il traduira ensuite, un peu comme un être humain qui revient à des mots-clés spécifiques lors de la rédaction d'une traduction. Multi-hop attention est une version améliorée de ce mécanisme, qui permet au réseau de créer plusieurs aperçus afin de produire de meilleures traductions. Ces aperçus dépendent également les uns des autres. Par exemple, le premier aperçu pourrait se concentrer sur un verbe et le deuxième aperçu sur le verbe auxiliaire associé.

Explication

Dans la figure 1.29, le système lit une phrase française (encodage) puis génère une traduction anglaise (décodage). L'encodeur est exécuté pour créer un vecteur pour chaque mot français en utilisant un CNN, et le calcul est effectué simultanément. Ensuite, le décodeur CNN produit des mots anglais, un à la fois. À chaque étape, l'attention perçoit la phrase française pour décider quels mots sont les plus pertinents pour prédire le prochain mot anglais de la traduction. Il y a deux soi-disant couches dans le décodeur. La force des lignes vertes indique le degré de concentration du réseau sur chaque mot français. Lors de la formation du réseau, la traduction est toujours disponible et le calcul des mots anglais peut également être effectué simultanément. (Référence : GEHRING, 9 Mai 2017)

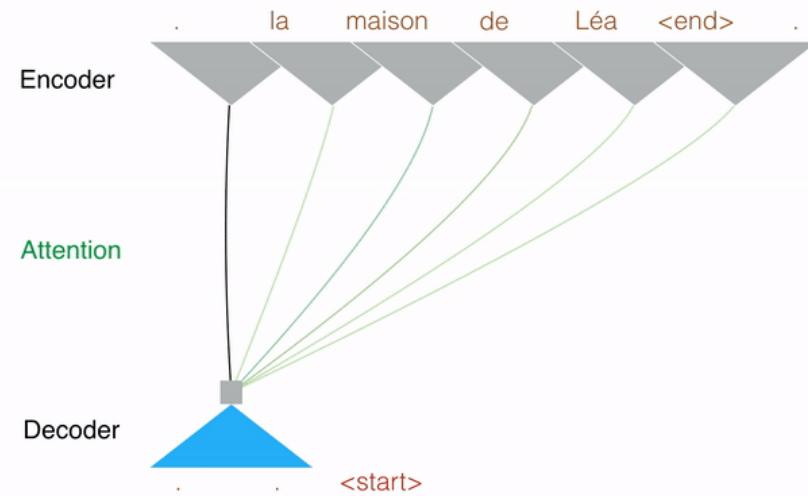


FIGURE 1.29 – Traduction d'une phrase en anglais avec les CNN
partie 1

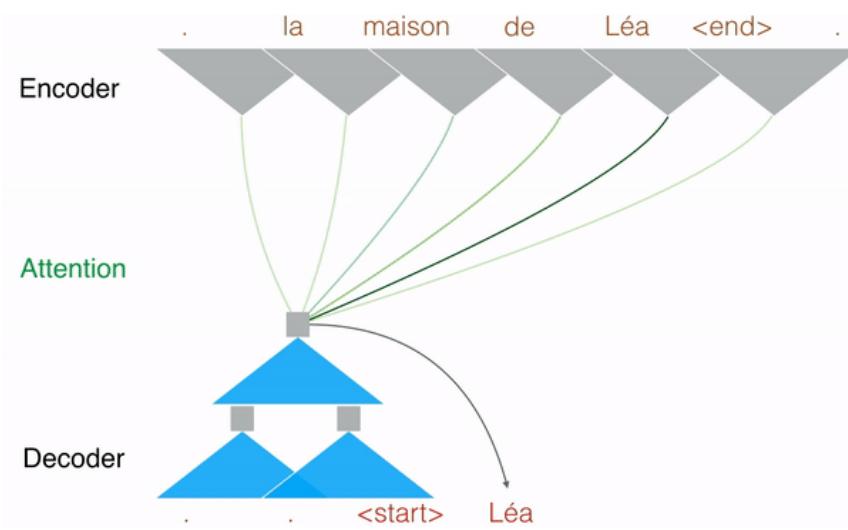


FIGURE 1.30 – Traduction d'une phrase en anglais avec les CNN
partie 2

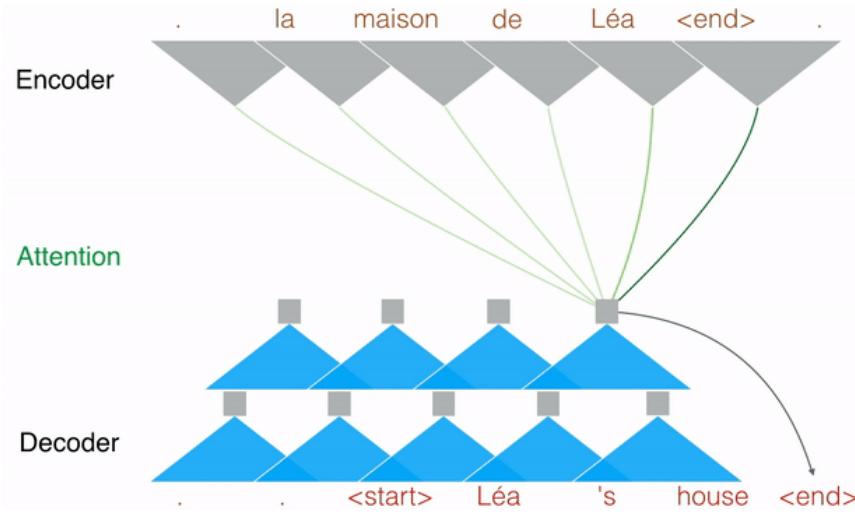


FIGURE 1.31 – Traduction d'une phrase en anglais avec les CNN partie finale

L'essentiel est que les éléments d'entrée proches interagissent au niveau des couches inférieures, tandis que les interactions distantes interagissent au niveau des couches supérieures. L'empilement des couches de convolution est utilisé pour évaluer les dépendances à longue portée entre les mots. Tout cela est possible malgré les noyaux de filtres de convolution à largeur fixe. Les auteurs ont également inclus deux astuces

- **Intégrations de position** : Ils sont ajoutés aux incorporations d'entrée pour capturer un sens de l'ordre dans une séquence.
- **Attention multi-étape** : L'attention est calculée avec l'état actuel du décodeur et l'intégration du jeton de mot cible précédent

Avantage

Elle peut paralléliser l'ensemble du processus d'apprentissage et peut être très rapide à entraîner contrairement au RNN où l'étape actuelle dépend du vecteur de l'étape précédente et nécessite un apprentissage séquentiel. Et adapte l'intuition selon laquelle la plupart des dépendances sont locales.

1.6 Les Transformers

Les architectures RNN sont donc toujours à la pointe de la technologie et méritent d'être approfondies. Jusqu'en 2018, cette architecture demeurait la principale architecture de la TLN. Certains commentateurs pensent qu'il est temps d'abandonner complètement les RNN. Aussi, de toute façon, il est peu probable qu'ils forment la base de nombreuses nouvelles recherches en 2019. Au lieu de cela, la principale tendance architecturale en TLN pour l'apprentissage en profondeur en 2019 sera le transformateur.

Le document « L'attention est tout ce dont vous avez besoin » publié par une équipe de chercheurs de Google, présente une nouvelle architecture appelée Transformer. Comme le titre l'indique, il utilise le mécanisme d'attention. Comme LSTM, Transformer est une architecture permettant de transformer une séquence en une autre, mais elle diffère des modèles séquence à séquence existants car elle n'implique aucun réseau récurrent (GRU, LSTM, etc.).

Les réseaux récurrents étaient, jusqu'à présent, l'un des meilleurs moyens de capturer les dépendances opportunes dans les séquences. Cependant, l'équipe présentant le document a prouvé qu'une architecture avec seulement des mécanismes d'attention sans RNN peut améliorer les résultats dans les tâches de traduction et d'autres tâches.

1.6.1 Une architecture dominante

L'architecture de transformateur, qui a été publiée pour la première fois fin 2017, résout le problème de séquence en créant un moyen d'autoriser les entrées en parallèle. Chaque mot peut avoir une incorporation séparée et être traité simultanément, ce qui améliore considérablement les temps de formation, ce qui facilite la formation sur des jeux de données beaucoup plus volumineux.

1.6.2 Perte d'informations

Actuellement, en TLN, la performance SOTA (state of the art) obtenue par les modèles seq2seq est axée sur l'idée de coder la phrase en entrée dans une représentation vectorielle à taille fixe. La taille du vecteur est fixe, quelle que soit la longueur de la phrase en entrée. De manière évidente, cela doit perdre certaines informations. Pour faire face à ce problème, Transformer utilise une approche alternative basée sur l'attention.

1.7 Conclusion

Dans ce chapitre, nous avons pu comprendre l'importance et l'impact du TLN dans la vie réelle, ainsi que les différentes difficultés qui y subsistent. Concernant la machine translation neural, plusieurs méthodes ont été évoquées en citant les avantages de chacune d'elles, ainsi que l'amélioration apportée par rapport aux anciennes approches.

Nous avons vu que les RNN et les CNN ont été des architectures puissantes, et qui ont fortement augmenté l'efficacité et l'ampleur de la traduction automatique en se basant sur le modèle séquence à séquence. Jusqu'à ce qu'une nouvelle architecture voit le jour, qui se nomme Les Transformers. C'est une architecture créée par Google qui se base sur le mécanisme de l'attention. Celle-ci a changé la donne dans le monde du TLN, en offrant des résultats extraordinaires avec une rapidité et des performances énormes. Cette architecture fera l'objet du chapitre suivant.

2

Chapitre de conception

2.1 Introduction

Les chercheurs de Google ont publié un article présentant une nouvelle architecture, le Transformer, basée uniquement sur des mécanismes d'attention, supprimant les récurrences et les convolutions. Le Transformateur du papier de recherche « L'attention est tout ce dont vous avez besoin » a beaucoup préoccupé l'esprit des gens, et celui-ci surpasserait tout autre modèle de NMT vu avant. Outre des améliorations majeures de la qualité de la traduction. Il fournit une nouvelle architecture pour de nombreuses autres tâches de la TLN. Le Transformer a pu atteindre un nouvel état de la technique en matière de traduction.

Dans ce chapitre, une comparaison sera faite entre les architectures RNN et CNN, puis les avantages qu'apporte le Transformer par rapport à ceux-ci. Dans la section qui suit, On analysera le document publié par google « L'attention est tout ce dont vous avez besoin », Après, nous entrerons en détail sur le fonctionnement de l'architecture Transformer. Puis on parlera de la fonction de perte ainsi que de la fonction d'optimisation, et on s'intéressera au score BLEU, qui est une métrique pour mesurer la qualité de traduction. Puis on parlera des techniques telles que les représentations vectorielles, et la normalisation. Et enfin on finira avec une conclusion.

2.2 Comparaison entre les architectures existantes

2.2.1 L'association de la linguistique informatique

L'Association pour la linguistique informatique (ACL) est la société scientifique et professionnelle internationale pour les personnes qui travaillent sur des problèmes impliquant le langage naturel et l'informatique. Une réunion annuelle a lieu chaque été dans des lieux où se déroulent d'importantes recherches en linguistique informatique. En 2014, ils organisent un workshop (WMT). Parmi les tâches proposées on y trouve une tâche de traduction Anglais-français. Ils fournissent un corpus sous forme de données de formation, il contient comme par exemple le corpus du parlement européen.

Le corpus du WMT2014 est utilisée comme corpus de formation et d'évaluation pour les différents modèles de traduction existants. (Référence : RICARDOKLEINK-LEIN, 26-27 juin 2014)

2.2.2 Les architectures actuelles

L'architecture du Transformer est destinée au problème de la transduction de séquence, signifiant toute tâche où les séquences d'entrée sont transformées en séquences de sortie. L'objectif consiste essentiellement à concevoir un cadre unique permettant de gérer autant de séquences que possible.

Les réseaux de neurones récurrents, en particulier la mémoire à court terme et les réseaux de neurones récurrents, ont été fermement établis comme approches de pointe en matière de modélisation de séquences et de problèmes de transduction tels que la modélisation de langage et la traduction automatique. Depuis, de nombreux efforts ont été déployés pour repousser les limites des modèles de langage récurrents et des architectures décodeur-décodeur.

Actuellement, les RNN et CNN complexes basés sur un schéma codeur-décodeur dominent les modèles de transduction. Les modèles récurrents dus à la nature séquentielle (les calculs centrés sur la position du symbole en entrée et en sortie) ne permettent pas la parallélisation au cours de la formation, ce qui pose un problème d'apprentissage des dépendances à long terme de mémoire. Plus la mémoire est grande, mieux c'est, mais elle finit par limiter le traitement par lots pour les exemples d'apprentissage de longues séquences et c'est pourquoi la parallélisation ne peut pas aider. La réduction de cette contrainte fondamentale du calcul séquentiel a été la cible de nombreuses recherches telles que Wavenet, Bytenet ou ConvS2S.

La nouvelle approche de Transformer consiste toutefois à éliminer complètement la récurrence et à la remplacer par une attention particulière afin de gérer les dépendances entre l'entrée et la sortie. Le Transformer déplace entièrement le point sensible des idées actuelles vers l'attention. Il élimine non seulement la récurrence, mais aussi la convolution en faveur de l'application de l'attention personnelle (alias intra-attention). De plus, Transformer laisse plus de place à la parallélisation.

Le Transformer est le premier à compter entièrement sur l'attention personnelle pour calculer les représentations des entrées et des sorties.

2.2.3 Les Transformers

Le Transformer est plus performant que les modèles récurrents et convolutionnels en termes de références académiques anglais-français. Outre que la qualité de traduction supérieure, celui-ci nécessite moins de calcul et est plus adapté aux matériels d'apprentissage automatique modernes, car il accélère la formation.

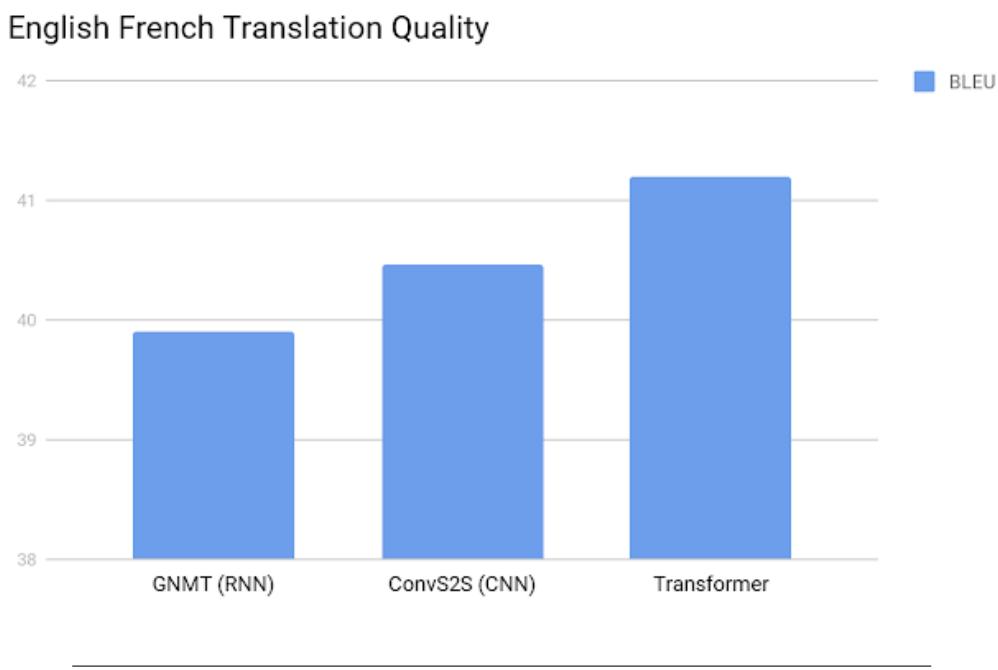


FIGURE 2.1 – Scores BLEU de modèles individuels sur la traduction standard WMT 2014 anglais-français (Référence : AI, Jeudi, 31 août 2017)

Les Transformers surpassent le modèle de traduction de Google Neural Machine dans des tâches spécifiques. Le principal avantage, cependant, provient de la manière dont le Transformer se prête à la parallélisation.

2.2.4 Exactitude et efficacité dans la compréhension du langage

Les réseaux de neurones traitent généralement le langage en générant des représentations d'espace vectoriel de longueur fixe ou variable. Après avoir commencé avec des représentations de mots individuels ou même de morceaux de mots, ils agrègent les informations des mots environnants afin de déterminer la signification d'un bit de langage donné dans son contexte. Par exemple, pour déterminer le sens le plus probable et la représentation appropriée du mot « banque » dans la phrase « Je suis arrivé à la banque après avoir traversé la ... », les humains peuvent rapidement déterminer que le sens du mot « banque » dépend de « route » ou « rivière ». Toutefois, les machines utilisant des RNN traitent la phrase de manière séquentielle, en lisant chaque mot avant d'établir une relation entre « banque » et « route » ou « rivière ».

Cela devient un goulot d'étranglement pour l'ordinateur fonctionnant sur des dispositifs informatiques rapides modernes tels que les TPU et les GPU qui excellent en

traitement parallèle et non séquentiel, car la nature séquentielle des RNN rend également plus difficile de tirer pleinement parti de ces dispositifs. Bien que les réseaux de neurones convolutifs (CNN) soient moins séquentiels que les RNN, mais dans les architectures CNN telles que ByteNet ou ConvS2S, le nombre d'étapes requises permettant d'établir des relations significatives afin de combiner des informations à partir de parties distantes de l'entrée augmente toujours avec la distance. Ainsi, les deux systèmes luttent avec des phrases comme celles présentées ci-dessus.

Les RNN sont devenus ces dernières années l'architecture réseau typique de la traduction, en traitant le langage de manière séquentielle de gauche à droite ou de droite à gauche. La lecture d'un mot à la fois oblige les RNN à effectuer plusieurs étapes pour prendre des décisions qui dépendent de mots éloignés les uns des autres. En reprenant l'exemple ci-dessus, un RNN n'a pu déterminer que « banque » est susceptible de faire référence à la rive d'une rivière après avoir lu chaque mot entre « banque » et « rivière », étape par étape. Des recherches antérieures ont montré qu'en gros, plus de telles étapes nécessitent de prendre des décisions, plus il est difficile pour un réseau récurrent d'apprendre à prendre ces décisions (Référence : AI, Jeudi, 31 août 2017).

2.2.5 Solution du Transformer

Le Transformer présente une nouvelle approche. Il propose d'encoder chaque position et d'appliquer le mécanisme d'attention afin de relier deux mots distants, qui peuvent ensuite être parallélisés, accélérant ainsi l'apprentissage. Il applique un mécanisme d'auto-attention. Dans l'exemple précédent « Je suis arrivé à la banque après avoir traversé la rivière », pour déterminer que le mot « banque » fait référence à la rive d'une rivière et non à une institution financière, le Transformer peut apprendre à s'occuper immédiatement du mot « rivière », et prendre cette décision en une seule étape.

Pour calculer la représentation suivante pour un mot donné ("banque" par exemple), le Transformer la compare à tous les autres mots de la phrase. Le résultat de ces comparaisons est un score d'attention pour chaque mot de la phrase. Ces scores d'attention déterminent la contribution de chacun des autres mots à la prochaine représentation de « banque ». Dans cet exemple, la « rivière » qui suscite l'ambiguïté pourrait recevoir un score élevé d'attention lors du calcul d'une nouvelle représentation pour « banque ». Les scores d'attention sont ensuite utilisés comme pondération pour une moyenne pondérée de la représentation de tous les mots, qui est ensuite introduite dans un réseau entièrement connecté afin de générer une nouvelle représentation pour « banque », reflétant ainsi que la phrase parle d'une rive (Référence : AI, Jeudi, 31 août 2017).

2.2.6 L'attention avec le Transformer

Il est possible de visualiser les autres parties d'une phrase prises en charge par le réseau lors du traitement ou de la traduction d'un mot donné, ce qui permet de mieux comprendre la manière dont l'information circule dans le réseau. Soit les phrases suivantes et leurs traductions françaises :

*The animal didn't cross the street because it was too tired.
L'animal n'a pas traversé la rue parce qu'il était trop fatigué.*

*The animal didn't cross the street because it was too wide.
L'animal n'a pas traversé la rue parce qu'elle était trop large.*

FIGURE 2.2 – Traduction de l'anglais vers le français (Référence : AI, Jeudi, 31 août 2017)

Cette stratégie d'attention personnelle aborde la question de la résolution de co-référence, par exemple

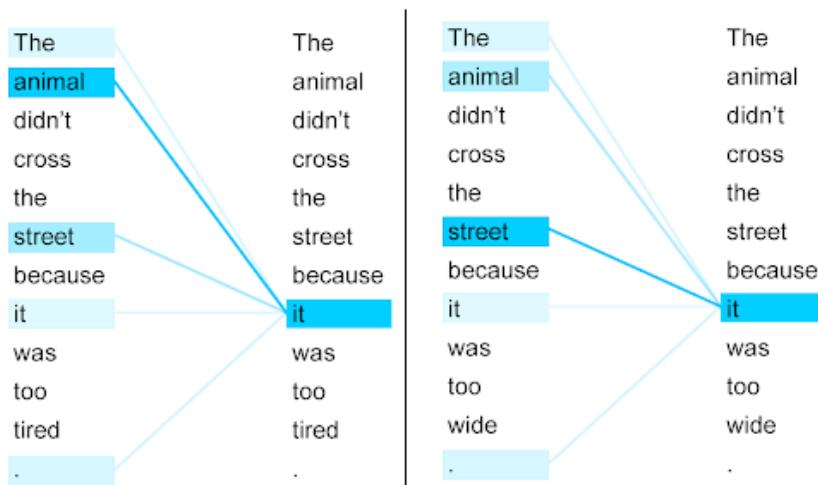


FIGURE 2.3 – La distribution de l'attention du mot « it » avec un Transformer (Référence : AI, Jeudi, 31 août 2017)

- Dans la mesure où le modèle peut visualiser d'autres parties d'une phrase que le réseau prend en compte lors du traitement ou de la traduction d'un mot donné, il permet de mieux comprendre la façon dont les informations transitent par le réseau.
- Visualiser les mots pris en charge par le codeur lors du calcul de la représentation finale du mot « it » permet de mieux comprendre la façon dont le réseau a pris sa décision. Dans une de ses étapes, le Transformer a clairement identifié les deux noms auxquels il pourrait faire référence et la quantité d'attention respective reflète son choix dans différents contextes.

Le mécanisme d'attention traditionnel a résolu en grande partie la première dépendance en donnant au décodeur l'accès à la totalité de la séquence d'entrée. L'idée nouvelle du Transformer est d'étendre ce mécanisme également au traitement des phrases d'entrée et de sortie. Au lieu d'aller de gauche à droite en utilisant des RNN, pourquoi ne pas permettre simplement au codeur et au décodeur de voir l'intégralité de la séquence d'entrée en une seule fois, en modélisant directement ces dépendances en utilisant l'attention.

2.2.7 Traitement avec RNN

Avec les RNNs, il faut y aller un mot à la fois pour arriver à la dernière cellule de mots. Si le réseau est formé avec une longue portée avec cela, il faudrait peut-être revenir sur beaucoup de pas pour s'en souvenir. Cette séquentialité est un obstacle à la parallélisation du processus. De plus, dans les cas où de telles séquences sont trop longues, le modèle a tendance à oublier le contenu des positions distantes en séquence ou à le mélanger avec le contenu des positions suivantes. Lorsqu'une séquence est traitée à l'aide de RNN, chaque état masqué dépend de l'état masqué précédent. Cela devient un problème majeur pour les GPU. Les RNN sont douloureusement inefficaces et lents sur le GPU.

L'attention résout tous les problèmes des architectures RNN et codeur-décodeur. Cependant, le Transformer tente de remédier à quelques défauts des RNN. Chaque fois que des dépendances à long terme sont impliquées, nous savons que les RNN (même en utilisant des hacks tels que des portes bidirectionnelles, multicouches, basées sur la mémoire LSTM/GRU) souffrent d'un problème de gradient en voie de disparition.

Les modèles récurrents factorisent généralement le calcul le long des positions de symbole des séquences d'entrée et de sortie. En alignant les positions sur les pas du temps de calcul, ils génèrent une séquence d'états cachés h_t , en fonction de l'état caché précédent h_{t-1} et de l'entrée pour la position t. Cette nature intrinsèquement séquentielle exclut la parallélisation dans les exemples d'apprentissage, ce qui devient critique pour les séquences plus longues, car les contraintes de mémoire limitent le traitement par lots entre les exemples. La contrainte fondamentale du calcul séquentiel reste toutefois.

2.2.8 Traitement avec CNN

Les modèles autorégressifs (un modèle autorégressif est simplement un modèle à rétroaction qui prédit les valeurs futures à partir des valeurs passées) impliquent des calculs séquentiels, extrêmement coûteux en termes de calcul. Afin de réduire ces coûts, les modèles tels que ByteNet et ConvS2S utilisent des CNN faciles à paralléliser, ce qui n'est pas possible avec les RNN, bien qu'ils semblent mieux convenir à la modélisation de séquence. La convolution active la parallélisation pour le traitement du processeur graphique.

Les gens ont commencé à essayer de résoudre le problème de dépendance avec des convolutions seq2seq pour une solution au RNN. Une longue séquence est prise et des convolutions sont appliquées. L'inconvénient est que les approches de CNN nécessitent beaucoup de couches pour capturer les dépendances à long terme dans la structure séquentielle des données, sans réussir toujours dans la tentative ou en rendant le réseau si grand qu'il finit par devenir impraticable. La capacité d'un modèle à apprendre les dépendances entre les positions diminue rapidement avec la distance, ce qui rend fondamental de trouver une approche capable de traiter de manière parallèle ces données séquentielles, et c'est là que Transformer intervient.

2.2.9 Dépendances à long termes

L'apprentissage des dépendances à longue distance est un défi majeur dans de nombreuses tâches de transduction de séquences. Un facteur clé affectant la capacité à apprendre de telles dépendances est la longueur des chemins que les signaux avant et arrière doivent traverser dans le réseau. Plus ces chemins sont courts entre toute combinaison de positions dans les séquences d'entrée et de sortie, plus il est facile d'apprendre les dépendances à longue portée.

Le modèle RNN n'est pas inférieur aux modèles ConvS2S et Transformer lorsqu'il crée des dépendances à longue portée, pourquoi n'est-il pas aussi efficace que ConvS2S et Transformer ? Cela est toujours dû au manque de représentation du modèle RNN. Alors que la longueur de la séquence augmente, RNN établit des dépendances à long terme, mais sa perte d'informations est devenue très grave. L'attention portée à la séquence RNN semble également collecter beaucoup d'informations, mais il existe un très grave chevauchement d'informations (le contenu dans différentes positions de la séquence a des effets différents sur la sortie du codeur final). En revanche, les informations dans différentes positions de la séquence d'entrée dans le CNN ont le même effet sur la sortie du codeur (le Transformer est similaire). Pour résoudre ce problème, ConvS2S et Transformer ont mis les informations de localisation directement dans l'entrée du modèle.

Théoriquement, les LSTM (et les RNN en général) peuvent avoir une mémoire à long terme. Mais se souvenir de choses pendant de longues périodes reste un défi, et les RNN peuvent toujours avoir des problèmes de mémoire à court terme. En outre, certains mots ont plusieurs sens qui ne se manifestent que dans le contexte. Par exemple, le mot « que », comme dans « Elle est plus grande que moi » et « Je n'ai pas d'autre choix que d'écrire ce billet de blog », ils utilisent « que » de différentes manières. Les jetons de sortie dépendent également les uns des autres. Si la phrase cible n'était « Ni lui ni moi ne connaissions l'apprentissage en profondeur », le symbole « ni » dépend de la clause négative « Ni ».

Dans les Transformers, étant donné que l'attention individuelle est appliquée à chaque mot et à tous les mots ensemble, quelle que soit leur distance, le chemin le plus long possible est un chemin permettant au système de capturer les relations de dépendance lointaines. Le modèle ne contenant ni récurrence ni convolution, pour que celui-ci utilise l'ordre de la séquence, des informations doivent être injectées sur la position relative ou absolue de la séquence.

Il existe essentiellement trois types de dépendances. Les dépendances entrent :

- Les jetons d'entrée et de sortie
- Les jetons d'entrée eux-mêmes
- Les jetons de sortie eux-mêmes

2.2.10 Les difficultés rencontrées avec le modèle Seq2Seq

- La première est la complexité de calcul totale par couche.
- Une autre est la quantité de calcul qui peut être parallélisée, telle que mesurée par le nombre minimum d'opérations séquentielles requises.
- La troisième est la longueur du chemin entre les dépendances à longue portée sur le réseau.

2.2.11 Les solutions apportées du Transformer

Les solutions apportées sont :

- Minimiser la complexité informatique totale par couche. Les couches auto-attentives connectent toutes les positions avec le nombre $O(1)$ d'opérations exécutées séquentiellement.
- Maximiser la quantité de calculs parallélisables, qui est mesurée par le nombre minimal d'opérations séquentielles requises. Pour une longueur de séquence $n >$ dimensionnalité de représentation d , l'attention personnelle ne peut prendre en compte que le voisinage d'une certaine taille r dans la séquence d'entrée centrée autour de la position de sortie respective, augmentant ainsi la longueur de trajet maximale à $O(n/r)$.
- Minimiser la longueur de chemin maximale entre deux positions d'entrée et de sortie quelconques dans un réseau composé des différents types de couche. Plus le chemin entre les combinaisons de positions dans les séquences d'entrée et de sortie est court, plus il est facile d'apprendre les dépendances à longue portée.

Type	Couche	Opération	Longueur
Self-Attention	$O(n^2 * d)$	$O(1)$	$O(1)$
Récurrent	$O(n * d^2)$	$O(n)$	$O(n)$
Convolutionnel	$O(k * n * d^2)$	$O(1)$	$O(\log_k(n))$
Self-attention(restricted)	$O(r * n * d)$	$O(1)$	$O(n/r)$

TABLE 2.1 – Tableau de complexités (Référence : AI, 12 Juin 2017)

Le tableau ci-dessus représente les complexité par couche, par nombre minimal d'opérations séquentielles pour différents types de couche et par la longueur maximale des chemins.

- **n** : est la longueur de la séquence.
- **d** : est la dimension de la représentation.
- **k** : est la taille du noyau des convolutions.
- **r** : est la taille du voisinage dans une attention personnelle limitée.

2.2.12 Comparaison de complexité

En termes de complexité de calcul, les couches d'auto-attention sont plus rapides que les couches récurrentes lorsque la longueur de la séquence n est inférieure à la dimensionnalité de la représentation d , ce qui est le cas le plus souvent utilisé avec les représentations de phrases utilisées par les modèles les plus récents dans les traductions automatiques. Pour améliorer les performances de calcul pour les tâches comportant de très longues séquences, il est possible que l'attention personnelle soit limitée à la considération d'un voisinage de taille r dans la séquence d'entrée centrée autour de la position de sortie respective. Cela augmenterait la longueur maximale du chemin jusqu'à $O(n/r)$. Cette approche sera approfondie lors de travaux futurs.

Les couches convolutives sont généralement plus chères que les couches récurrentes, d'un facteur k . Les convolutions séparables diminuent considérablement la complexité, jusqu'à $O(k * n * d^2)$. Même avec $k = n$, la complexité d'une convolution séparable est égale à la combinaison d'une couche d'attention propre et d'une couche de position wise feed-forward, approche adoptée dans le modèle.

Une couche d'auto-attention connecte toutes les positions avec un nombre constant d'opérations exécutées séquentiellement, alors qu'une couche récurrente nécessite $O(n)$ opérations séquentielles.

Une seule couche convective de largeur de noyau $k < n$ ne connecte pas toutes les paires de positions d'entrée et de sortie. Cela nécessite une pile de couches convolutionnelles $O(n/k)$ dans le cas de noyaux contigus, et $O(\log_k(n))$ dans le cas de convolutions dilatées, ce qui augmente la longueur du plus long chemin entre deux positions dans le réseau.

2.2.13 Comparaison de résultat entre différentes architectures

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	41.29	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1		$3.3 \cdot 10^{18}$
Transformer (big)	28.4	41.8		$2.3 \cdot 10^{19}$

FIGURE 2.4 – Résultat de traduction avec différents systèmes de l'anglais vers le français (WMT 14, newstest2014) (Référence : AI, 12 Juin 2017)

Des expériences sur deux tâches de traduction automatique ont montré que ces modèles étaient de qualité supérieure, tout en effectuant des calculs en parallèle et nécessitaient beaucoup moins de temps de formation avec des résultats bien que d'autres modèles. L'estimation de qualité se fait à l'aide du score bleu. Sur la tâche de traduction anglais-français du WMT 2014, un modèle établit un nouveau score BLEU ultramoderne de 41.8. (Pour plus d'informations concernant le score, consultez la section 2.6 Le score BLEU)

Le Transformer par exemple a pris 3.5 jours de formation avec 8 GPU, alors que cela appris 1 semaine pour le GNMT.

2.3 L'auto-attention

2.3.1 L'attention, c'est tout ce dont vous avez besoin

Ces dernières années, les mécanismes d'attention ont trouvé une large application dans toutes sortes de tâches de traitement du langage naturel basées sur l'apprentissage en profondeur.

Grâce aux recherches plus approfondies sur ce mécanisme, les types d'attention suscités par les chercheurs se sont multipliés. En juin 2017, l'équipe de traduction automatique de Google a publié un article sur arXiv intitulé "L'attention, c'est tout ce dont vous avez besoin", qui a attiré l'attention du secteur. Les mécanismes d'auto-attention sont devenus un sujet intéressant dans la recherche sur l'attention des réseaux de neurones et se sont avérés utiles dans beaucoup de tâches.

Le document influent au titre accrocheur qui a fondamentalement changé le domaine de la traduction automatique. Auparavant, les RNN étaient considérés comme l'architecture de référence pour la traduction. Ce document a surpris tout le monde en présentant le Transformer, un réseau sans récurrence qui n'utilisait que l'attention (ainsi que quelques autres composants) uniquement.

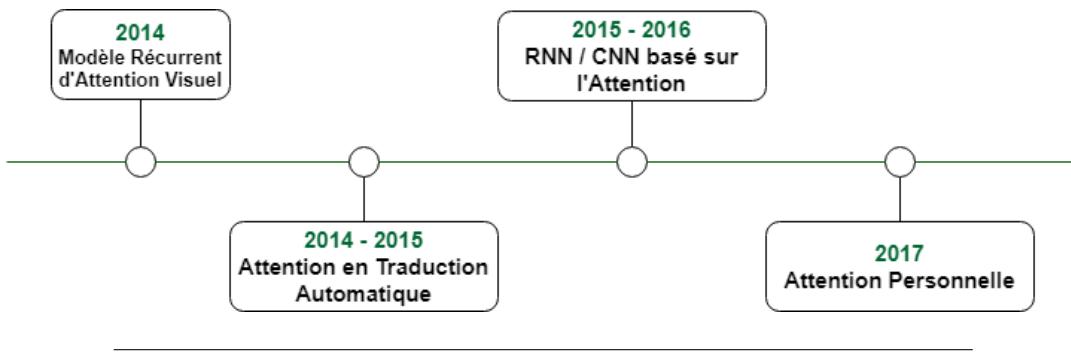


FIGURE 2.5 – Recherche sur les mécanismes d'attention (Référence : ZHANG, 18 Septembre 2018)

2.3.2 Les points importants du papier de recherche

Les points principaux à retenir à propos du papier :

- Il introduit une nouvelle architecture, appelée **Transformer**, qui n'utilise ni les réseaux de neurones convolutionnels (CNN) ni les réseaux de neurones récurrents (RNN) dans sa structure, mais uniquement des mécanismes d'attentions.
- En particulier, il introduit le concept de **mécanisme d'attention multi-têtes**.
- Il décrit une nouvelle façon de positionner les mots.

2.3.3 L'importance du transformer

- Le cadre de modèle RNN et Seq2Seq, sont difficiles à mettre en parallèle et peuvent avoir du mal à apprendre les dépendances à longue portée au sein des séquences d'entrée et de sortie. Le mécanisme d'attention de l'essai remplace RNN pour construire un cadre de modèle complet.

- Le Transformer modélise toutes ces dépendances en utilisant l'attention.
- Au lieu d'utiliser une seule série d'attention, le Transformer utilise plusieurs « têtes » (plusieurs distributions d'attention et plusieurs sorties pour une seule entrée)
- Outre l'attention, le Transformer utilise la normalisation des couches et les connexions résiduelles pour faciliter l'optimisation.
- L'attention ne peut pas utiliser les positions des entrées. Pour résoudre ce problème, le Transformer utilise un codage de position explicite. Des encodages de pistions sont ajoutés aux intégrations d'entrée.
- L'attention personnelle pourrait produire des modèles plus interprétables. Non seulement les têtes d'attention individuelles apprennent clairement à effectuer différentes tâches, mais beaucoup semblent présenter un comportement lié à la structure syntaxique et sémantique des phrases.
- Cette architecture réalise des performances de pointe en matière de traduction anglais-français.
- Réalise nouveaux résultats l'état de l'art dans NMT lors de l'utilisation des WMT 2014 anglais-français des jeux de données. Ce système a obtenu des résultats impressionnants et peut être formé plus rapidement que les modèles courants.

2.3.4 Le Transformer

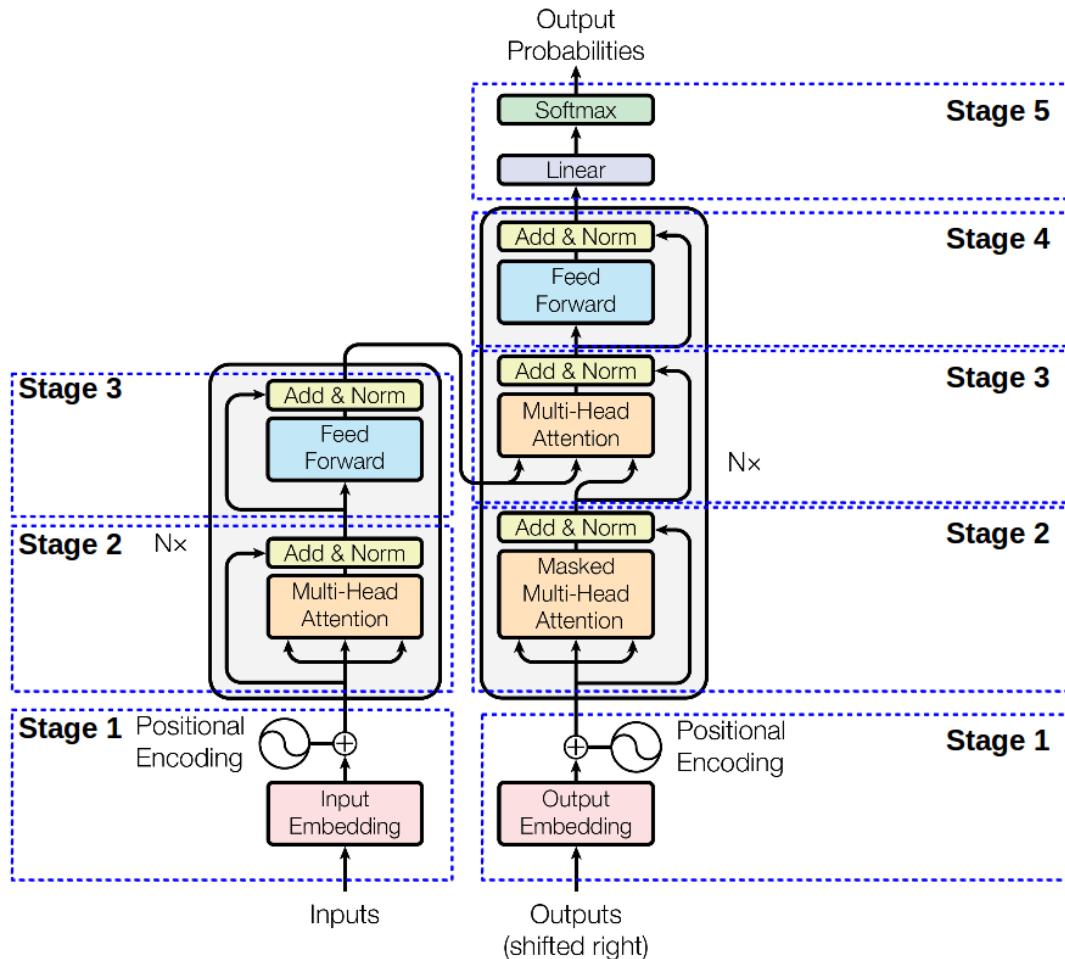


FIGURE 2.6 – Architecture du Transformer (Référence : CHROMIAK, Mardi, 12 Septembre 2017)

Le Transformer utilise toujours la conception de base codeur-décodeur des systèmes de traduction automatique neuronaux traditionnels. Le côté gauche est la couche unique de codeur et le côté droit est le décodeur. Ils sont construits à partir de $N = 6$ couches identiques selon le document(mais c'est possible d'en rajouter plus). Les couches de l'encodeur ont tous une structure similaire, mais ils ne disposent pas du même poids. Les entrées initiales du codeur sont les intégrations de la séquence d'entrées, et les entrées initiales du décodeur sont les intégrations des sorties.

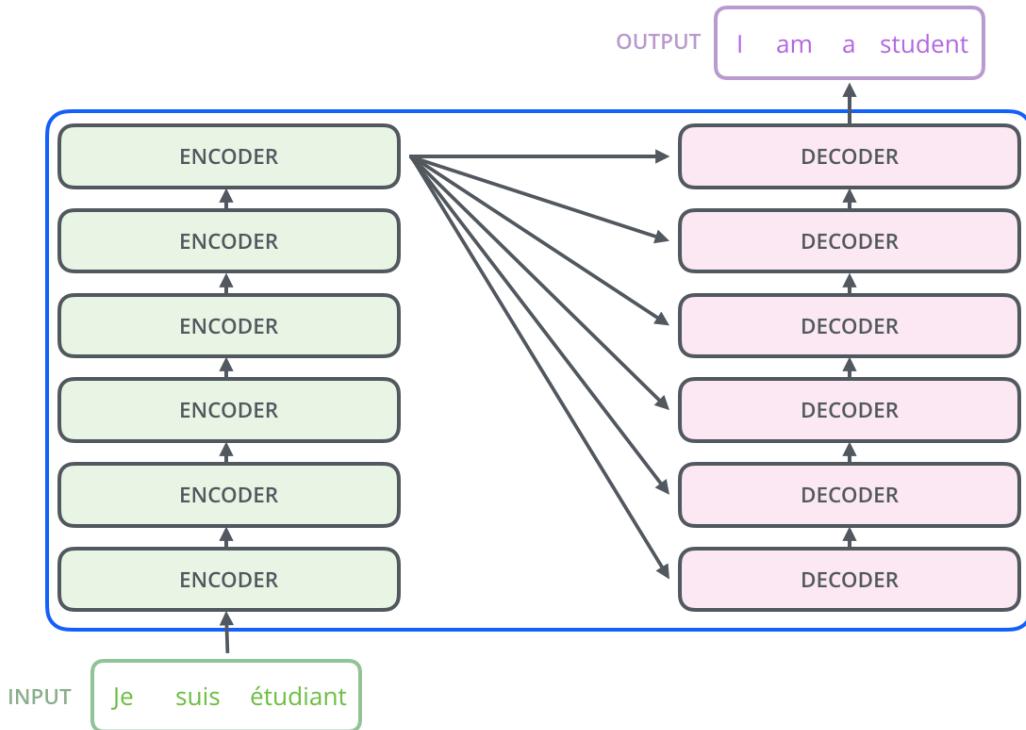


FIGURE 2.7 – Une pile de six encodeurs et de décodeurs (Référence : ALAMMAR, 20 Février 2017)

Les informations sur l'ordre des séquences sont très importantes. Comme il n'y a ni récurrence ni convolution, ces informations sur la position absolue (ou relative) de chaque jeton dans une séquence sont représentées par l'utilisation de "codages de position"

Pour optimiser le réseau profond, l'ensemble du réseau utilise une connexion résiduelle et applique Add et Norm à la couche.

La méthode Multi-Head Attention soulevée par Google utilise plusieurs itérations de calcul pour capturer des informations pertinentes à partir de différents espaces enfants. Ce qui rend l'auto-attention unique, c'est qu'elle ignore la distance entre les mots et calcule directement les relations de dépendance, ce qui la rend capable d'apprendre la structure interne d'une phrase, et plus simplement, de calculer en parallèle.

2.3.5 La couche Encodeur

Le codeur est composé de deux blocs (appelé des sous-couches pour les distinguer des N blocs composant le codeur et le décodeur). L'un est la sous-couche Attention multi-têtes sur les entrées. L'autre est un simple réseau à feed-forward. Entre chaque sous-couche, il existe une connexion résiduelle suivie d'une normalisation de couche. Une connexion résiduelle consiste simplement à saisir l'entrée et à l'ajouter à la sortie du sous-réseau, et est un moyen de faciliter la formation de réseaux profonds.

Les entrées de la couche d'un encodeur (codeur) passent en premier sur la couche d'auto-attention, celle-ci aide le codeur à voir les autres mots de la phrase d'entrée

lorsqu'il code un mot spécifique. Les sorties de la couche d'auto-attention sont envoyées vers un réseau de neurone feedforward. Le même réseau feedforward est appliqué indépendamment à chaque position.

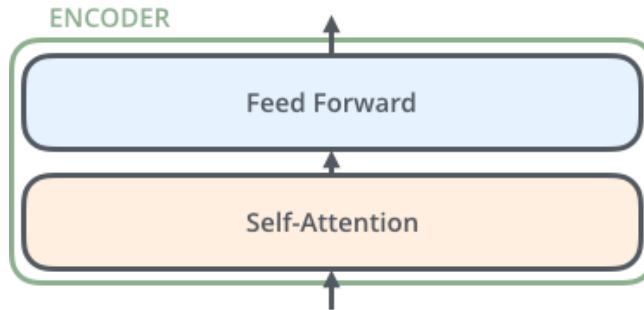


FIGURE 2.8 – Les composants d'une couche d'encodeur(Référence : ALAMMAR, 20 Février 2017)

2.3.6 La couche Décodeur

Le décodeur possède les deux mêmes couches que celui d'un codeur, mais entre elles se trouve une couche d'attention qui l'aide à se concentrer sur les parties pertinentes de la phrase en entrée.

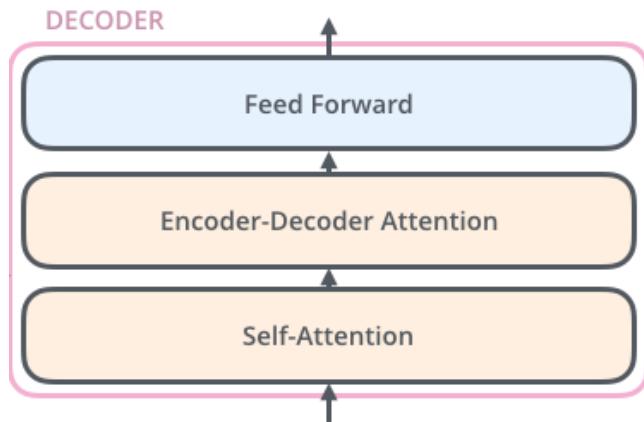


FIGURE 2.9 – Les composants d'une couche de décodeur sans l'attention masquée (Référence : ALAMMAR, 20 Février 2017)

Mais en plus, il possède une couche d'attention multi-têtes appelée réseau « attention masquée multi-têtes ». Ce réseau surveille les états précédents du décodeur et joue donc un rôle similaire à l'état caché du décodeur dans les architectures de traduction automatique traditionnelles.

La raison pour laquelle le bloc d'attention multi-têtes masqué est appelé ainsi, c'est qu'il est nécessaire de masquer les entrées du décodeur à partir de futurs pas de temps pour empêcher les positions d'assister aux positions suivantes. Les décodeurs sont généralement formés pour prédire les phrases en fonction de tous les mots précédant le mot actuel. Si la séquence du décodeur n'est pas décalée, le modèle apprend

à simplement copier l'entrée du décodeur. L'entrée du décodeur est l'intégration de la sortie décalée d'une position pour garantir que la prédiction pour la position i ne dépend que des positions antérieures à / inférieures à i .

Une des raisons est que le modèle ne doit pas apprendre à copier les entrées de décodeur pendant la formation, mais il doit apprendre que, étant donné la séquence de codeur et une séquence de décodeur particulière déjà vue par le modèle, le prochain mot sera prédit.

L'architecture de Transformer est parallèle au moment de la formation et périodique au moment de la production. Pendant la formation, toute la phrase correcte est l'entrée du décodeur, décalée et masquée, afin que le transformateur puisse s'entraîner en parallèle. Dans la production, le système ne peut pas avoir accès à la phrase entière et travaille donc mot par mot.

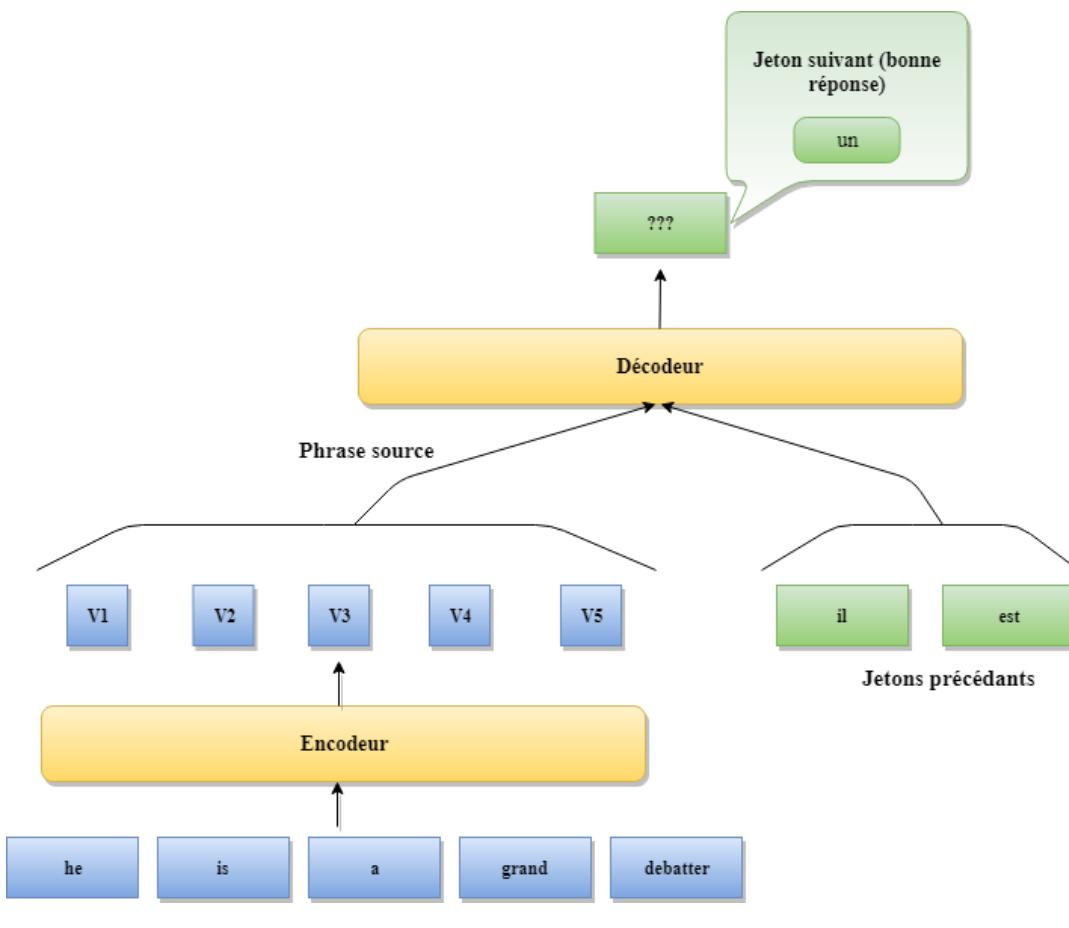


FIGURE 2.10 – La bonne façon de former un décodeur (Référence : KEITAKURITA, 29 decembre 2017)

Lorsque le réseau est formé pour traduire la phrase « he is a grand debatter » avec « il est un grand débatteur », le réseau est entraîné pour prédire « un » arrive après « il est » quand la phrase est « he is a grand debatter ».

La première position de l'entrée du décodeur avec un jeton de début de phrase, car cet endroit serait sinon vide à cause du décalage à droite. De même, un jeton de fin de phrase est ajouté à la séquence d'entrée du décodeur pour marquer la fin de cette séquence. Il est également ajouté à la phrase de sortie.

Dans la figure 2.11, la phrase cible « il est un grand débatteur » est directement copiée dans le décodeur, donc celui-ci connaît les positions futures, d'où le fait qu'il a reconnu directement la prochaine sortie qui est « un ».

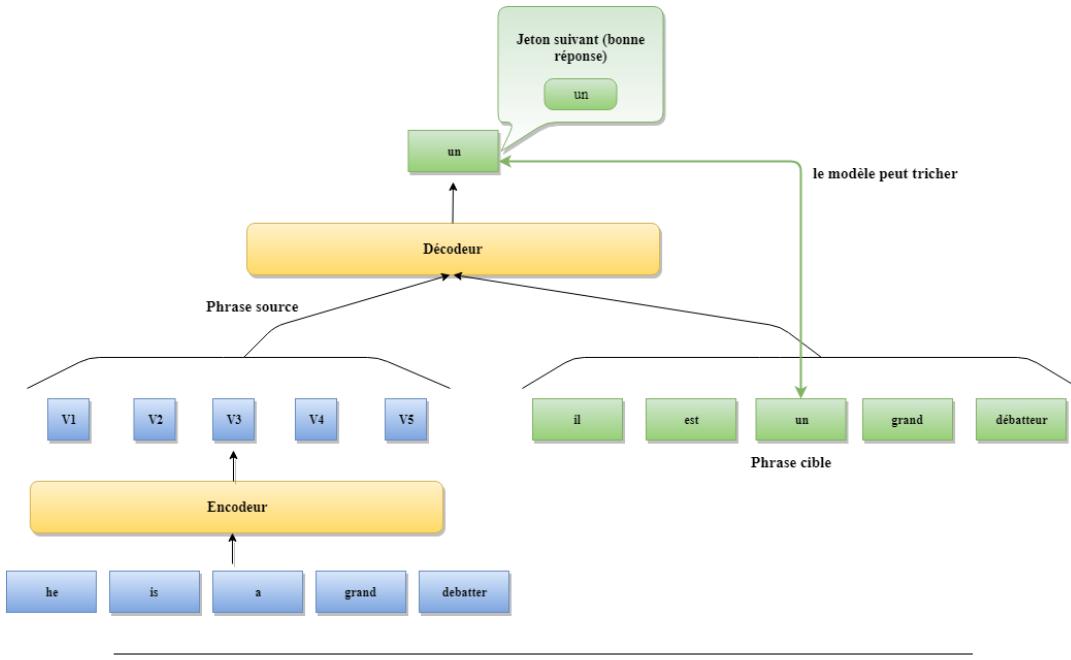


FIGURE 2.11 – La mauvaise façon de former un décodeur (Référence : KEITAKURITA, 29 décembre 2017)

Pour éviter que cela ne se produise, le décodeur masque les jetons « futurs » lors du décodage d'un mot donné. Ce masquage masque les caractéristiques appartenant aux états futurs de la séquence (il s'agit de la couche d'auto-attention masquée). Ceci est spécifique à l'architecture de Transformer car il n'y a pas de RNN pour faire entrer la séquence de manière séquentielle. Le processus d'introduction de l'entrée décalée correcte dans le décodeur est également appelé « Teacher-Forcing ».

Avec cette astuce, Transformer devient autorégressif, donc $p(o_t) = p(o_t | o_1, o_2, \dots, o_{t-1})$ avec o_i indiquant la sortie (traduction générée). La génération de l'état actuel n'est alors en aucune manière conditionnée par les états futurs.

Chaque mot prédit assiste ensuite aux mots précédemment générés du décodeur au niveau de cette couche et à la représentation finale du codeur, similaire à une architecture typique de codeur-décodeur.

Exemple d'attention masquée

Sois la phrase suivante en français « tu t'appelles lebowski », cette phrase sera traduite vers l'anglais « your name is lebowski ». Au moment de générer un nouveau mot, la matrice générée dans la couche d'attention du décodeur devrait sans masquage ressembler comme à la figure 2.12. (Référence : DENOYES, 10 Avril 2019)

	your	name	is	Lebowski
your	-26.845	-13,4225	34,749	-10,0113
name	0.093	-0.00693	0.006189	-0.0062
is	3.435	-495.767	3.75	-53.4913
Lebowski	11.964	-0.02413	8.7227	-0.00061

FIGURE 2.12 – Représentation de mot dans le décodeur sans masque

Maintenant que les valeurs correspondantes aux mots à venir n’ont aucune attention, il faut donc une multiplication avec une matrice de valeur V.

	your	name	is	Lebowski
your	-26.845	0	0	0
name	0.093	-0.00693	0	0
is	3.435	-495.767	3.75	0
Lebowski	11.964	-0.02413	8.7227	-0.00061

FIGURE 2.13 – Représentation de mots dans le décodeur sans attention

Et puis il ne reste plus qu’à obtenir des valeurs moins l’infini.

	your	name	is	Lebowski
your	-26.845	-∞	-∞	-∞
name	0.093	-0.00693	-∞	-∞
is	3.435	-495.767	3.75	-∞
Lebowski	11.964	-0.02413	8.7227	-0.00061

FIGURE 2.14 – Représentation de mots dans le décodeur avec masque

2.3.7 Inférence (prédiction)

Inférer avec ces modèles est différent de la formation. Il faut ré-alimenter le modèle pour chaque position de la séquence de sortie jusqu'au jeton de fin de phrase. Une méthode plus progressive serait :

- Une séquence complète du codeur sera entrée (phrase source) et comme entrée du décodeur, une séquence vide est prise avec seulement un jeton de début de phrase sur la première position. Cela produira une séquence où le premier élément sera pris.
- Cet élément sera ajouté à la deuxième position à la séquence d'entrée de décodeur, qui contient maintenant un jeton de début de phrase et un premier mot/caractère.
- La nouvelle séquence du décodeur dans le modèle. Le deuxième élément de la sortie sera pris et inséré dans la séquence d'entrée du décodeur.
- Cette opération est répétée jusqu'à ce qu'un jeton de fin de phrase soit prédit, qui marquera la fin de la traduction.

Nous voyons qu'il faut plusieurs passages dans le modèle pour traduire une phrase.

2.3.8 L'auto-attention détaillée

L'auto-attention, parfois appelée intra-attention, est la méthode utilisée par les Transformers afin d'intégrer la compréhension d'autres mots pertinents. Le mécanisme d'attention est interprété comme un moyen de calculer la pertinence d'un ensemble de valeurs (informations) en fonction de certaines clés et requêtes. Fondamentalement, le mécanisme d'attention est utilisé pour que le modèle se concentre sur les informations pertinentes en fonction de ce qu'il traite actuellement.

L'auto-attention est une fonction qui mappe l'entrée à 2 éléments (requête, paires clé-valeur) à une sortie. La sortie fournie par la fonction de mappage est une somme pondérée des valeurs. Où les pondérations pour chaque valeur mesurent combien chaque clé d'entrée interagit avec (ou répond) à la requête.

Le codeur utilise les incorporations de la phrase source pour ses clés, ses valeurs et ses requêtes, tandis que le décodeur utilise les sorties du codeur pour ses clés et ses valeurs et les incorporations de la phrase cible pour ses requêtes.

Les requêtes, clé, et valeur, ceci étant des vecteurs :

- **Vecteur de requête** : Un seul vecteur est utilisée pour demander les clés(mot).
- **Vecteur de clé** : Une séquence de vecteurs également appelée mémoire.
- **Vecteur de valeur** : Une séquence de vecteurs où la sortie est agrégée par une combinaison linéaire pondérée. Il détermine la pertinence d'un mot.
- **La sortie** : Un seul vecteur qui est dérivé d'une combinaison linéaire des valeurs en utilisant les probabilités de l'étape précédente comme poids.

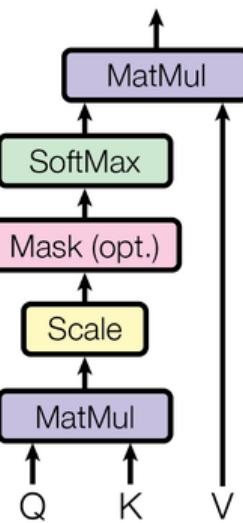


FIGURE 2.15 – Produit scalaire pour le calcul de l'attention (Référence : KEITAKURITA, 29 décembre 2017)

En termes de codeur-décodeur, la requête est généralement l'état caché du décodeur. Tandis que la clé est l'état caché du codeur, la valeur correspondante est un poids normalisé, représentant le niveau d'attention d'une clé. La sortie est calculée comme une somme pondérée, dans ce cas, le produit scalaire de la requête et de la clé est utilisé pour obtenir une valeur.

Transformer imite le mécanisme d'attention classique (connu par exemple de ConvS2S) dans lequel les requêtes de couche codeur-décodeur sont formées de la couche de décodeur précédente, et les clés et valeurs (mémoire) proviennent de la sortie du codeur. Par conséquent, chaque position dans le décodeur peut occuper toutes les positions de la séquence d'entrée.

Bien que l'attention soit un objectif pour de nombreuses recherches, la nouveauté de l'attention des transformateurs réside dans le fait qu'elle est multi-têtes.

Problème avec une seul attention

Self-Attention



FIGURE 2.16 – Une seule attention (Référence : BRIJESH, 09 Décembre 2018)

Avec une seule attention, on ne peut récupérer différentes informations provenant d'autre mots. On aura la même combinaison linéaire. Donc on aimerait bien disposer d'autres informations.

Soit la phrase suivante : « The **animal** did not cross the **street** because **it** was too tired ». Au fur et à mesure que le modèle traite chaque mot, l'auto-attention lui permet de chercher d'autres indices dans la séquence d'entrée afin de trouver des indices permettant d'améliorer l'encodage de ce mot.

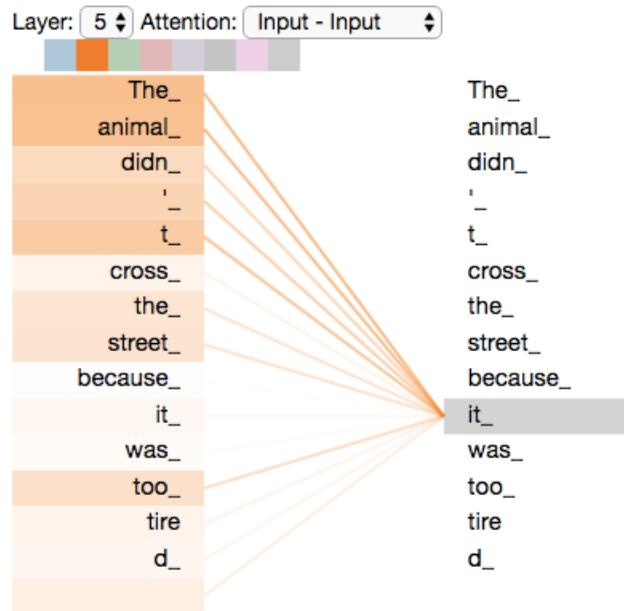


FIGURE 2.17 – Encodage du mot « it » dans l'encodeur n5 (Référence : ALAMMAR, 20 Février 2017)

Dans l'exemple ci-dessus, l'attention se focalise sur le mot « the animal », et donc peut être on aura la même combinaison, nous voudrons bien pouvoir avoir d'autres degrés de concentration sur différents mots.

2.3.9 L'attention multi-tête

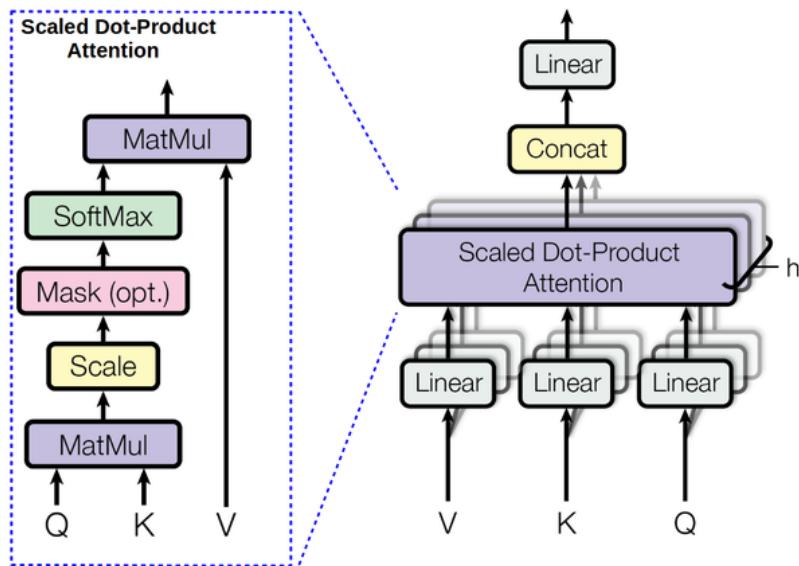


FIGURE 2.18 – L'attention multi-têtes consiste en h couches d'attention fonctionnant en parallèle (Référence : KEITAKURITA, 29 décembre 2017)

Si le calcul se fait sur qu'une seule somme de valeurs pondérées en fonction de l'attention, il serait difficile de saisir différents aspects de l'entrée. Pour résoudre ce problème, le transformateur utilise le bloc Attention multi-têtes. Ce bloc calcule plusieurs sommes pondérées en fonction de l'attention au lieu d'un seul passage de l'attention sur les valeurs, d'où le nom "Multi-Head Attention".

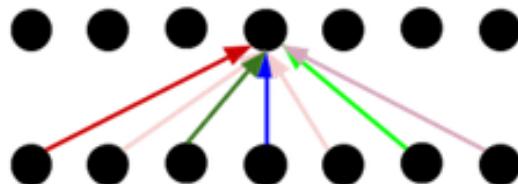


FIGURE 2.19 – Multi-Head attention (Référence : BRIJESH, 09 Décembre 2018)

Dans Multi-Head Attention, il y aura plusieurs têtes d'attention regardant différents mots et connaissant les positions.

L'attention multiple est un mécanisme qui permet d'améliorer la performance de la couche d'attention de deux façons :

- Il étend la capacité du modèle à se concentrer sur les différentes positions. Chaque tête utilise différentes transformations linéaires, et différentes têtes peuvent apprendre différentes relations.
- Il donne à la couche d'attention plusieurs « sous espace de représentation ». Avec une attention multiple, plusieurs matrices de pondération requête, clé et

valeur sont créées. Chacun de ces ensembles est initialisé de manière aléatoire. Après la formation, chaque ensemble est utilisé pour projeter les incorporations d'entrée (ou les vecteurs des encodeurs/décodeurs inférieurs) dans un sous-espace de représentation différent. Le Transformer utilise huit têtes d'attention.

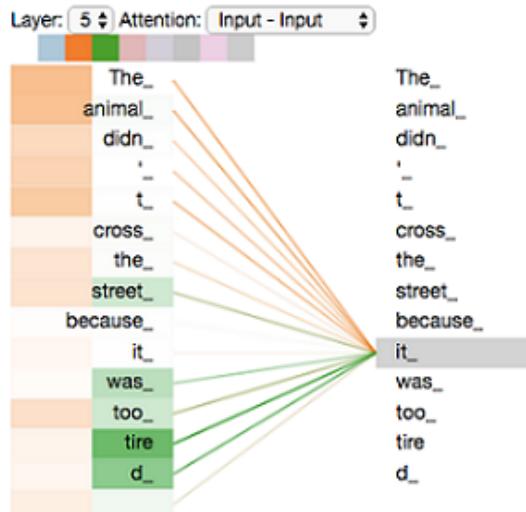


FIGURE 2.20 – Concentration sur le mot « it » sur 2 têtes (Référence : ALAMMAR, 20 Février 2017)

Lors d'encodage du mot « it », une des têtes d'attention se concentre plus sur « animal », tandis que l'autre se concentre sur « tired », la représentation du mot « it » par le modèle se retrouve dans une partie de la représentation « animal » et « tired ».

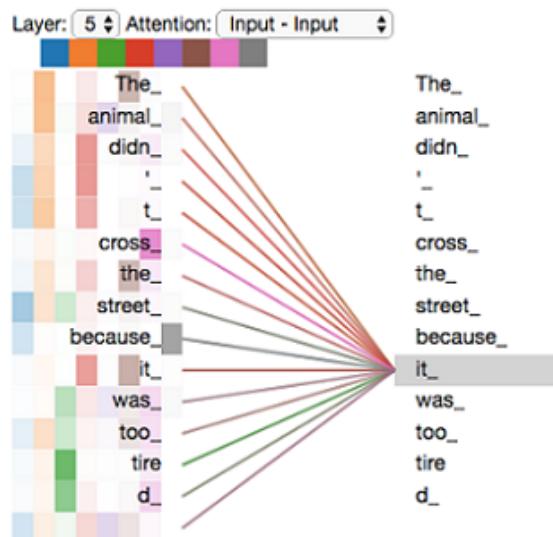


FIGURE 2.21 – Concentration sur le mot « it » sur 8 têtes. La référence est plus difficile à déterminer. (Référence : ALAMMAR, 20 Février 2017)

2.3.10 L'application de l'auto-attention

Le Transformer utilise l'attention de plusieurs têtes de trois manières différentes (Référence : BRIJESH, 09 Decembre 2018) :

1. Encodeur auto-attention

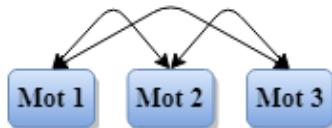


FIGURE 2.22 – Encodeur auto-attention : tout regarde tout

Dans la phase du codeur, le transformateur génère d'abord une représentation (incorporation) initiale pour chaque mot de la phrase d'entrée. Ensuite, pour chaque mot, l'auto-attention agrège les informations de tous les autres mots dans le contexte de la phrase et crée une nouvelle représentation. Le processus est répété pour chaque mot de la phrase. La construction successive de nouvelles représentations, basée sur les précédentes, est répétée plusieurs fois et en parallèle pour chaque mot.

Le codeur contient en interne des couches « auto-attention ». Dans une couche auto-attentive, toutes les clés, valeurs et requêtes proviennent du même endroit, c'est-à-dire le résultat de la couche précédente du codeur. L'entrée à l'attention de plusieurs têtes est la séquence d'entrée elle-même (les clés, les valeurs et également les requêtes dans différentes têtes transformées linéaires). Chaque position dans le codeur peut concernez(traiter) toutes les positions de la couche précédente de l'encodeur.

2. Encodeur-Décodeur attention

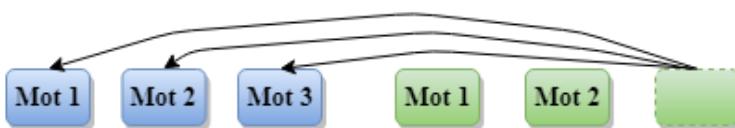


FIGURE 2.23 – Encodeur-Décodeur attention

Dans les couches « attentions codeur-décodeur », les requêtes proviennent de la couche de décodeur précédente, et les clés et les valeurs de mémoires proviennent de la sortie du codeur. Cela permet à chaque position du décodeur d'assister à toutes les positions de la séquence d'entrée. Le décodeur s'occupe des mots précédemment générés du décodeur et de la représentation finale du codeur, c'est à dire que le mot qui doit être généré, celui-ci regardera tous les mots de l'encodeur.

3. Décodeur auto-attention

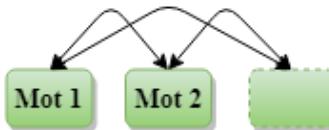


FIGURE 2.24 – Décodeur auto-attention

Une séquence entre dans le premier encodeur, la sortie prédit un élément, puis toute la séquence est repassée dans l'encodeur et l'élément prédit dans le décodeur en parallèle afin de générer un deuxième élément, puis à nouveau la séquence dans l'encodeur et tous les éléments déjà prédits dans le décodeur en parallèle, le mot a généré regarde tous les mots déjà générés par le décodeur, et ceux déjà générés se regardent tous entre eux. Jusqu'à prédire en sortie un <fin de séquence>.

2.3.11 Connexion résiduelle

Les modèles d'apprentissage en profondeur sont difficiles à former, et plus il y a de couches, plus cela devient difficile. Afin de traiter ce problème, l'idée est de conserver une copie intacte des données qui sont transmises aux couches supérieures. Ceci est très important pour conserver les informations relatives à la position ajoutée à la représentation / incorporation en entrée sur le réseau. Le réseau a affiché des résultats catastrophiques lors de la suppression des connexions résiduelles. Il fonctionne en supposant que la fonction à modéliser (les poids du réseau) est plus proche d'un mapping d'identité que d'un mapping nul, de sorte que les perturbations sur cette modélisation devraient être plus faciles à détecter lors du référencement de l'identité.

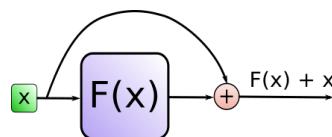


FIGURE 2.25 – Connexion résiduelle (Référence : RICARDOKLEINK-LEIN, 16 Novembre 2017(c))

De plus, ce type de connexions nous permet d'avoir des modèles beaucoup plus profonds avec à peine des coûts de calcul supplémentaires.

2.3.12 Position-wise Feed-Forward Networks

En plus des sous-couches de l'attention, chacune des couches du codeur et du décodeur contient un réseau feed-forward entièrement connecté (ffn), qui est appliqué à chaque position séparément et à l'identique. Cela consiste en deux transformations linéaires avec une activation ReLU entre les deux.

Formule du FFN pour la couche encodeur et décodeur (Référence : AI, 12 Juin 2017).

$$FFN(x) = \max(0, xW_1 + b_1)W_2 + b_2 \quad (2.1)$$

La dimensionnalité de l'entrée et de la sortie est : $d_{model} = 512$, et la couche interne a des dimensions $d_{ff} = 2048$, et cela dépend du modèle

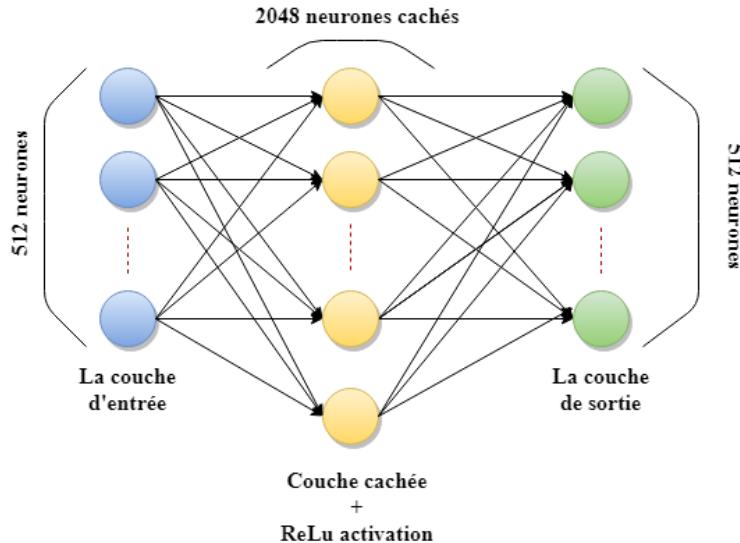


FIGURE 2.26 – Réseau feedforward du transformer basique

L'intérêt de ce module est qu'il est appliqué positionnellement à l'entrée, cela signifie que le même réseau de neurones s'applique à chaque vecteur « jeton » appartenant à la séquence de phrase.

2.3.13 Normalisation

Elle est conçue comme un moyen d'accélérer la convergence de la formation du réseau de neurones, ce qui nécessite généralement des temps incroyablement longs. Le Transformer utilise la normalisation par couche celle-ci consiste ici à extraire des informations statistiques au sein de chaque passe avant que la non-linéarité ne se produise.

Elle peut être facilement appliquée aux réseaux CNN et aux réseaux entièrement connectés, et nécessite juste un petit réglage pour les réseaux RNN, ce qui le rend adapté à toutes les architectures étudiées jusqu'à présent.

2.3.14 Incorporation et Softmax

Des systèmes d'intégration sont utilisés pour convertir les jetons d'entrée et les jetons de sortie en vecteurs de dimension d_{model} . Les fonctions habituelles de transformation linéaire et de softmax acquises pour convertir la sortie du décodeur en probabilités prédites au prochain jeton sont utilisées.

2.3.15 La puissance du Transformer

Chaque bloc codeur ne fait qu'un tas de multiplications matricielles suivies de quelques transformations élémentaires. C'est pourquoi le Transformer est si rapide, tout n'est que multiplication par matrice en parallèle. Le fait est qu'en empilant ces transformations les unes sur les autres, un réseau très puissant est créé.

2.3.16 L'encodage des positions

Dans RNN (LSTM), la notion de pas de temps est codée dans la séquence car les entrées/sorties défilent une par une. Contrairement aux réseaux récurrents, le réseau multi-têtes d'attention ne peut naturellement pas utiliser la position des mots dans la séquence d'entrée. Sans codage de position, la sortie du réseau d'attention à têtes multiples serait la même pour les phrases « J'aime les chats plus que les chiens » et « J'aime les chiens plus que les chats ». Les codages de position codent explicitement les positions relatives / absolues des entrées en tant que vecteurs et sont ensuite ajoutés aux inclusions d'entrée.

Disons que la séquence d'entrée brute peut être décrite comme étant $x = (x_1, x_2, \dots, x_m)$ et qu'elle est convertie en une représentation incorporée $w = (w_1, w_2, \dots, w_m)$ avec $w_j \in R^f$. Chaque w_j est un vecteur colonne de la matrice incorporant appartenant à l'espace R^V , avec V le nombre d'incorporations et f le nombre de caractéristiques de chaque incorporation.

Le point consiste à ajouter une structure séquentielle aux incorporations. Pour y parvenir, à la matrice d'origine de manière élémentaire est ajouté un vecteur $p = (p_1, p_2, \dots, p_m)$, avec $p_j \in R^f$.

Le vecteur résultant est $e = (w_1 + p_1, w_2 + p_2, \dots, w_m + p_m)$, ce qui implicitement permet de dire au réseau neuronal qu'il existe une structure séquentielle sous-jacente.

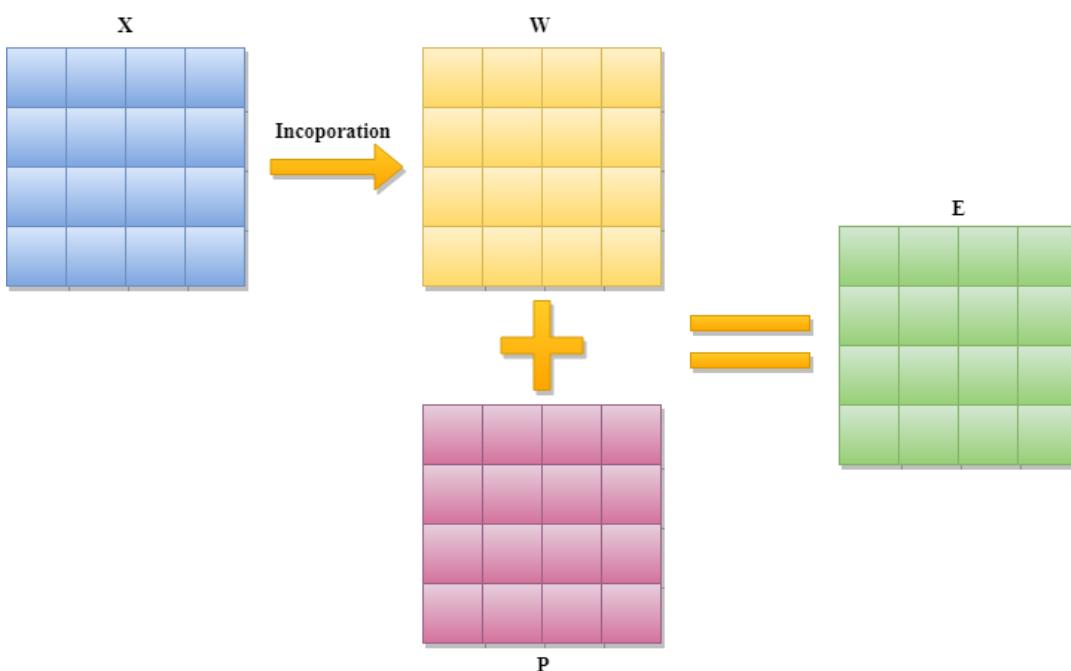


FIGURE 2.27 – Exemple d'encodage du Transformer

La matrice de codage de position est une constante dont les valeurs sont définies. Lorsqu'elle est ajoutée à la matrice d'incorporation, chaque mot incorporé est modifié d'une manière spécifique à sa position. En général, l'ajout d'encodages de position aux incorporations d'entrée est un sujet assez intéressant. Une solution consiste à intégrer la position absolue des éléments d'entrée (comme dans ConvS2S).

Les auteurs proposent de coder le temps en onde sinusoïdale, en tant qu'entrée supplémentaire. Ce signal est ajouté aux entrées et aux sorties pour représenter le temps passé. Cependant, les auteurs utilisent des "fonctions sinus et cosinus de fréquences différentes". La version "sinusoïdale" est assez compliquée, tout en offrant des performances similaires à la version en position absolue. Elles donnent de petites valeurs entre 0 et 1

Les encodages de position sont ajoutés aux incorporations d'entrée situées au bas des piles de codeurs et de décodeurs. Les encodages de position ont la même dimension d_{model} que les incorporations, de sorte que les deux puissent être additionnés.

Formule d'encodage de position (Référence : AI, 12 Juin 2017).

$$\begin{aligned} PE(pos, 2i) &= \sin(pos/10000^{2i/d_{model}}) \\ PE(pos, 2i+1) &= \cos(pos/10000^{2i/d_{model}}) \end{aligned} \quad (2.2)$$

Pos fait référence à l'ordre dans la phrase, et **i** à la position le long de la dimension du vecteur d'inclusion.

La fonction sinus c'est pour les valeurs paire de i, et cosinus pour les valeurs impaires.

Ce qui donne la matrice suivante à sommer avec la matrice des embeddings de chaque mot

$$\left(\begin{array}{c|c|c|c|c} & & & \xleftarrow{\hspace{1cm}} & \\ \text{dmodel} & \xrightarrow{\hspace{1cm}} & & & \dots \\ \hline \begin{matrix} \sin\left(\frac{0}{10000^{\frac{0}{dmodel}}}\right) \\ \sin\left(\frac{1}{10000^{\frac{0}{dmodel}}}\right) \\ \sin\left(\frac{2}{10000^{\frac{0}{dmodel}}}\right) \\ \sin\left(\frac{3}{10000^{\frac{0}{dmodel}}}\right) \\ \sin\left(\frac{4}{10000^{\frac{0}{dmodel}}}\right) \\ \sin\left(\frac{5}{10000^{\frac{0}{dmodel}}}\right) \end{matrix} & \begin{matrix} \cos\left(\frac{0}{10000^{\frac{0}{dmodel}}}\right) \\ \cos\left(\frac{1}{10000^{\frac{0}{dmodel}}}\right) \\ \cos\left(\frac{2}{10000^{\frac{0}{dmodel}}}\right) \\ \cos\left(\frac{3}{10000^{\frac{0}{dmodel}}}\right) \\ \cos\left(\frac{4}{10000^{\frac{0}{dmodel}}}\right) \\ \cos\left(\frac{5}{10000^{\frac{0}{dmodel}}}\right) \end{matrix} & \begin{matrix} \sin\left(\frac{0}{10000^{\frac{2}{dmodel}}}\right) \\ \sin\left(\frac{1}{10000^{\frac{2}{dmodel}}}\right) \\ \sin\left(\frac{2}{10000^{\frac{2}{dmodel}}}\right) \\ \sin\left(\frac{3}{10000^{\frac{2}{dmodel}}}\right) \\ \sin\left(\frac{4}{10000^{\frac{2}{dmodel}}}\right) \\ \sin\left(\frac{5}{10000^{\frac{2}{dmodel}}}\right) \end{matrix} & \begin{matrix} \cos\left(\frac{0}{10000^{\frac{2}{dmodel}}}\right) \\ \cos\left(\frac{1}{10000^{\frac{2}{dmodel}}}\right) \\ \cos\left(\frac{2}{10000^{\frac{2}{dmodel}}}\right) \\ \cos\left(\frac{3}{10000^{\frac{2}{dmodel}}}\right) \\ \cos\left(\frac{4}{10000^{\frac{2}{dmodel}}}\right) \\ \cos\left(\frac{5}{10000^{\frac{2}{dmodel}}}\right) \end{matrix} & \dots \\ \hline \end{array} \right)$$

FIGURE 2.28 – Matrice d'encodage de position (Référence : DE-NOYES, 10 Avril 2019)

En dessous de l'encodage de position s'ajoutera une onde sinusoïdale basée sur la position.

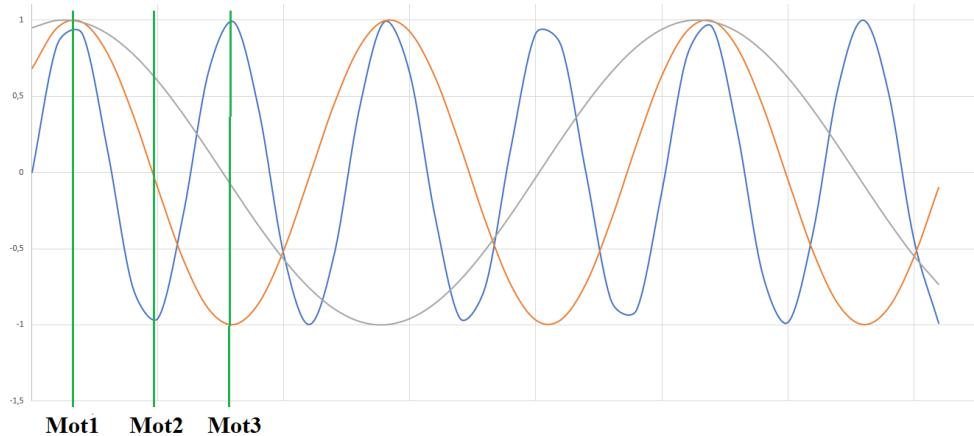


FIGURE 2.29 – Signaux sinusoïdaux (Référence : DENOYES, 10 Avril 2019)

La fréquence des signaux sinusoïdaux varie d'une dimension à l'autre, de sorte que la caractéristique i avec une petite valeur a une période inférieure à celle de i avec une grande valeur. C'est la seule mention d'une structure séquentielle dans les données transmises à Transformer. Des valeurs en réalité sont récupérées comme vu dans la figure 2.29.

Les longueurs d'onde forment une progression géométrique de 2π à $10000 * 2\pi$. Cette fonction a été choisie car par supposition cela permettrait au modèle d'apprendre facilement à y assister par positions relatives, car pour tout décalage fixe k , PE_{pos+k} peut être représenté sous la forme d'une fonction linéaire de $PE[pos]$. Puisque $PE[pos + k]$ peut être représenté comme une fonction linéaire de $PE[pos]$, de sorte que la position relative entre différentes imbrications puisse être facilement déduite. Cela permet au modèle d'apprendre facilement à tenir compte des positions relatives des jetons par rapport à la séquence. Bien que les auteurs aient tenté d'utiliser des codages de position appris, ils ont constaté que ces codages prédéfinis fonctionnaient tout aussi bien.

Une expérience basée sur l'utilisation d'embeddings positionnels a été faite. Les deux versions produisaient des résultats presque identiques. Mais la version sinusoïdale, peut permettre au modèle de s'extraire pour séquencer des longueurs plus longues que celles rencontrées lors de la formation.

Pourquoi sinus et cosinus

L'utilisation de la fonction sin où la fréquence est augmentée de deux fois, est très similaire à l'encodage d'un nombre sous une forme binaire. En binaire, la représentation d'un nombre est fragmentée (matrice avec énormément de 0) en utilisant seulement deux chiffres. Utiliser seulement 2 chiffres avec des flotteurs serait un gaspillage, donc en utilisant la fonction sin, le même effet est obtenu. Et les réseaux neuronaux s'adaptent probablement mieux à la représentation creuse (Les matrices creuses à l'inverse des matrices denses, ce sont des matrices qui contiennent beaucoup de 0. Donc la matrice creuse peut être compressée pour accélérer l'apprentissage) et continuer des nombres, ce qui est exactement ce que la méthode sin permet d'obtenir.

2.3.17 Analyse et tests

	N	d_{model}	d_{ff}	h	d_k	d_v	P_{drop}	ϵ_{ls}	train steps	PPL (dev)	BLEU (dev)	params $\times 10^6$
base	6	512	2048	8	64	64	0.1	0.1	100K	4.92	25.8	65
(A)										5.29	24.9	
										5.00	25.5	
										4.91	25.8	
										5.01	25.4	
(B)									16	5.16	25.1	58
									32	5.01	25.4	60
(C)									2	6.11	23.7	36
									4	5.19	25.3	50
									8	4.88	25.5	80
									256	5.75	24.5	28
									1024	4.66	26.0	168
									1024	5.12	25.4	53
									4096	4.75	26.2	90
									0.0	5.77	24.6	
(D)									0.2	4.95	25.5	
									0.0	4.67	25.3	
									0.2	5.47	25.7	
									positional embedding instead of sinusoids	4.92	25.7	
big	6	1024	4096	16				0.3	300K	4.33	26.4	213

FIGURE 2.30 – Une série de test concernant l'architecture Transformer de l'anglais vers l'allemand (Référence : AI, 12 Juin 2017)

Les auteurs ont mené une série de tests au cours desquels ils ont discuté de la recommandation de $N = 6$ couches de taille de modèle 512 sur la base de $h = 8$ têtes avec des dimensions de clé, valeurs de 64, et avec des pas de 100.000(100K).

Les tests réalisés en utilisant PPL (c'est l'exponentiation de l'entropie, plus elle est faible mieux c'est, et plus elle est haute plus c'est mauvais) et BLEU.

Il est également indiqué que la fonction de compatibilité de produit scalaire pourrait être optimisée d'avantage car la qualité du modèle est réduite avec un d_k plus petit (rangée B).

Les codages de position sinusoïdaux fixes proposés sont supposés produire un score presque égal par rapport aux codages de position appris. Dans la rangée (E), le codage de position sinusoïdal est remplacé par des positions appris, et nous observons des résultats presque identiques à ceux du modèle de base.

Dans la ligne (B), nous observons que réduire la taille de la clé d'attention d_k nuit à la qualité du modèle.

Nous observons en outre aux lignes (C) et (D) que les modèles plus grands sont préférables et que le dropout est très utile pour éviter les surajustements.

Dans la ligne big, on remarque qu'avec une dimension plus longue ainsi qu'un réseau plus grand et avec 16 tête que le modèle à réaliser de bonne qualité en 300k

2.4 L'architecture du Transformer

Les figures suivantes proviennent de la référence : ALAMMAR, 20 Février 2017.

2.4.1 L'incorporation

Chaque mot d'une phrase est converti en un vecteur grâce à un algorithme d'intégration de mot.



FIGURE 2.31 – Intégration des mots en des vecteurs/tenseurs (en réalité une taille de 512)

L'incorporation se fait dans la première pile de l'encodeur (la plus basse). Les autres encodeurs reçoivent un vecteur de taille 512, qui est en général la sortie d'un encodeur inférieur, sauf dans le premier encodeur qui reçoit le mot incorporé. La taille de cette liste est un hyper paramètre qui peut être déterminé, mais en règle générale, c'est la longueur de la phrase la plus longue du jeu de données. (Pour plus d'informations 2.8.2 Word Embeddings)

2.4.2 L'encodage des positions

Le transformateur ajoute un vecteur d'encodage de position, à chaque intégration d'entrée. Ces vecteurs suivent un modèle spécifique appris par le modèle, ce qui l'aide à déterminer la position de chaque mot ou la distance entre différents mots de la séquence. L'intuition ici est que l'ajout de ces valeurs aux incorporations crée des distances significatives entre les vecteurs d'intégration une fois projetés dans des vecteurs Q/K/V et pendant l'attention du produit scalaire.

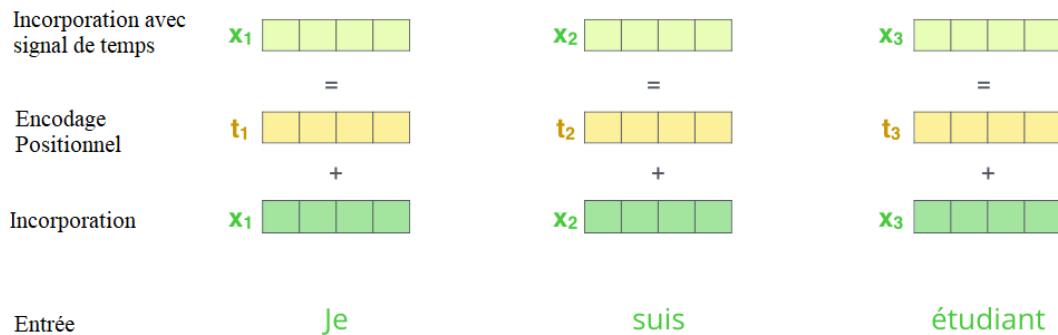


FIGURE 2.32 – Utilisation de l'encodage positionnel

Supposons que l'incorporation a une dimensionnalité de 4, les encodages de position réels pourraient ressembler comme dans la figure suivante :

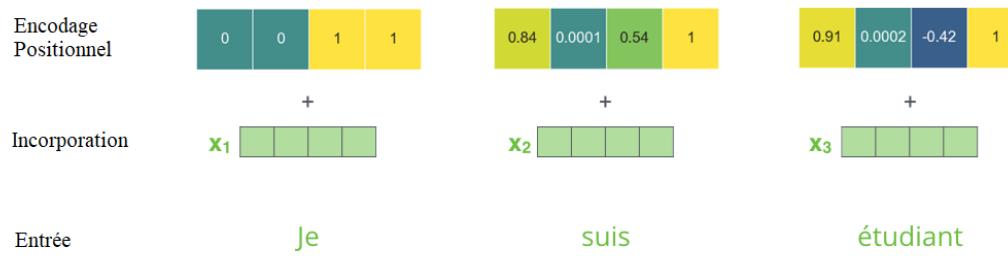


FIGURE 2.33 – Un exemple réel d'encodage de position avec une incorporation d'une taille de 4

Le motif

Dans la figure 2.34, chaque ligne correspond à l'encodage de position d'un vecteur. Ainsi, la première ligne serait le vecteur qui sera ajouté à l'intégration du premier mot dans une séquence d'entrée. Chaque ligne contient 512 valeurs, chacune avec une valeur comprise entre 1 et -1. Ils sont codés en couleur pour une meilleure représentation.

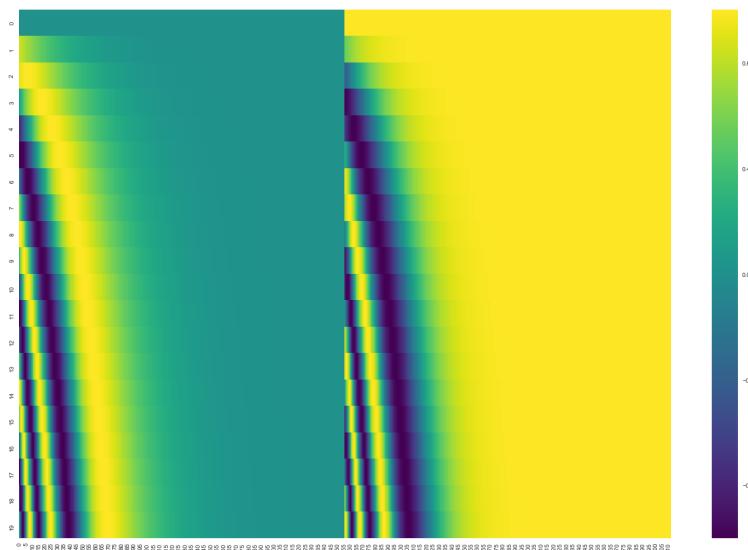


FIGURE 2.34 – Exemple réel d'un encodage de position

Celui-ci possède 20 mots (lignes) avec une taille d'incorporation de 512 (colonnes). Celui-ci est divisé en deux. Les valeurs de la moitié gauche sont générées par une fonction qui utilise le sinus, et la moitié droite est générée par une autre fonction qui utilise le cosinus. Ils sont ensuite concaténés pour former chacun des vecteurs de codage de position.

2.4.3 Le premier encodeur

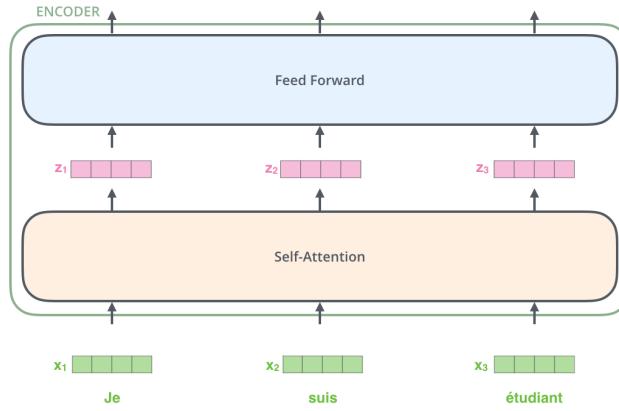


FIGURE 2.35 – Le premier encodeur

On remarque sur la figure 2.35, que chaque mot suit son propre chemin dans l'encodeur. Il existe des dépendances entre ces chemins dans la couche d'auto-intention. La couche feedforward ne dispose pas de ces dépendances, la particularité c'est que les différents chemins peuvent donc être exécutés en parallèle tout en parcourant la couche de feedforward. Ceci est l'une des propriétés clé du Transformer.

Exemple

Voici un exemple de phrase pour voir ce qui se passe dans chaque sous-couche de l'encodeur.

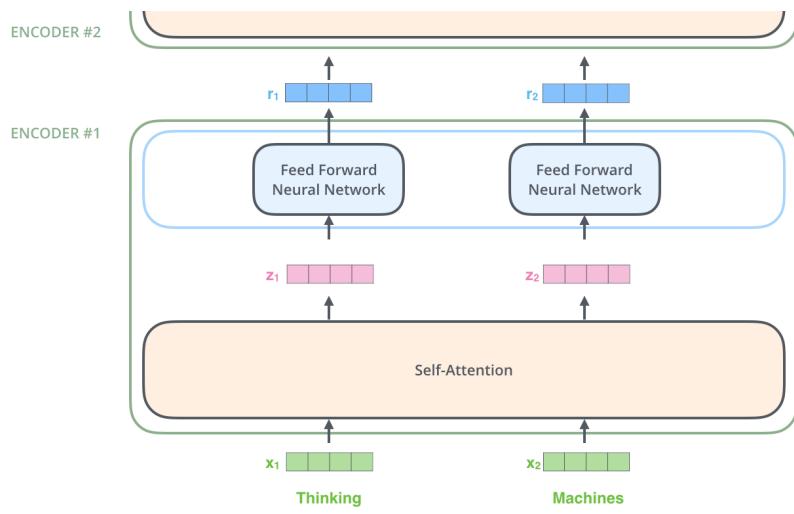


FIGURE 2.36 – La sortie du premier encodeur

Le premier encodeur reçoit sur la couche d'auto-attention une séquence de mots qui sera incorporée en un vecteur, puis chaque vecteur sera envoyé dans un réseau de neurones feedforward où chacun le traverse séparément, et enfin celui-ci envoie la sortie vers l'encodeur supérieur.

2.4.4 Le calcul de L'attention personnelle

Les étapes qui suivent expliqueront comment se fait le calcul de l'auto-attention personnel.

Première étape

La première étape consiste en la création de trois vecteurs à partir de chacun des vecteurs d'entrée de l'encodeur (dans ce cas, l'incorporation de chaque mot). Donc, pour chaque mot, nous créons, un vecteur de requête, de clé, et de valeur. Ces vecteurs sont créés en multipliant l'incorporation par trois matrices créées lors du processus de formation.

Ces nouveaux vecteurs ont une dimension plus petite que le vecteur d'incorporation. Leurs dimensions sont de 64, alors que les vecteurs d'incorporation et d'entrée/sortie de l'encodeur ont une dimension de 512.

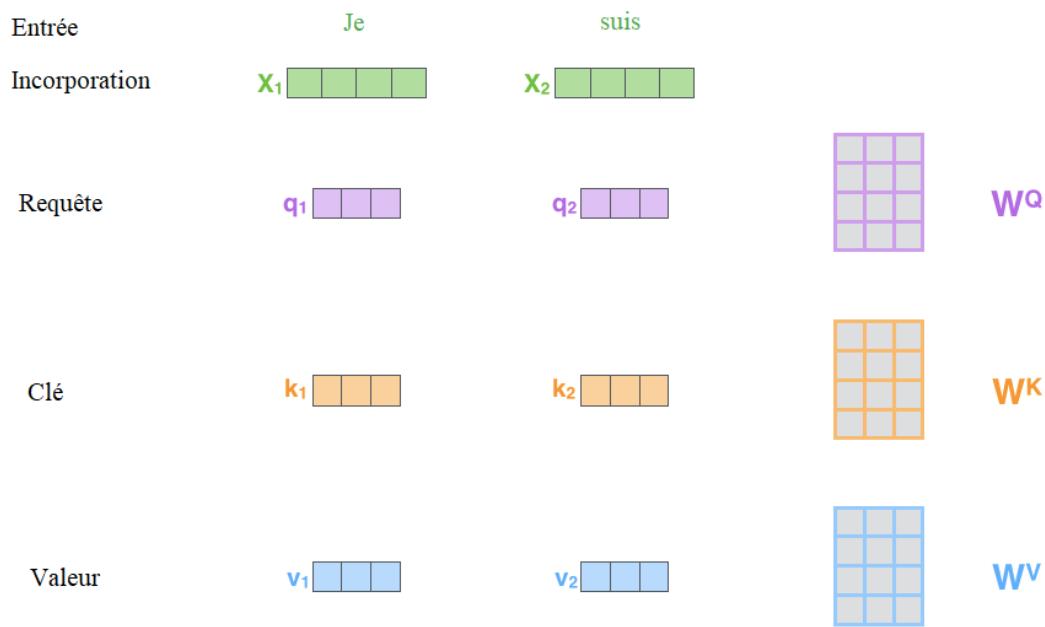


FIGURE 2.37 – Les différents vecteurs nécessaires pour l'auto-attention

On multiplie x_1 par la matrice de pondération W^Q afin d'obtenir le vecteur de requête q_1 associé au mot. A la fin, les vecteurs de requête, de clé, et une projection de valeur de chaque mot de la phrase d'entrée sont créés de chaque mot de la phrase d'entrée.

Deuxième étape

Cette étape consiste à calculer un score de concentration. Soit le mot « je ». Chaque mot de la phrase d'entrée est évalué par rapport à ce mot afin d'obtenir un score. Le score détermine le degré de concentration à accorder aux autres parties de la phrase d'entrée lorsqu'un mot est encodé à une certaine position.

Le score est calculé en prenant le produit scalaire du vecteur de requête avec le vecteur clé du mot en question. Donc pour le mot en position N° 1, le premier score serait le produit scalaire de q_1 et k_1 . Le second score serait le produit scalaire de q_1 et k_2 .

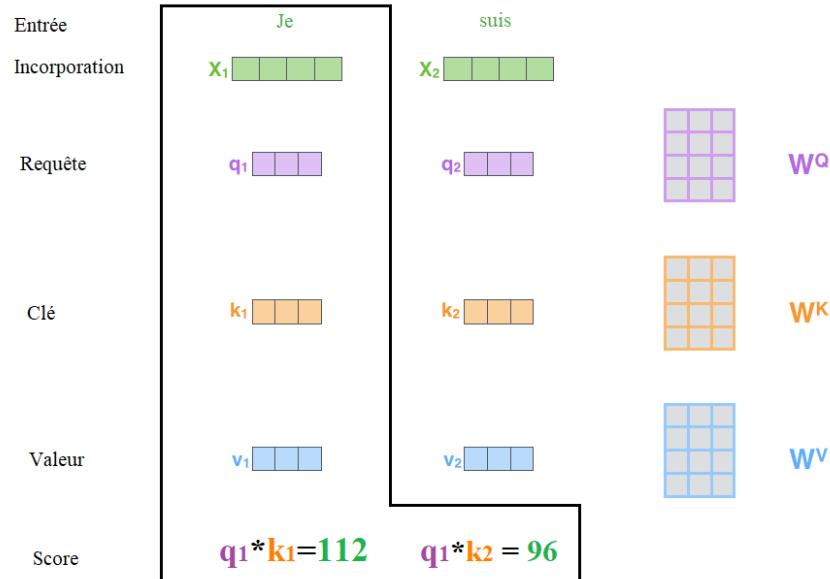


FIGURE 2.38 – Calcul du score pour le mot « je »

Troisième étape

Elle consiste à diviser le score obtenu par la racine carré de la dimension du vecteur clé (dimension 64 par défaut). Cela conduit à des gradients plus stables. Il pourrait y avoir d'autres valeurs possibles, mais il s'agit de la valeur par défaut.

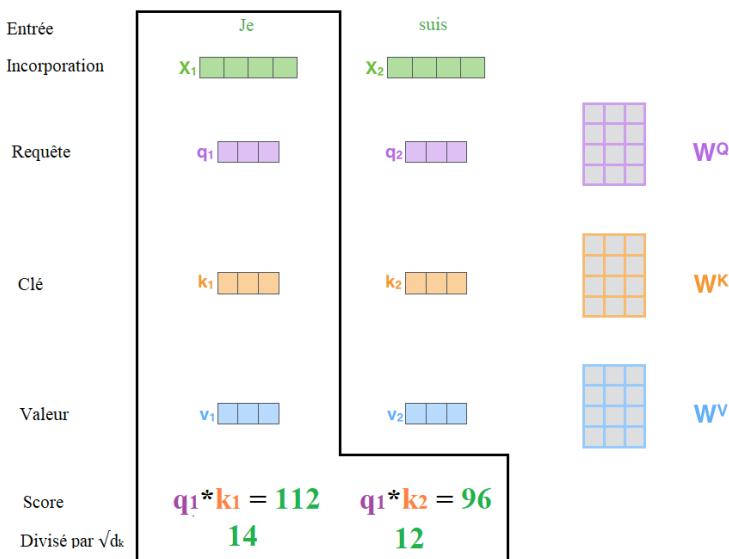


FIGURE 2.39 – Score divisé par la valeur 8

Quatrième étape

Le résultat obtenu après la division du score sera transmis à une fonction Softmax. Celle-ci normalise les scores. Ils seront donc tous positifs et donnent un total de 1.

Valeur	v_1	v_2
Score	$q_1 * k_1 = 112$	$q_1 * k_2 = 96$
Divisé par $\sqrt{d_k}$	14	12
Softmax	0.88	0.12

FIGURE 2.40 – Softmax appliqu 

Ce score d t rmine le degr  d'expression de chaque mot   cette position. Le mot trait  actuellement aura le score Softmax le plus  lev , mais il est parfois utile de s'occuper d'un autre mot qui est pertinent pour le mot actuel.

Cinqui me  tape

On multiplie chaque score Softmax par le vecteur de valeur (pour pr parer le r sum ). Afin de garder les mots pertinents intacts sur lesquels se concentrer, et d'oublier les mots dont on n'a plus besoin ou non pertinents, pour cela en multipliant par des nombres minuscules comme 0.0001.

Valeur	v_1	v_2
Score	$q_1 * k_1 = 112$	$q_1 * k_2 = 96$
Div� par $\sqrt{d_k}$	14	12
Softmax	0.88	0.12
Softmax * Valeur	v_1	v_2

FIGURE 2.41 – Multiplication du score Softmax par le vecteur de valeur

Sixième étape

Cette étape consiste à résumer les vecteurs de valeur pondérés. Cela produit la sortie de la couche d'auto-intention à cette position (pour le premier mot).

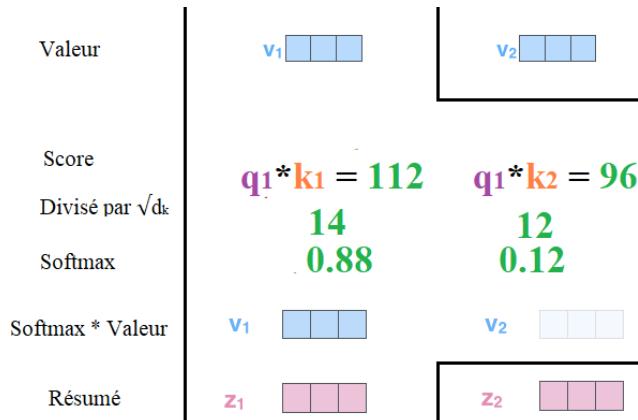


FIGURE 2.42 – Résumé des vecteurs de valeurs pondérées

Ceci finalise le calcul de l'attention personnelle. Le vecteur résultant sera envoyé au réseau de neurones feedforward. Ce calcul est effectué sous forme de matrice pour un traitement plus rapide.

2.4.5 Calcul matriciel de l'attention personnelle au niveau du mot

En premier lieu, le calcul des matrices de requêtes, clés et de valeurs est réalisé. Pour ce faire, les matrices sont encapsulées dans une matrice X et une multiplication se fait par les matrices de poids déjà formées (W_Q , W_K , W_V).

$$\begin{array}{ccc}
 \mathbf{X} & \times & \mathbf{W^Q} \\
 \begin{matrix} \text{green} \\ \text{grid} \end{matrix} & \times & \begin{matrix} \text{purple} \\ \text{grid} \end{matrix} = \begin{matrix} \text{purple} \\ \text{grid} \end{matrix} \\
 \\
 \mathbf{X} & \times & \mathbf{W^K} \\
 \begin{matrix} \text{green} \\ \text{grid} \end{matrix} & \times & \begin{matrix} \text{orange} \\ \text{grid} \end{matrix} = \begin{matrix} \text{orange} \\ \text{grid} \end{matrix} \\
 \\
 \mathbf{X} & \times & \mathbf{W^V} \\
 \begin{matrix} \text{green} \\ \text{grid} \end{matrix} & \times & \begin{matrix} \text{blue} \\ \text{grid} \end{matrix} = \begin{matrix} \text{blue} \\ \text{grid} \end{matrix}
 \end{array}$$

FIGURE 2.43 – Calcul des matrices de requêtes, clés et de valeurs

Chaque ligne de la matrice X correspond à un mot de la phrase en entrée. La taille du vecteur d'incorporation est par défaut 512 (4 cases dans la figure 2.43) et des vecteurs Q/K/V de taille de 64 (3 cases dans la figure 2.43).

Donc, le calcul de la fonction d'attention est fait sur un ensemble de requêtes simultanément, regroupées dans une matrice Q. Les clés et les valeurs sont également regroupées dans les matrices K et V. Celle-ci est calculée comme suit (Référence : AI, 12 Juin 2017) :

$$\text{Attention}(Q, V, K) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (2.3)$$

Où d est le nombre de colonnes de K.

La formule d'attention détaillée (Référence : BRIJESH, 09 Decembre 2018).

$$\text{Attention}(q, V, K) = \sum_i \frac{e^{q*K_i}}{\sum_j e^{q*k_j}} V_i \quad (2.4)$$

D'après le papier de recherche, les produits scalaires deviennent de grande ampleur avec des valeurs importantes de d_k , ceci a pour effet d'obtenir des valeurs de gradient très petit à cause des résultats obtenus en appliquant softmax. Pour contrer cet effet, le résultat des produits scalaires est multiplié par $\frac{1}{\sqrt{d_k}}$. Ceci est choisi en raison de sa rapidité et de son encombrement réduit, car il utilise un code de multiplication de matrice optimisé.

La fonction de compatibilité est considérée en termes de deux variantes, il existe deux méthodes de calcul :

- L'Attention produit scalaire qui a été utilisé dans le papier de recherche à l'exception du facteur scalaire $\frac{1}{\sqrt{d_k}}$.
- L'attention additive.

Attention additive

L'attention additive calcule la fonction de compatibilité en utilisant un réseau à feed-forward avec une seule couche cachée. Elle surpassé l'attention du produit ponctuel sans mise à l'échelle pour des valeurs plus grandes de d_k .

Bien que leur complexité théorique soit similaire, l'attention des produits ponctuels est en pratique beaucoup plus rapide et plus économique en espace, car elle peut être mise en œuvre à l'aide d'un code de multiplication de matrice hautement optimisé.

Remarque

Les produits scalaires peuvent devenir grands. Supposons que les composantes de q et k sont des variables aléatoires indépendantes avec une moyenne de 0 et une variance de 1. Alors leurs produits scalaires, $q * k = \sum_{i=1}^{d_k} q_i k_i$ (Référence : AI, 12 Juin 2017).

2.4.6 L'attention multiple

Au lieu d'exécuter une seule fonction d'attention avec des clés, des valeurs et des requêtes, la projection linéaire des requêtes, des clés et des valeurs h fois avec différentes projections linéaires apprises sur les dimensions d_q , d_k et d_v a été plus avantageuse, respectivement, sur chacune de ces versions projetées de requêtes, de clés et de valeurs, ensuite la fonction attention en parallèle est exécutée, ce qui donne des valeurs de sortie d_v . Celles-ci sont concaténées et à nouveau projetées, ce qui donne les valeurs finales.

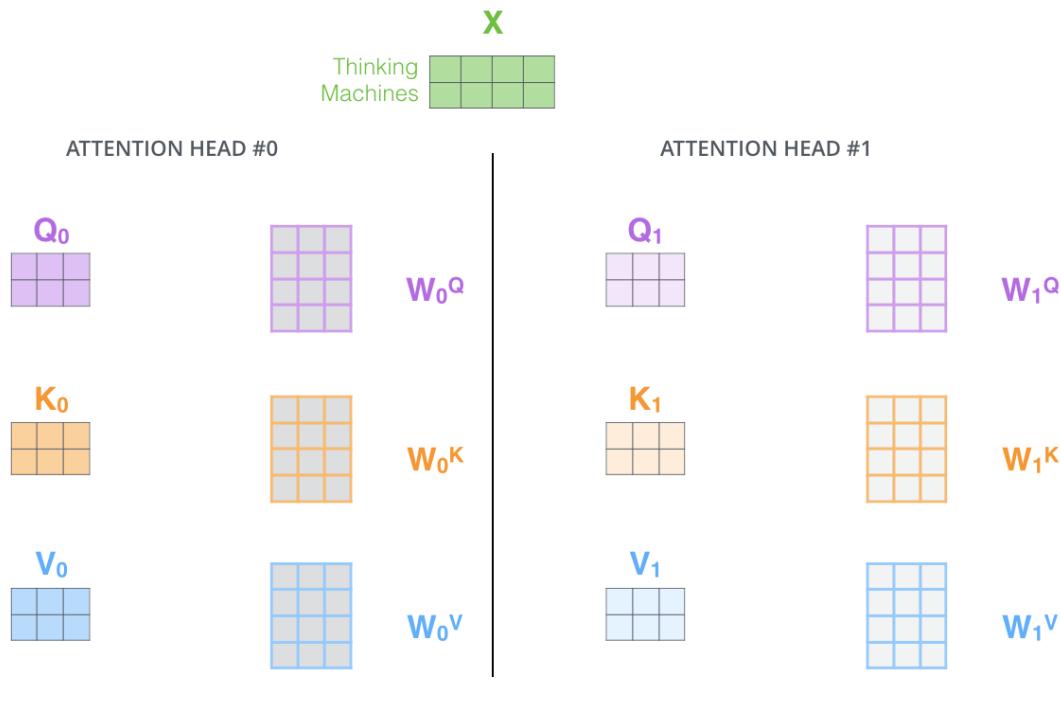


FIGURE 2.44 – L'attention multiple

Avec une attention multiple, les matrices Q/K/V seront séparées pour chaque, ce qui donne des matrices différentes. Donc huit matrices Z différentes seront obtenues. La couche de feed-forward ne prévoit pas huit matrices, mais une seule matrice (un vecteur pour chaque mot). Pour cela :

1. Les huit éléments doivent être condensés en une seule matrice en les concaténant.

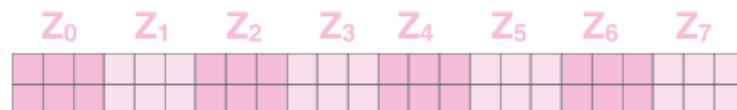


FIGURE 2.45 – Concaténation des têtes d'attention

2. La multiplication est faite avec le résultat obtenu par une matrice de poids W^O qui a été formée conjointement avec le modèle.

3. Le résultat serait la matrice Z qui capture les informations de toutes les têtes d'attention, qui sera envoyée au réseau de neurones.

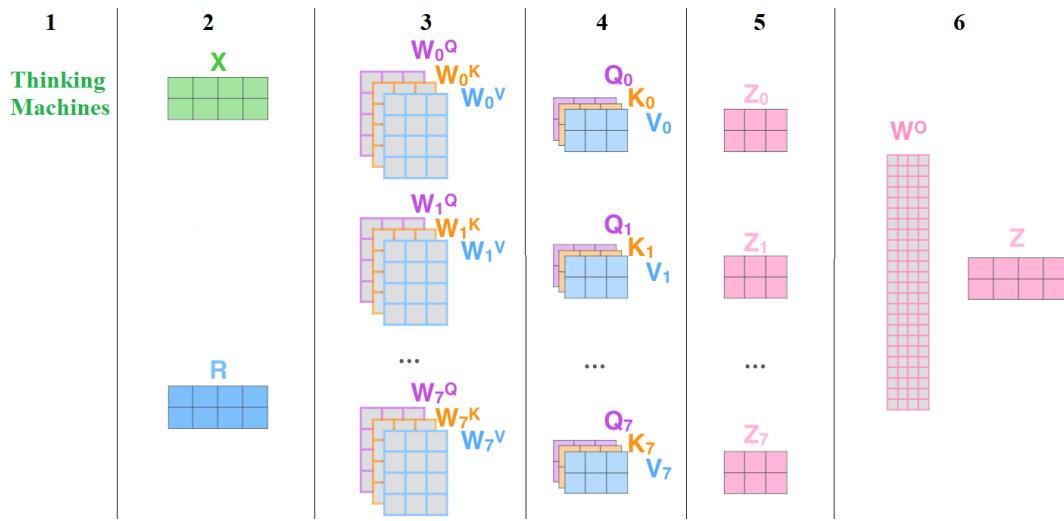


FIGURE 2.46 – Les étapes de l'attention multiple

Les étapes de l'attention multiple :

1. Le mot en entrée.
2. Nous incorporons le mot. Dans tous les encodeurs autres que le premier, il n'y a pas besoin d'intégration, le calcul se fait directement avec la sortie de l'encodeur précédent qui est représenté par la matrice R dans la figure 2.46.
3. On divise en 8 têtes, puis on multiplie X ou R avec des matrices de poids.
4. On calcule l'attention en utilisant les résultats Q/K/V.
5. On concatène les matrices résultantes Z.
6. On multiplie par la matrice de poids W^O pour produire la sortie de la couche.

Formule de l'attention multiple (Référence : AI, 12 Juin 2017) :

$$\begin{aligned} MHSA(H) &= \text{Concat}(\text{head}_1, \dots, \text{head}_k)W^O \\ \text{head}_i &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \end{aligned} \quad (2.5)$$

Ou MHSA = MultiHeadSelfAttention, et avec les paramètres de matrices suivantes :

$W_i^Q, W_i^K \in R^{d_{model} \times d_k}, W_i^V \in R^{d_{model} \times d_v}, W^V \in R^{hd_v \times d_{model}}$ avec h = nombre d'entête.

Par défaut sur le papier de recherche $h = 8$ couches d'attention parelle. $d_k = d_v = d_{model}/h = 64$. En raison de la dimension réduite de chaque tête, le coût total de calcul est similaire à celle de l'attention d'une seule tête avec une dimensionnalité complète.

Une attention scalaire en points est appliquée en parallèle sur chaque couche (différentes projections linéaires de k, q, v) résulte une sortie en dimension d_v .

Le transformateur utilise l'attention multi-têtes (fonctions d'attention parallèle d_{model}/h) à la place de la fonction d'attention unique ($d_{model-dimension}$) (c'est-à-dire q, k, v tous $d_{model-dimension}$).

2.4.7 Les résidus

L'architecture de l'encodeur est que chaque composant encodeur (self-attention, ffnn) est entouré d'une connexion résiduelle et est suivie d'une étape de normalisation de couche (Pour plus d'informations : 2.9 Normalisation).

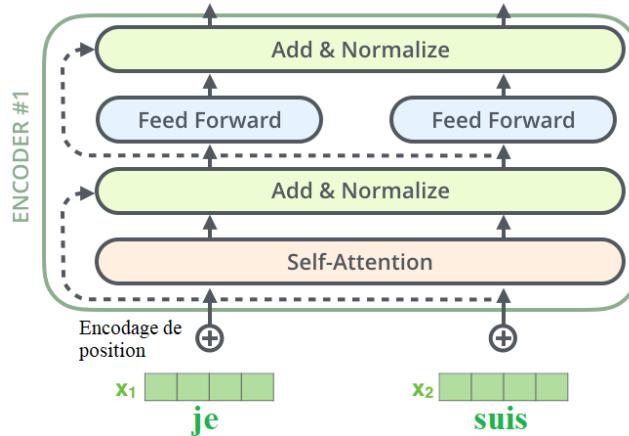


FIGURE 2.47 – Connexion résiduelle

La sortie de chaque sous couche est : LayerNorm ($x + \text{Sublayer}(x)$) ou Sublayer (x) est la fonction implémentée par la sous-couche elle-même. Pour faciliter ces connexions résiduelles, toutes les sous-couches du modèle, ainsi que les couches d'intégration, produisent des sorties de dimension $d_{model} = 512$.

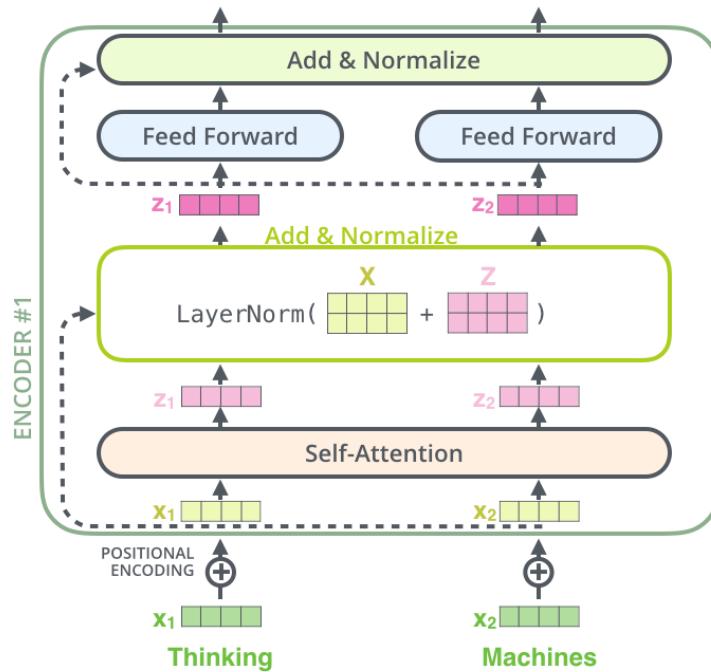


FIGURE 2.48 – Visualisation des vecteurs et d'opération couche-norme associées à l'attention propre

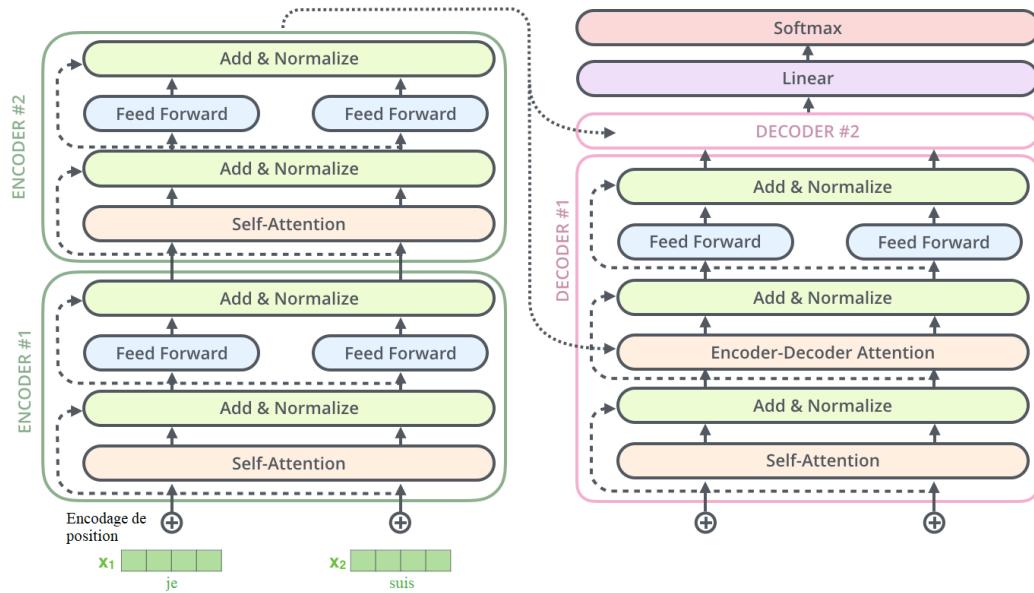


FIGURE 2.49 – Un Transformer composé de deux encodeurs et décodeurs avec la normalisation de couches

2.4.8 Le décodeur

La sortie du traitement de l'encodeur de la couche supérieure est transformée en un ensemble de vecteurs d'attention K et V. Ceux-ci doivent être utilisés par chaque composant du décodeur, pour que celui-ci se concentre sur les emplacements appropriés de la séquence d'entrée.

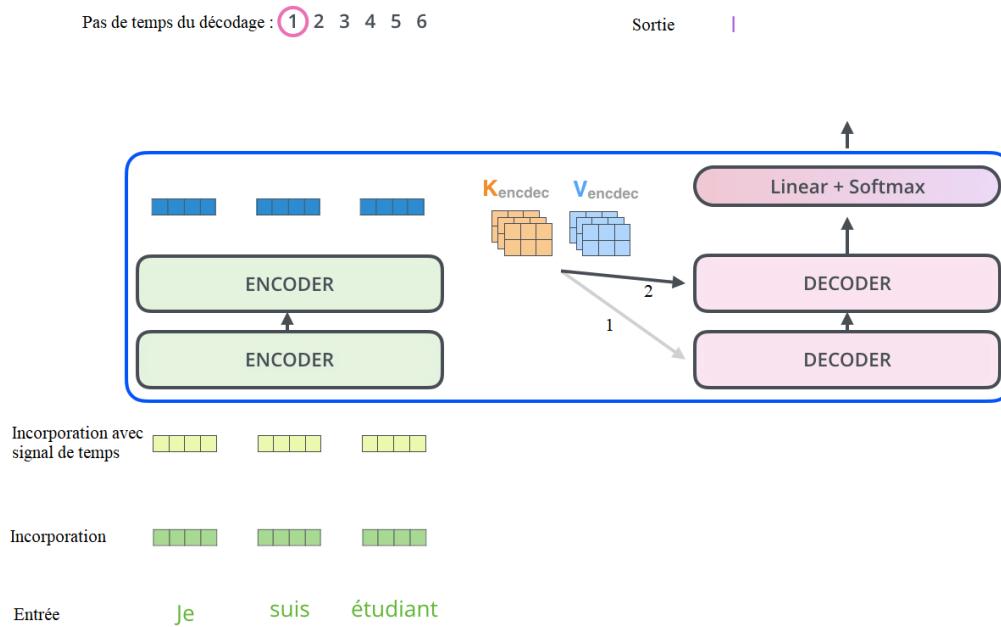


FIGURE 2.50 – Le décodeur lors du premier pas

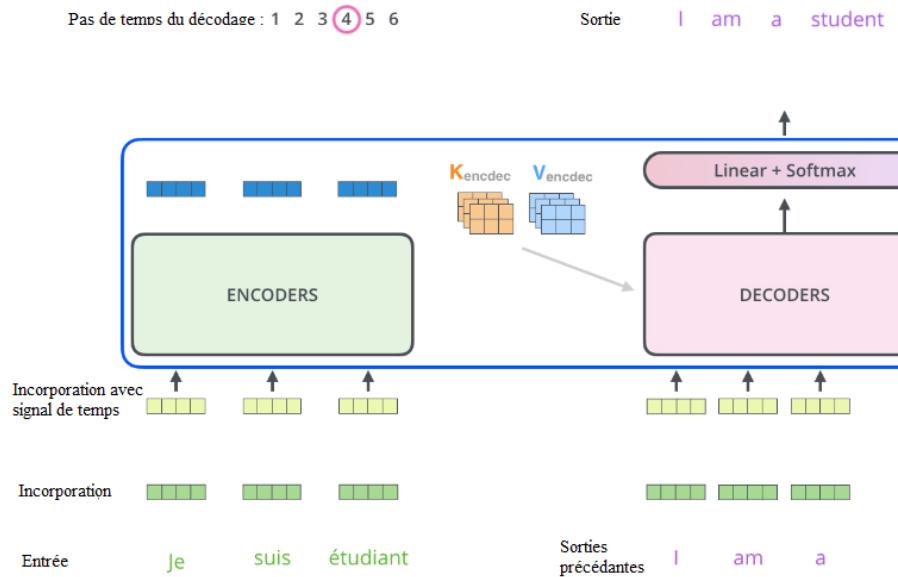


FIGURE 2.51 – Le décodeur lors du quatrième pas

K et V sont produits en projetant la sortie du codeur deux fois (séparément) avec des matrices de projection respectives. Les valeurs K et V seront différentes, mais uniquement parce que la matrice de projection produisant K est différente de la matrice de projection produisant V . Toutefois, les entrées des matrices de projection sont les mêmes pour toutes les couches du décodeur. Donc chaque décodeur utilise les mêmes K et V .

Dans le codeur, les entrées des matrices de projection d'une couche sont les mêmes (K et V étant toutes deux dérivées de la sortie de la couche située au-dessous d'elles), mais entre les couches, elles sont bien sûr différentes puisque la sortie des différentes couches est différente. Les poids ne sont pas partagés dans ou entre les couches d'encodeur/décodeur.

Une fois la phase d'encodage terminée, la phase de décodage est entamée. Chaque étape de la phase de décodage génère un élément de la séquence de sortie.

Les étapes se répètent jusqu'à que le processus arrive à un symbole spécial indiquant que le décodeur du transformateur a terminé sa sortie. La sortie de chaque étape est transmise au décodeur inférieur lors de la prochaine étape, et les décodeurs affichent leurs résultats de décodage de la même manière que les encodeurs. Des encodages de position sont intégrés à ces entrées de décodeur pour indiquer la position de chaque mot, comme pour l'encodeur.

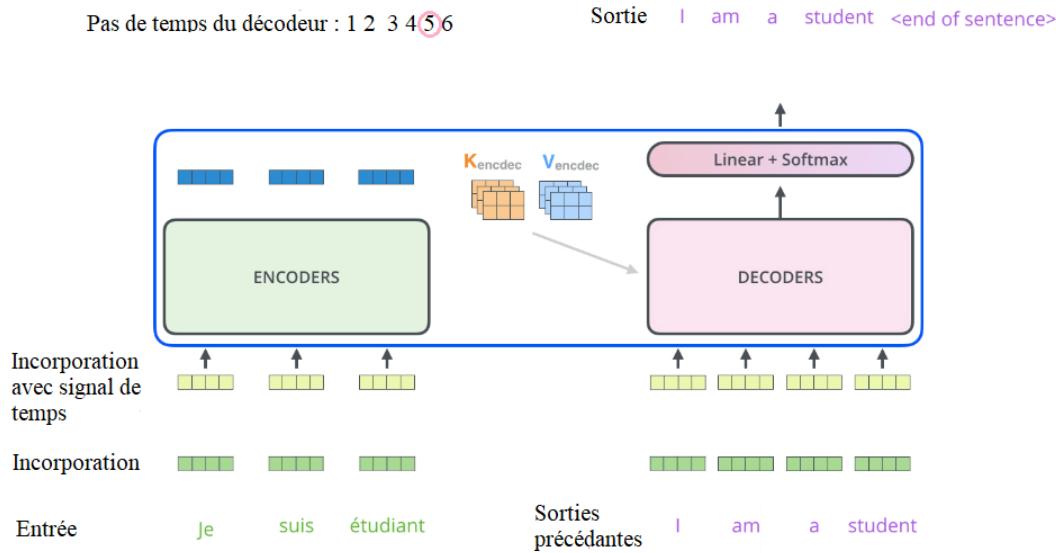


FIGURE 2.52 – Décodage complet

La couche « Encoder-Decoder Attention » fonctionne comme une auto-attention multi-têtes, à ceci près qu'elle crée sa matrice de requêtes à partir de la couche située en dessous et extrait la matrice de clés et de valeurs de la sortie de la pile de codeurs.

2.4.9 La couche finale linéaire et Softmax

La pile du décodeur génère un vecteur de float, et c'est ensuite au travail de la dernière couche linéaire, qui est suivie par une couche softmax qui permet de transformer le vecteur en mot.

La couche linéaire est un simple réseau de neurone (ffnn) qui projette le vecteur produit par la pile du décodeur dans un vecteur plus grand, appelé vecteur logits. Logits correspondent aux notes finales non normalisées du modèle.

Supposant qu'un modèle connaît 10.000 mots anglais uniques (vocabulaire de sortie). Cela fera un vecteur logits avec une taille en largeur de 10.000 cellules, chaque cellule correspond au score d'un mot unique. C'est ainsi que se fait l'interprétation de la sortie du modèle suivie par la couche linéaire.

La couche softmax transforme (normalise) ensuite ces scores en probabilités (tous positifs, tous totalisent 1,0). La cellule avec la probabilité la plus élevée est choisie et le mot qui lui est associé est produit comme sortie pour ce pas de temps. (Pour plus d'informations consultez la section 2.5 La fonction de perte et l'optimiseur)

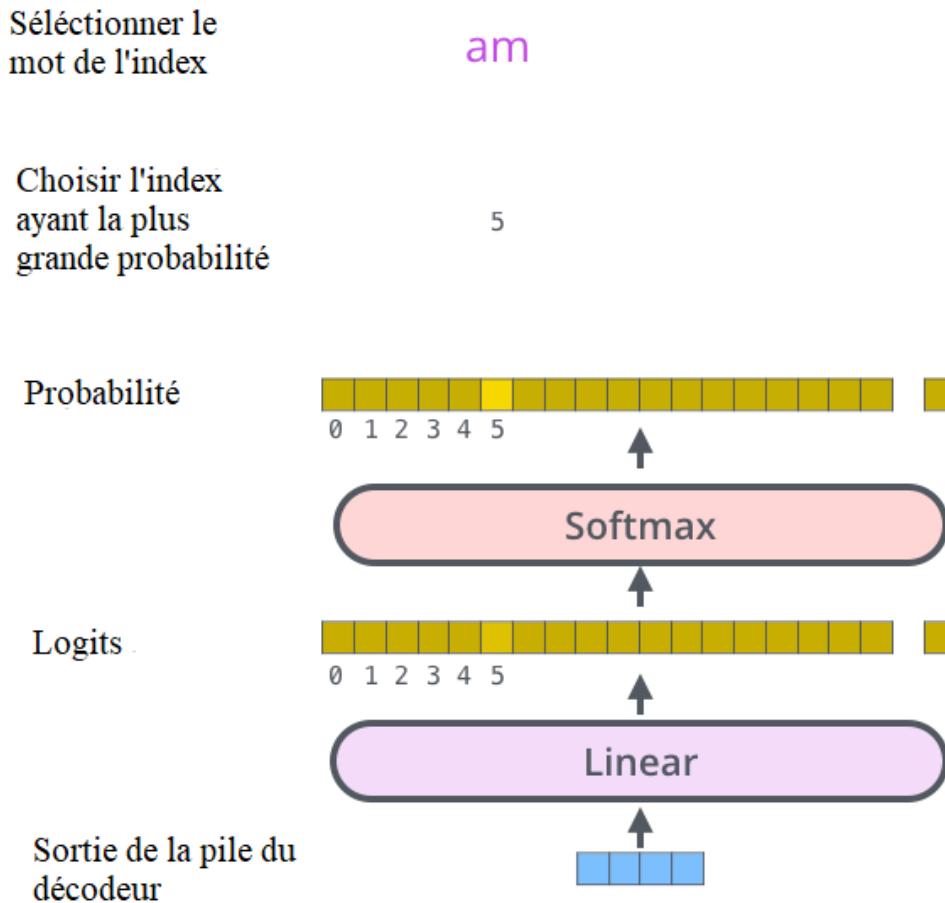


FIGURE 2.53 – Application de la couche linéaire et Softmax

2.4.10 Résumé de la formation

Vu que l'apprentissage utilisé est un apprentissage supervisé, on peut donc comparer les résultats de la machine avec les résultats réels. Pour visualiser cela, supposons que le vocabulaire de sortie ne contienne que six mots (“a”, “am”, “i”, “merci”, “étudiant” et “<eos>” (abréviation de “end of sentence”)).

Output Vocabulary						
WORD	a	am	i	thanks	student	<eos>
INDEX	0	1	2	3	4	5

FIGURE 2.54 – Exemple de vocabulaire de sortie

Le vocabulaire de sortie sera créé lors de la phase de prétraitement, avant même que la formation ne s'entame. Ceci fait, un vecteur de même largeur est utilisé pour indiquer chaque mot du vocabulaire. Cela s'appelle codage one-hot. Par exemple, le mot « am » est indiqué en utilisant le vecteur suivant (Pour plus d'informations 2.8.1 One-Hot encodage)) :

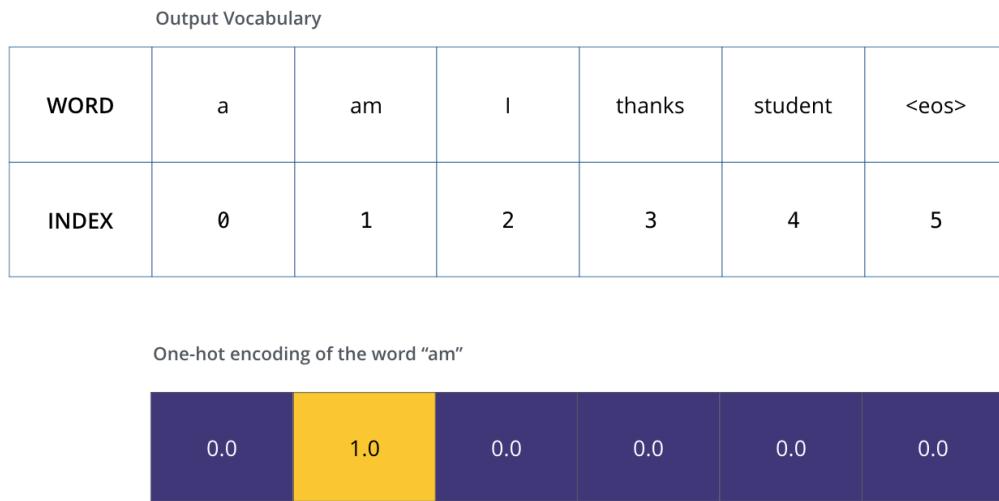


FIGURE 2.55 – Exemple d'encodage à chaud (one-hot)

2.4.11 La fonction de perte

Soit une formation de modèle. Supposant que c'est la première étape dans la phase de formation et que la formation se fait sur un exemple simple : traduire « merci » en « thanks ». Mais comme ce modèle n'est pas encore formé, la sortie de distribution de probabilité ne sera pas satisfaisante.

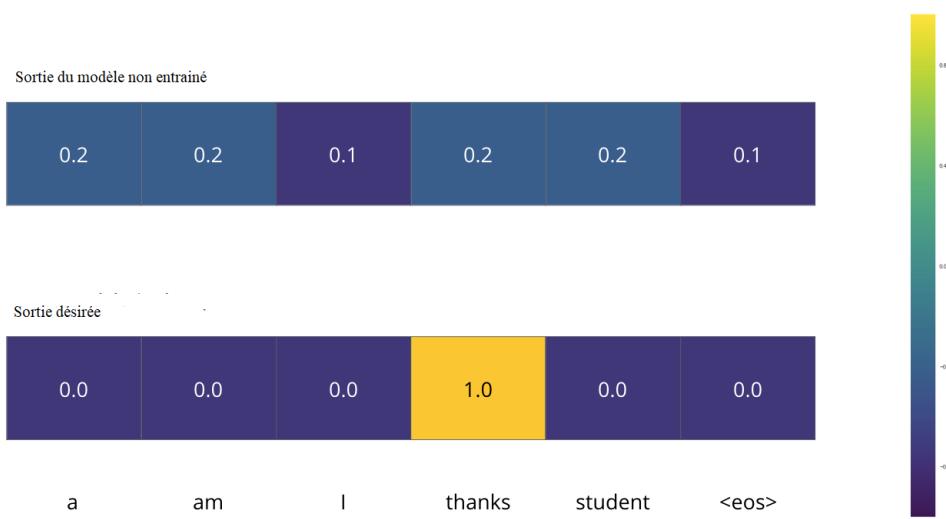


FIGURE 2.56 – Sortie d'un modèle non entraîné

Le modèle (non entraîné) génère une distribution de probabilité avec des valeurs arbitraires pour chaque cellule / mot car les poids du modèle sont tous initialisés de manière aléatoire. Nous pouvons le comparer à la sortie réelle, puis ajuster tous les poids du modèle en utilisant la rétropropagation pour rendre la sortie plus proche de la sortie désirée. La comparaison entre les deux distributions se fait en les soustrayant l'un à l'autre, en utilisant l'entropie croisée ou la divergence de Kullback-Leibler, mais c'est L'entropie croisée qui est la plus utilisée

Exemple plus compliquer

Soit la traduction de la phrase « je suis étudiant » vers « i'm a student », plusieurs vecteurs de probabilité seront générés.

- Chaque distribution de probabilité est représentée par un vecteur de largeur (6 dans l'exemple, mais dans la réalité ça peut aller jusqu'à 3000, 10 000, ou encore 32000).
- La première distribution de probabilité a la probabilité la plus élevée à la cellule associée au mot « i ».
- La deuxième distribution de probabilité a la probabilité la plus élevée à la cellule associée au mot « am ».
- Et ainsi de suite, jusqu'à ce que la cinquième distribution en sortie indique le symbole <end of sentence>symbole, auquel est également associée une cellule du vocabulaire des 10 000 éléments.

Sortie du modèle cible

Vocabulaire de sortie : a am I thanks student <eos>

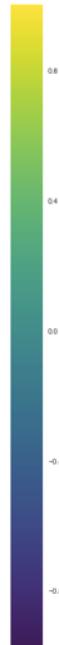


FIGURE 2.57 – Encodage one-hot avec plusieurs mots

Après avoir suffisamment formé le modèle, Les distributions de probabilité générées peuvent ressembler à ceci :

Sortie du modèle formé

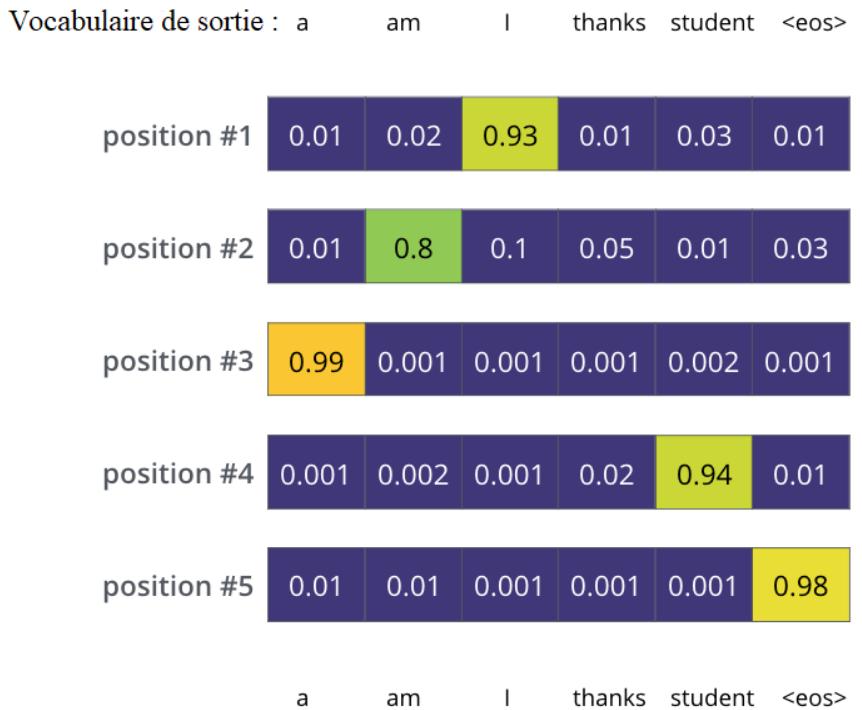


FIGURE 2.58 – Distribution de probabilités avec plusieurs mots

Chaque position a un peu de probabilité même s'il est peu probable qu'elle soit le résultat de ce pas de temps, c'est une propriété très utile de softmax qui facilite le processus de formation.

2.4.12 Type de prédiction

Décodage glouton

Etant donné que le modèle produit les sorties une par une, il sélectionne le mot avec la probabilité la plus élevée parmi cette distribution de probabilité. Cette méthode est appelée décodage glouton. Cette technique peut amener à des traductions sous-optimales.

Recherche de faisceau (beam search)

C'est un algorithme de recherche heuristique qui explore un graphe en développant le nœud le plus prometteur dans un ensemble limité (Référence : MISHRA, 20 Février 2018).

Deux mots les plus fréquents, par exemple « I » et « a » seront conservés, puis à l'étape suivante, le modèle sera exécuté deux fois. Une fois que la première position de sortie était le mot « I » et une autre fois, en supposant que la première position de sortie était le mot "me", et quelle que soit la version générant le moins d'erreurs compte tenu des positions $N^o 1$ et $N^o 2$ conservées. Cela se répète pour les positions $N^o 2$ et $N^o 3$, etc. Dans l'exemple, beam_size est égal à deux (car les résultats sont comparés après avoir calculé les faisceaux pour les positions $N^o 1$ et $N^o 2$), et top_beams est également deux (puisque deux mots ont été gardés). Ce sont deux hyperparamètres qui peuvent être modifiés.

- Si beam_size = 1, la technique est identique à la recherche de glouton.
- Plus grand le beam_size est grand, plus il y'a de bonnes chances d'obtenir de meilleures séquences de sortie, mais consommerait plus de ressources et de puissance de calcul.
- Plus petit le beam_size est petit, plus le résultat ne sera pas si bon, mais ça sera beaucoup plus rapide et efficace en mémoire.

La valeur de la largeur du faisceau à des fins de production est généralement maintenue entre 10 et 100, et à des fins de recherche, elle est généralement comprise entre 1 000 et 3 000. Plus la largeur du faisceau est grande, plus la possibilité de trouver une phrase probable est grande, mais cela entraîne des frais de calcul et l'exigence de mémoire considérablement élevée.

2.5 La fonction de perte et l'optimiseur

Les machines apprennent au moyen d'une fonction de perte. C'est une méthode permettant d'évaluer dans quelle mesure un algorithme spécifique modélise les données fournies. Si les prévisions s'écartent trop des résultats réels, la fonction de perte en couvrirait un très grand nombre. Progressivement, à l'aide d'une fonction d'optimisation, la fonction de perte apprend à réduire l'erreur de prévision.

La fonction de coût ou de perte joue un rôle important dans la mesure où elle doit représenter fidèlement tous les aspects du modèle en un nombre unique, de telle sorte que les améliorations de ce nombre sont le signe d'un meilleur modèle. Dans ce projet, l'entropie croisée a été utilisée

2.5.1 L'entropie croisée

L'entropie croisée est une fonction de perte qui indique la distance entre ce que le modèle estime que la distribution de la production devrait être et la distribution originale.

La mesure de l'entropie croisée est une alternative largement utilisée de l'erreur carrée. Elle est utilisée lorsque les activations de nœud peuvent être comprises comme représentant la probabilité que chaque hypothèse soit vraie, c'est-à-dire lorsque la sortie est une distribution de probabilité. Ainsi, elle est utilisée comme fonction de perte dans les réseaux de neurones qui ont des activations softmax dans la couche de sortie.

Après cela l'application de softmax, un codage à chaud est appliqué transforme les sorties en forme binaire. C'est pourquoi, softmax et un codage à chaud seraient appliqués respectivement à la couche de sortie des réseaux de neurones. Ici, la fonction d'entropie croisée établit une corrélation entre les probabilités et une étiquette codée à chaud.

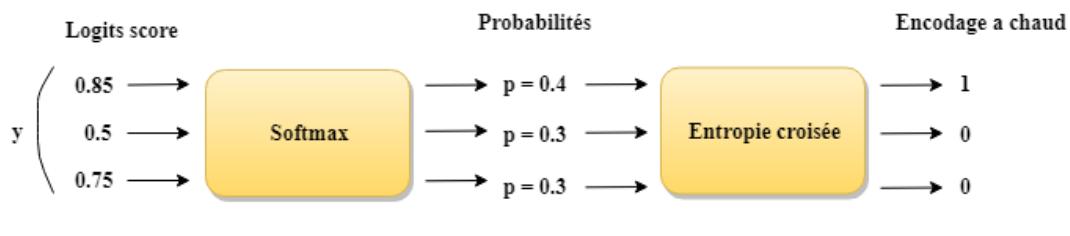


FIGURE 2.59 – L'entropie croisée

La perte d'entropie croisée mesure la performance d'un modèle dont la sortie est une valeur de probabilité comprise entre 0 et 1. La perte d'entropie croisée augmente à mesure que la probabilité prédite diverge de l'étiquette réelle. Donc, prédire une probabilité de 0,012 lorsque l'étiquette d'observation réelle est 1 serait mauvais et entraînerait une valeur de perte élevée. Un modèle parfait aurait une perte de journal de 0.

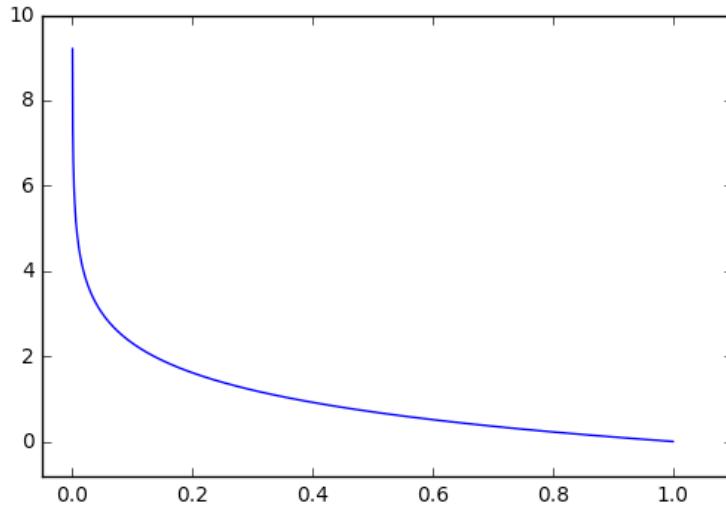


FIGURE 2.60 – Plage de valeur de l’entropie croisée (Référence : MACHINE LEARNING CHEATSHEET, 2017)

Formule mathématique de l’entropie croisée :

$$H(p, q) = - \sum_x p(x) \log q(x) \quad (2.6)$$

$H(p, q)$ signifie que l’on calcule l’attente en utilisant p et la taille de codage en utilisant q . En tant que tel, $H(p, q)$ et $H(q, p)$ ne sont pas nécessairement identiques, sauf lorsque $q = p$, auquel cas $H(p, q) = H(p, p) = H(p)$ et elle devient l’entropie (une autre fonction de perte) elle-même.

Ce point est subtil mais essentiel. Pour l’attente, nous devrions utiliser la probabilité réelle p . Pour la taille d’encodage, nous devrions utiliser q . Puisque l’entropie est la taille moyenne minimale théorique, l’entropie croisée est supérieure ou égale à l’entropie mais pas inférieure à celle-ci. En d’autres termes, si l’estimation est parfaite, $q = p$ et donc $H(p, q) = H(p)$. Sinon, $H(p, q) > H(p)$. (Référence : SHIBUYA, 28 Octobre 2018).

2.5.2 Logits

En apprentissage en profondeur, le terme couche logits est couramment utilisé pour désigner la dernière couche de neurones du réseau de neurones qui produit des valeurs de prédiction brutes sous forme de nombres réels. Ce sont les scores bruts générés par la dernière couche d’un réseau de neurones. Avant l’activation.

Etant x , une probabilité, la valeur de la fonction logit se dirige vers l’infini à mesure que x approche de 1 et vers l’infini négatif à l’approche de 0. (Référence :).

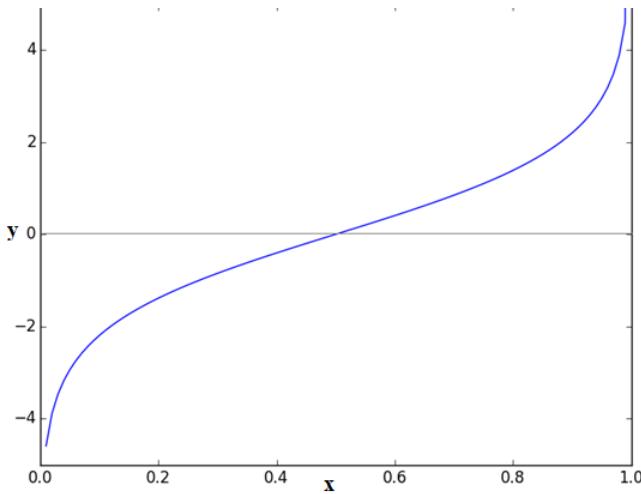


FIGURE 2.61 – Logit

Ce qui suit est la formule du logit :

$$f(x) = \log \frac{x}{1-x} \quad (2.7)$$

La fonction logit est utile en analytique car elle mappe les probabilités (valeurs comprises dans la plage $[0, 1]$) à la plage complète des nombres réels. Il est utile de les transformer en quantités à valeurs réelles

2.5.3 Softmax

Softmax est une fonction d'activation qui transforme les logits en chiffres de probabilités, plus précisément en une distribution de probabilité d'une liste de résultats potentiels. La sortie Softmax est grande si le score (logit) est grand. Sa sortie est petite si le score est petit. Softmax est exponentiel et élargit les différences, rapprochez un résultat de 1 et un autre de 0. Il transforme les scores en logits en probabilités égales à un. La somme des résultats de Softmax à 1 constitue une excellente analyse de probabilité. (Référence : “Classification and Loss Evaluation - Softmax and Cross Entropy Loss”)

$$\sigma(a)_j = \frac{e^{a_j}}{\sum_{k=1}^N e^{a_k}} \quad \text{avec } j \in \{1, \dots, N\} \quad (2.8)$$

2.5.4 Dérivation

Softmax est fondamentalement une fonction vectorielle. Il prend un vecteur en entrée et produit un vecteur en sortie. En d'autres termes, il a plusieurs entrées et plusieurs sorties. La perte d'entropie croisée avec la fonction Softmax est largement utilisée comme couche de sortie. Nous avons besoin de connaître la dérivée de la fonction de perte pour effectuer une rétrodiffusion(backpropagation).

2.5.5 L'optimiseur Adam

Adam (Référence : BROWNLEE, 03 Juillet 2017) est un algorithme populaire dans le domaine de l'apprentissage en profondeur car il permet d'obtenir rapidement de bons résultats. Il est l'un des algorithmes d'optimisation de descente de gradient les plus populaires.

C'est une méthode d'apprentissage adaptative, ce qui signifie qu'il calcule les taux d'apprentissage individuels pour différents paramètres. Son nom est dérivé de l'estimation adaptative du moment, et la raison pour laquelle il est appelé ainsi est qu'Adam utilise des estimations des premier et second moment de gradient pour adapter le taux d'apprentissage pour chaque poids du réseau neuronal. Il est spécialement conçu pour la formation de réseaux de neurones profonds.

Les auteurs décrivent Adam comme combinant les avantages de deux autres extensions de descente de gradient stochastique. Plus précisément :

- Algorithme de gradient adaptatif (AdaGrad) qui maintient un taux d'apprentissage par paramètre qui améliore les performances sur les problèmes de gradients en voie de disparition (par exemple, les problèmes de langage naturel et de vision par ordinateur).
- Propagation quadratique moyenne (RMSProp) qui maintient également des vitesses d'apprentissage.

Avantage

- Mémoire requise relativement faible (bien qu'elle soit supérieure à la descente et à la descente progressive).
- Cela fonctionne généralement bien, même avec un petit réglage d'hyperparamètres (sauf alpha).

Paramètre d'adam

- α : Aussi appelé taux d'apprentissage. La proportion de pondérations mises à jour. Des valeurs plus grandes (par exemple 0,3) entraînent un apprentissage initial plus rapide avant la mise à jour du taux. Des valeurs plus faibles (par exemple 1.0E-5) ralentissent l'apprentissage tout au long de la formation. Par default elle est réglée à 0.1
- β_1 : Le taux de décroissance exponentiel pour le premier moment estimé (par exemple 0,9 par défaut).
- β_2 : Le taux de décroissance exponentiel pour les estimations du deuxième moment. Cette valeur doit être proche de 1.0 pour les problèmes de faible gradient (par exemple, problèmes de TLN et de vision par ordinateur). Sa valeur par défaut est 0.999.
- ϵ : C'est un très petit nombre pour éviter toute division par zéro dans la mise en œuvre. Par défaut sa valeur est : 108.

2.6 Le score BLEU

2.6.1 Vérification de modèle

Il serait difficile de vérifier les modèles de traduction automatique en utilisant des traducteurs humains, non seulement à cause du temps qu'il faudrait, mais aussi parce que cela n'aurait aucun sens. Il est donc raisonnable de trouver une sorte de mesure permettant d'évaluer automatiquement la qualité des traductions.

Les algorithmes de recherche constituent une partie cruciale de la traduction automatique probabiliste. Leur performance affecte directement la qualité de la traduction. Sans un décodeur fiable et efficace, un système de traduction automatique peut manquer la traduction d'une phrase source, même si elle était une phrase du corpus d'entraînement.

Le BLEU

BLEU (bilingual evaluation understudy) est un algorithme d'évaluation de la qualité du texte qui a été traduit par la machine d'une langue naturelle à une autre. La qualité est considérée comme la correspondance entre la production d'une machine et celle d'un humain : « plus une traduction automatique est proche d'une traduction humaine professionnelle, mieux c'est ». Elle reste l'une des métriques automatisées les plus populaires et les moins couteuses.

Les notes sont calculées pour chaque segment traduit, généralement des phrases, en les comparant avec un ensemble de traductions de référence de bonne qualité. La moyenne de ces notes est ensuite calculée sur l'ensemble du corpus pour obtenir une estimation de la qualité globale de la traduction. L'intelligibilité ou l'exactitude grammaticale ne sont pas prises en compte.

L'idée de Bleu est de comparer les phrases (traduction, référence) en se basant sur les séquences de mots (n-gram), un modèle de langage probabiliste souvent utilisé en linguistique informatique. Une traduction est d'autant meilleure qu'elle partage un grand nombre de n-gram avec une ou plusieurs traductions de référence.

La sortie de BLEU est toujours un nombre compris entre 0 et 1, qui indique dans quelle mesure le texte candidat est similaire aux textes de référence, les valeurs plus proches de 1 représentant des textes plus similaires. Peu de traductions humaines atteindront une note de 1, car cela indiquerait que le candidat est identique à l'une des traductions de référence. Pour cette raison, il n'est pas nécessaire d'obtenir un score de 1, car il y a plus de possibilités d'appariement, l'ajout de traductions de référence supplémentaires augmentera le score BLEU.

2.7 Le calcul de BLEU

Les formules suivantes proviennent des références : RAJ, 16 Septembre 2017 et RECHERCHE, Juillet 2002.

$$Count_{clip} = \min(Count, Max_Ref_Count) \quad (2.9)$$

Avec *Max_Ref_Count* qui est le nombre maximum de fois qu'apparaît le mot entre différentes références.

On calcule les différentes précisions.

$$P_n = \frac{\sum_{ngrams \in \hat{y}} Count_{clip}(ngram)}{\sum_{ngrams \in \hat{y}} Count(ngram)} \quad (2.10)$$

Avec \hat{y} comme sortie de la machine

2.7.1 Combinaison du score BLEU

Pour inclure tous les scores de précision n-grammes dans une précision finale, La moyenne géométrique est prise. Ceci est fait car il a été constaté que la précision décroît de façon exponentielle avec n et que, par conséquent, nous aurions besoin d'une moyenne logarithmique pour représenter toutes les valeurs de manière équitable.

$$Precision = \exp\left(\sum_{n=1}^N w_n \log p_n\right), \text{ avec } w_n = 1/n \quad (2.11)$$

2.7.2 Longueur optimale

Bien que le calcul de précision soit relativement simple, le problème du rappel est qu'il peut exister de nombreux textes de référence. Il est donc difficile de calculer la sensibilité du candidat par rapport à une référence générale. Cependant, il est intuitif de penser qu'un texte de candidat plus long est plus susceptible de contenir une fraction plus grande de certaines références qu'un candidat plus court.

Par conséquent, un rappel peut être introduit pour pénaliser simplement la brièveté dans les textes candidats. Ceci est fait en ajoutant un facteur multiplicatif BP avec la précision modifiée de n-gramme comme suit.

$$BP = \begin{cases} 1, & \text{si } r > c \\ \exp(1 - \frac{r}{c}), & \text{sinon} \end{cases}$$

- c : est la longueur totale du corpus de traduction candidat.
- r : est la longueur de référence effective du corpus, c'est-à-dire la longueur moyenne de toutes les références.

Les longueurs sont considérées comme moyennes sur l'ensemble du corpus pour éviter de punir sévèrement les écarts de longueur des phrases courtes. Au fur et à mesure que la longueur candidate diminue, le rapport r / c augmente et le BP diminue de façon exponentielle.

2.7.3 Calcul final du BLEU

$$BLEU = BP * \exp\left(\sum_{n=1}^N w_n \log p_n\right) \quad (2.12)$$

2.8 Représentation vectorielle

Afin de mettre les mots dans l'algorithme d'apprentissage automatique, les données de texte doivent être converties en représentations vectorielles. La plupart des approches peuvent être classées en deux catégories : la méthode du nombre et la méthode de la prévision.



FIGURE 2.62 – Deux techniques utilisées pour la représentation vectorielle

2.8.1 One-Hot encodage

Un encodage à chaud est une représentation des variables catégorielles sous forme de vecteurs binaires. Les données catégoriques sont des variables contenant des valeurs d'étiquette plutôt que des valeurs numériques. Chaque valeur entière de l'encodage à chaud est représentée sous la forme d'un vecteur binaire qui correspond à zéro, à l'exception de l'index de l'entier, marqué d'un 1.

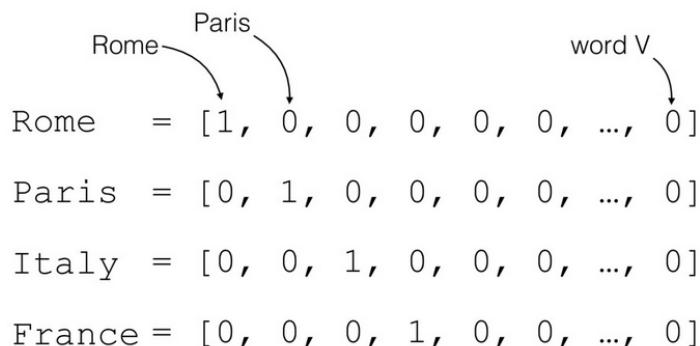


FIGURE 2.63 – One-hot encodage représentation (Référence : SHAFFY, 8 Novembre 2017)

L'importance De l'encodage à chaud

Un encodage à chaud permet à la représentation de données catégorielles d'être plus expressive. De nombreux algorithmes d'apprentissage machine ne peuvent pas travailler directement avec des données catégorielles. Les catégories doivent être converties en nombres. Cela est nécessaire pour les variables d'entrée et de sortie qui sont catégoriques. Elle présentent plusieurs avantages :

- Méthode la plus simple à implémenter
- Non ordonné, le contexte des mots est donc perdu.

- Non ordonné, le contexte des mots est donc perdu.

2.8.2 Word Embeddings

Comment Google Translate peut-il comprendre le texte et le convertir d'une langue à une autre ? Comment faire comprendre à un ordinateur le contexte des mots ? C'est là où le Word Embeddings (l'incorporation de mot) entre en jeu.

Comprendre l'incorporation de mot

La grande idée de l'incorporation de mots est de transformer le texte en chiffres. Cette transformation est nécessaire car de nombreux algorithmes d'apprentissage automatique (y compris les réseaux profonds) exigent que leur entrée soit un vecteur de valeurs continues, ils ne fonctionneront tout simplement pas avec les chaînes de texte brut.

Rome = [0.91, 0.83, 0.17, ..., 0.41]	Rome = [0.91, 0.83, 0.17, ..., 0.41]
Paris = [0.92, 0.82, 0.17, ..., 0.98]	Paris = [0.92, 0.82, 0.17, ..., 0.98]
Italy = [0.32, 0.77, 0.67, ..., 0.42]	Italy = [0.32, 0.77, 0.67, ..., 0.42]
France = [0.33, 0.78, 0.66, ..., 0.97]	France = [0.33, 0.78, 0.66, ..., 0.97]

FIGURE 2.64 – Exemple d'incorporation de mots (Référence : SHAFFY, 8 Novembre 2017)

L'incorporation de mots vise à créer une représentation vectorielle avec un espace dimensionnel beaucoup plus petit que celui du sac de mots. Les vecteurs créés par l'intégration de mots préservent les similitudes. Ils représentent les mots en tant que nombres multidimensionnels continus à virgule flottante où des mots sémantiquement similaires sont mappés sur des points proches de l'espace géométrique.

En plus de pouvoir être traitée par des algorithmes d'apprentissage, cette représentation vectorielle a deux propriétés importantes et avantageuses :

- **Réduction de la dimensionnalité** : Une représentation plus efficace
- **Similarité contextuelle** : c'est une représentation plus expressive

2.9 Normalisation

Former des réseaux de neurones profonds à la pointe de la technologie est coûteux en calcul. Une façon de réduire le temps d'entraînement est de normaliser les activités des neurones. C'est une technique souvent appliquée dans le cadre de la préparation de données. L'objectif de la normalisation est de modifier les valeurs des colonnes numériques du jeu de données pour utiliser une échelle commune, sans distorsion des différences dans les plages de valeurs ni perte d'informations. La normalisation est également nécessaire pour certains algorithmes afin de modéliser les données correctement. Cela réduit considérablement le temps de formation dans les réseaux de neurones.

2.9.1 La normalisation des couches

La normalisation des couches est une approche de la normalisation qui utilise des statistiques indépendantes du mini-lot. Pour comprendre la normalisation des couches, un mini-lot est constitué de plusieurs exemples avec le même nombre de caractéristiques. Les mini-lots sont des matrices (ou tenseurs si chaque entrée est multidimensionnelle), un axe correspondant au lot et l'autre axe, correspondant aux dimensions de la fonction. La normalisation de lot normalise les entités en entrée dans la dimension de lot. La fonctionnalité principale de la normalisation de couche est la normalisation des entrées à travers les fonctionnalités.

Formule mathématique pour la normalisation de couches (Référence : JIMMY LEI BA, 13 Janvier 2018) :

$$\begin{aligned}\mu^l &= \frac{1}{H} \sum_{i=1}^H a_i^l \\ a^l &= \sqrt{\frac{1}{H} \sum_{i=1}^H (a_i^l - \mu^l)^2} \\ h_i &= f\left(\frac{g_i}{\sigma_i}(a_i - \mu_i) + b_i\right)\end{aligned}\tag{2.13}$$

μ et σ représentent la moyenne et l'écart type

La normalisation de couches peut être résumée comme normalisant les entrées totales a_i d'un neurone via les deux scalaires μ et σ . Il apprend également un biais adaptatif b et un gain g pour chaque neurone après la normalisation.

2.9.2 Avantages

- L'indépendance entre les entrées signifie que chaque entrée a une opération de normalisation différente, ce qui permet d'utiliser des tailles de mini-lots arbitraires.
- La normalisation des couches n'impose aucune contrainte sur la taille du lot.
- Le calcul est extrêmement rapide
- Les résultats expérimentaux montrent que la normalisation des couches fonctionne bien pour les réseaux de neurones récurrents.

2.10 Conclusion

Dans ce chapitre, nous avons vu la puissance qu'a apporté le Transformer par rapport au RNN et CNN pour les tâches de traduction. Il a défini une nouvelle référence de pointe et offre une base solide suggérant que l'avenir de nombreuses architectures de Deep Learning se tourneront vers les mécanismes d'auto-attention. Nous avons analysé le document « L'attention est tout ce dont vous avez besoin », et on a expliqué les différentes techniques employées, ainsi que l'analyse des résultats de Google. Aussi on a détaillé le Transformer de comment se fait le calcul de l'auto-intention avec l'attention multi-tête, et la façon dont le décodeur prédit les mots. Puis on a discuté des fonctions de perte et d'optimisation qui ont été utilisées, ainsi que du score BLEU. Enfin on a parlé de la normalisation ainsi que de la représentation vectorielle.

3

Chapitre d'implémentation

3.1 Introduction

Dans ce dernier chapitre. On parlera des différents choix d'hypermètres du modèle ainsi que ceux de la fonction d'optimisation. On verra aussi les parties essentielles du code source avec explication de paramètres, Puis on discutera à propos du résultat obtenu lors de la phase de formation du modèle. Enfin, les différentes librairies et outils qui ont aidé à la réalisation de ce projet seront présentée. Et on terminera par une conclusion.

Pour ce qui est de la formation, la tâche de la traduction demande des ressources assez grandes pour être réalisée. Google pour ses tests a dû utiliser 8 GPU P100. Donc pour pouvoir faire cette tâche, il fallait un énorme matériel, et c'est là qu'intervient Google Colaboratory.

Google Colaboratory est un service cloud gratuit de Google qui permet de gérer les modèles d'apprentissage en profondeur ou d'apprentissage automatique dans le cloud. Il offre une machine virtuelle très puissante gratuitement pour une durée de 12 heures en session qui est renouvelable. La machine contient un GPU Tesla K80 avec 15 GO, ainsi qu'un total de 12.9 Go de RAM, et les librairies du Machine Learning sont déjà installées. Il permet d'écrire et d'exécuter du code, de sauvegarder et partager des analyses, et d'accéder à de puissantes ressources informatiques, tout cela gratuitement, depuis un navigateur. Il offre un jupyter notebook avec python, Il ne nécessite aucune configuration et il s'exécute entièrement dans le cloud.

Et donc pour pouvoir réaliser la tâche de traduction, ce service a été utilisé.

3.2 L'implémentation

3.2.1 Le corpus de données et la taille du lot

La formation a été faite sur le jeu de données anglais-français standard du WMT 2014, composé de 36 millions de phrases et de jetons divisés en un vocabulaire de 32 000 mots. Les paires de phrases ont été regroupées par longueur approximative de la séquence. Chaque lot de formation contenait un ensemble de paires de phrases contenant environ 25 000 jetons source et 25 000 jetons cible.

3.2.2 Matériel et le temps d'entraînement

Le modèle a été formé sur une seule machine avec 1 GPU NVIDIA K80. Chaque étape de la formation prenait moyennement environ 150 secondes. Le big modèle a été formé pour un total de 521 000 pas(521k). La durée d'entraînement peut durer jusqu'à 13.5 jours avec 1 GPU K80.

3.2.3 Les hyperparamètres

Le big modèle est utilisé en vue de la grande taille du corpus. Pour plus d'informations, veuillez voir cette Figure 2.30 à la page 63. Une taille allant à 4096 de lots a été utilisée. Le nombre de couches cachés est de 4096. Et le nombre d'entrée et de sortie est de 1024.

3.2.4 Optimiseur

Les auteurs ont utilisé l'optimiseur Adam avec la configuration suivante :

$\beta_1 = 0.9$, $\beta_2 = 0.997$, et $\epsilon = 10^9$. Ils ont utilisé un programme dans lequel ils ont progressivement augmenté le taux d'apprentissage, puis ils l'ont diminué selon la formule suivante :

$$lrate = d_{model}^{0.5} * \min(step_{num}^{-0.5}, step_{num} * warmup_{steps}^{-1.5}) \quad (3.1)$$

avec $warmup_{steps} = 8000$

3.2.5 L'Abandon résiduel (Residual Dropout)

Les auteurs ont appliqué la suppression (dropout) à chaque sous-couche avant de l'ajouter à l'entrée d'origine. Ils ont également appliqué la suppression à la somme des incorporations et aux codages de position. Le taux d'abandon était de 0,1 par défaut.

3.2.6 Lissage des étiquettes (Label smoothing)

Pour pénaliser le modèle lorsqu'il devient trop confiant dans ses prévisions, les auteurs ont procédé au lissage des étiquettes.

3.2.7 Entrainement

```
t2t-trainer \
--data_dir=$DATA_DIR \
--problem=$PROBLEM \
--model=$MODEL \
--hparams_set=$HPARAMS \
--hparams='batch_size=$batch_size' \
--output_dir=$TRAIN_DIR \
--train_steps=$train_steps \
--eval_steps=$eval_steps
```

FIGURE 3.1 – Code pour entraîner le modèle

Avec :

- train_steps = 600000. C'est le nombre total d'étapes pour toutes les époques.
- eval_steps = 100. C'est le nombre d'étapes à effectuer pour chaque évaluation.
- batch_size = 4096. La taille du lot.
- save_checkpoints_steps = 1000. On Sauvegarde chaque 1000 étapes.
- PROBLEM="translate_enfr_wmt32k". On spécifie un problème de traduction de l'anglais vers le français en utilisant un vocabulaire de 32.000 mots unique.
- MODEL="transformer". Le modèle.
- HPARAMS="transformer_big". On choisit le big modèle. Car le corpus anglais-français contient énormément de données.

3.2.8 Prédiction

```
inp = "the business of the house"
output = "les affaires de la maison"

DECODE_FILE="Translate_En_Fr_decode_this.txt"
echo $inp >> $DECODE_FILE
echo -e $output > "ref-translation.fr"

t2t-decoder \
--data_dir=$DATA_DIR \
--problem=$PROBLEM \
--model=$MODEL \
--hparams_set=$HPARAMS \
--output_dir=$TRAIN_DIR \
--decode_hparams="beam_size=$BEAM_SIZE,alpha=$ALPHA" \
--decode_from_file=$DECODE_FILE \
--decode_to_file="translation.en"
```

FIGURE 3.2 – Code pour la prédiction

3.3 Résultat

3.3.1 Le BLEU

Pour ce qui du BLEU score, le résultat obtenu est : 21.41. Celui-ci n'a pas atteint l'état de l'art. La durée d'entraînement est estimée à 13.5 jours. Concernant le papier de recherche, ils ont testé le modèle sur 8 GPU, qui sont plus puissants que celui utilisé. D'où le fait d'une grande marge de différence de métrique. Donc pour avoir une bonne qualité, il faut entraîner le modèle 8 fois plus au minimum.

3.3.2 Le BLEU approximative

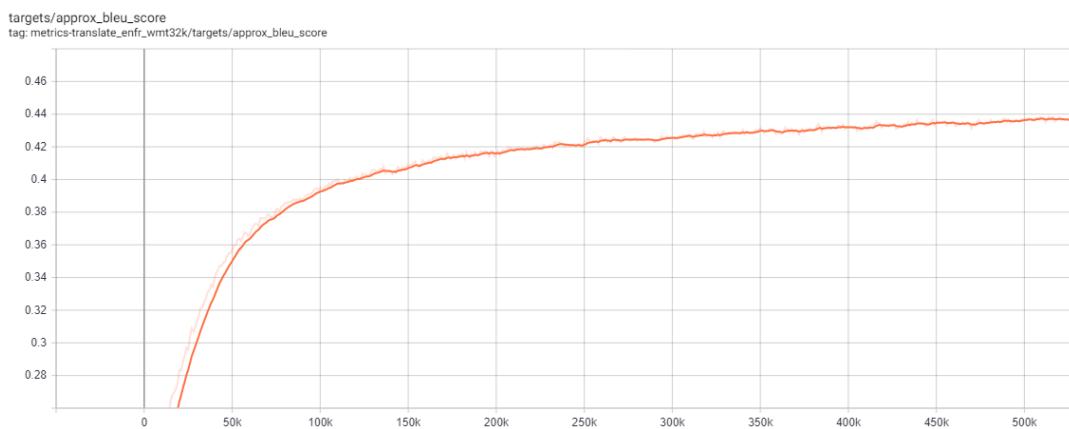


FIGURE 3.3 – Le BLEU approximative

Le BLEU approximatif a la différence du BLEU, celui-ci calcule le score avec des sous phrases au lieu de la séquence entière. On remarque que cela a donné de bons résultats, qui ont atteint un score de 43.

3.3.3 La perplexité

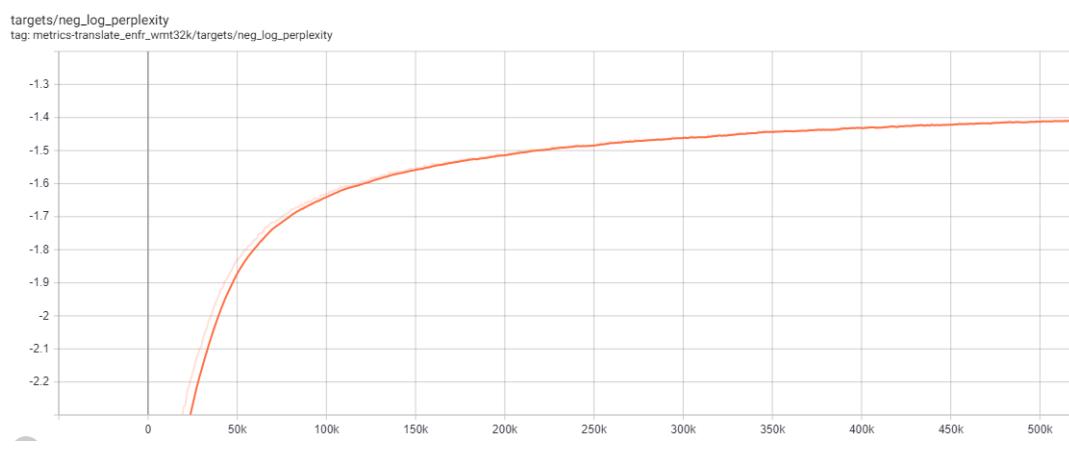


FIGURE 3.4 – La perplexité

Un bon modèle de langage est un modèle qui a tendance à attribuer des probabilités plus élevées aux données de test (il est capable de très bien prédire les phrases dans les données de test). La perplexité logarithme négative est une mesure de qualité avec laquelle un modèle de distribution de probabilités prédit un échantillon, c'est à dire qu'il prédit des données de test et évalue les modèles de langage.

Une très faible perplexité logarithme négative indique de bons résultats. A chaque fois que nous minimisons la perplexité, nous maximisons la probabilité. Un score de 0 signifie un score parfait.

Un score de -1.42 a été atteint.

3.3.4 La fonction de perte



FIGURE 3.5 – La fonction de perte

On remarque que le modèle n'a pas réussi à converger. il faut encore plus d'entraînement pour réussir à converger. Le modèle s'est entraîné sur 521.000 étapes d'entraînement sur un seul GPU, alors que celui de Google s'entraîne sur 8 GPU.

3.3.5 Comparaison de traductions

Phrase en entrée	Google Traduction	Modèle Transformer
the animal did not cross the river because it was too tired	L'animal n'a pas traversé la rivière car il était trop fatigué	L'animal n'a pas traversé la rivière parce qu'il était trop fatigué.
The city councilmen refused the female emonstrators a permit because they feared violence.	Les conseillers municipaux ont refusé un permis aux manifestantes parce qu'elles craignaient la violence.	Les conseillers municipaux ont refusé aux manifestants nourris un permis parce qu'ils craignaient la violence
The child finished the apple because it was delicious	L'enfant a fini la pomme parce que c'était délicieux	L'enfant a fini la pomme parce qu'elle était délicieuse et délicieuse, car elle était délicieuse.
my brother bought a car because it has his license	mon frère a acheté une voiture parce qu'elle a son permis	mon frère a acheté une voiture parce qu'il a sa licence de conduire une voiture parce qu'il a sa licence.
poverty is still existing in many countries. It is making people die from hunger everyday	la pauvreté existe encore dans de nombreux pays. il fait mourir chaque jour la faim	La pauvreté est toujours présente dans de nombreux pays, ce qui fait mourir des gens de faim chaque jour dans le monde.

FIGURE 3.6 – Comparaison de traductions

La figure ci-dessus contient une comparaison de résultats entre Google Traduction et le modèle Transformer formé. Malgré une métrique assez faible, le Transformer a réussi là où Google Translate a échoué. Donc avec une bonne métrique, de meilleurs résultats auraient pu être obtenus.

La première phrase a été bien traduite par les deux modèles. Ainsi qu'ils ont bien référencé le mot « il » vers « animal ».

Pour la 2 ème entrée, les 2 modèles contiennent des problèmes, mais celle de Google Traduction est plus pertinente, du fait où il s'est trompé sur la référence du mot « elle », qui devrait être plutôt le mot « il », mais celle-ci n'est pas survenue dans le Transformer. Sauf que celui-ci n'a pas reconnu le mot féminin.

Pour la troisième phrase, le mot « it » a bien fait référence à pomme, et ceci en dépit d'une mauvaise qualité de traduction qui est justifiée.

Concernant la 4 et la 5 ème phrase, Transformer a dépassé de loin Google Traduction.

3.3.6 Visualisation de l'attention

Dans la figure suivante, on remarque le mot « poverty » a une forte attention.

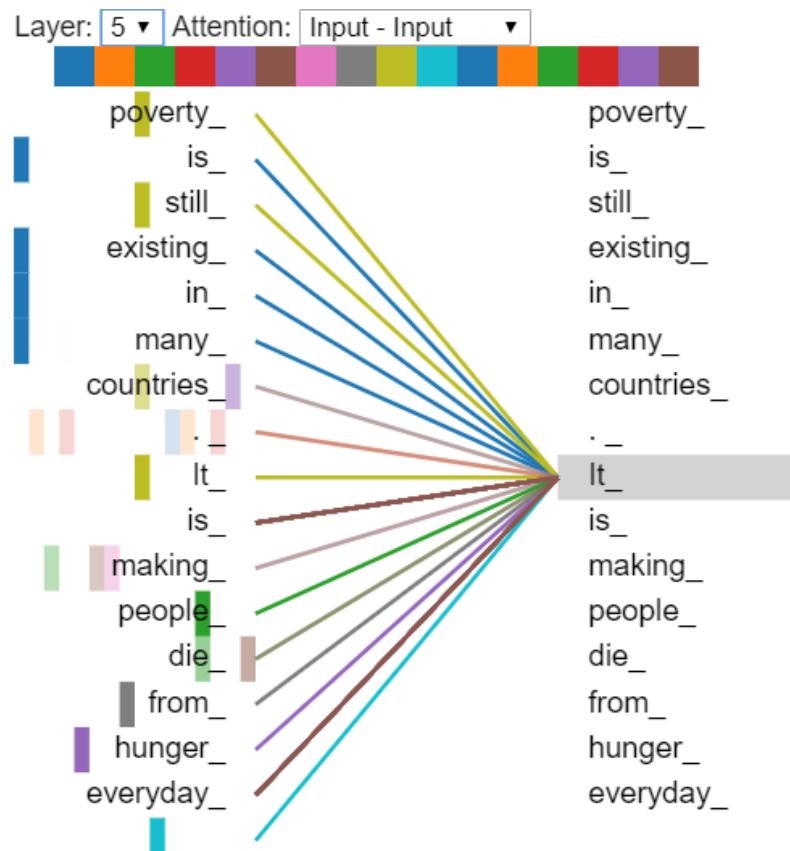


FIGURE 3.7 – L'attention visuelle avec 16 têtes dans la couche 5

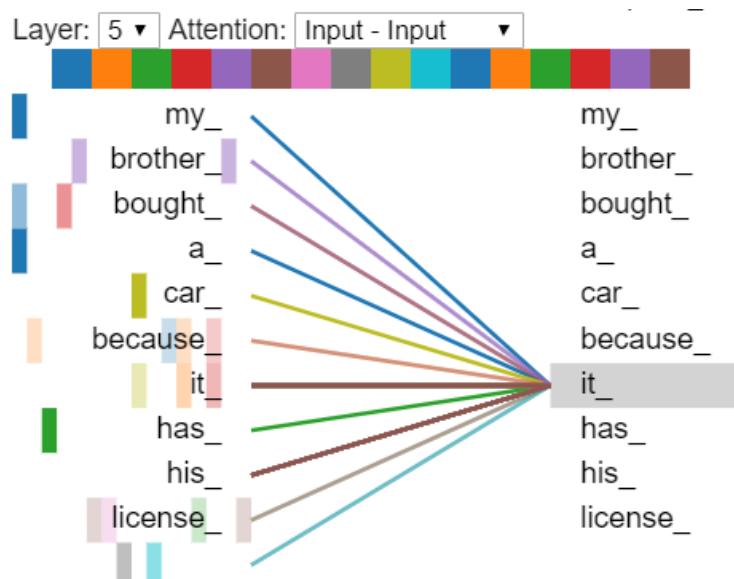


FIGURE 3.8 – L'attention visuelle avec 16 têtes dans la couche 5

3.4 Les technologies utilisées

3.4.1 Python



FIGURE 3.9 – Python logo

Python est un langage interprété populaire et puissant. Contrairement à R, Python est un langage complet et une plate-forme que vous pouvez utiliser à la fois pour la recherche et le développement et pour le développement de systèmes de production.

Dans le Machine Learning il permet de gagner un temps considérable dans l'élaboration du modèle, grâce à sa facilité syntaxique et sémantique ainsi que les différents outils disponibles pour celui-ci. Il est le principal langage utilisé dans la Machine Learning

3.4.2 Tensorflow



FIGURE 3.10 – TensorFlow logo

TensorFlow est une bibliothèque de logiciels open source pour le calcul numérique utilisant des graphiques de flux de données. Les nœuds de graphique représentent des opérations mathématiques, tandis que les arêtes de graphique représentent les tableaux de données multidimensionnels (tenseurs) qui circulent entre eux. Cette architecture flexible permet de déployer des calculs sur un ou plusieurs processeurs ou GPU sur un ordinateur de bureau, un serveur ou un périphérique mobile sans réécrire le code. TensorFlow inclut également TensorBoard, une boîte à outils de visualisation de données.

TensorFlow a été développé à l'origine par des chercheurs et des ingénieurs travaillant au sein de l'équipe Google Brain au sein de l'organisation de recherche sur l'intelligence artificielle de Google dans le but de mener des recherches sur la machine learning et les réseaux de neurones profonds. Le système est suffisamment général pour pouvoir s'appliquer à une grande variété d'autres domaines également.

3.4.3 Tensor2Tensor (T2T)

Tensor2Tensor, ou T2T, est une bibliothèque de modèles d'apprentissage en profondeur et permet d'utiliser des jeux de données conçus pour rendre l'apprentissage en profondeur plus accessible et pour accélérer la recherche sur le Machine Learning. T2T est activement utilisé et mis à jour par des chercheurs et des ingénieurs au sein de l'équipe Google Brain et d'une communauté d'utilisateurs.

C'est un outil très utile pour la tâche de traitement du langage naturel (TLN). Par exemple, l'utilisation de mots incorporés a révolutionné l'efficacité des techniques de compréhension du langage. Au lieu de coder et d'entraîner ce réseau de neurones à partir de rien, T2T fournit un cadre pour une formation rapide et facile et la production de modèles.

La bibliothèque T2T a été conçue pour être utilisée avec un script shell, mais c'est possible l'envelopper pour une utilisation en Python. L'API est multi-modulaire, ce qui signifie que n'importe quel modèle intégré peut être utilisé avec tout type de données (texte, image, audio, etc.). Cependant, les auteurs de l'API fournissent des ensembles de données suggérés et des modèles pour des tâches spécifiques telles que la traduction, la synthèse de texte, la reconnaissance vocale, etc.

3.5 Conclusion

Ce chapitre a été consacré à l'implémentation et à l'analyse ainsi que la visualisation des résultats, tout en présentant les hyperparamètres utilisés et les différentes technologies. Concernant le résultat, il n'a pas atteint la pointe de la technologie avec un score BLEU assez faible de 21.41.

Le projet sera clôturé avec une conclusion générale.

Conclusion général

Ce projet m'a permis de mettre en pratique mon esprit d'étude, d'analyse et de critique, j'ai pu mettre en pratique les connaissances que j'avais acquises, lors de la phase de mon apprentissage, et de découvrir le monde des projets de recherche. Cela m'a permis d'approfondir mes connaissances en analyse, a l'apprentissage de nouvelles technologies, et aussi la découverte du monde de la recherche.

Le projet a été réalisé dans le but de montrer la puissance du Transformer qui est une architecture de pointe. Et a quel point elle a pu surpasser les autres, que ce soit en termes de qualité ou de performance.

Ce travail a été composé de trois étapes. La première a été consacrée à la compréhension du traitement automatique du langage naturel avec la technologie moderne du Deep Learning. L'étape suivante décrivait l'architecture ainsi que les mécanismes et techniques utilisés, avec une comparaison avec les autres architectures, et il contenait une analyse du document de recherche « L'attention est tout ce dont vous avez besoin ». Dans la dernière étape la phase implémentation a été entamée, en spécifiant les hyperparamètre utilisés, avec une partie du code, et on a présenté les technologies utilisées.

Concernant les résultats obtenus, en se basant sur la métrique BLEU, celle-ci a été inférieure à l'état de l'art du Transformer, même par rapport aux autres architectures. Cela est dû au fait que la formation a été faite sur un matériel assez faible, donc il fallait entraîner le modèle encore plus pour avoir de meilleurs résultats. La métrique résultante n'aurait pu être obtenue en utilisant les RNN, car leur processus de formations est lent. Mais cela n'a pas empêché le Transformer de traduire quelques phrases avec une bonne qualité, et le tout visualisant l'attention des mots pertinents.

Concernant mes perspectives, le Transformer est l'avenir du TLN, grâce à son mécanisme d'auto-attention. De cette manière, l'architecture du transformateur ouvre une nouvelle phase de développement de nouveaux modèles. C'est pourquoi nous verrons beaucoup de nouvelles approches de la TLN en 2019.

La publication de l'architecture du transformateur a créé une nouvelle base de références pour les approches d'apprentissage approfondi en TLN. Les gens ont pu constater le potentiel de cette nouvelle architecture et ont rapidement essayé de trouver des moyens de l'intégrer à de nouvelles approches plus avancées des problèmes de TLN, tels que les Universal Transformers qui est une architecture améliorant encore la qualité de traduction du Transformer de base, et qui corrige les lacunes du Transformer de base. Google a créé Bert Transformer qui consiste à appliquer la formation bidirectionnelle de Transformer, avant les efforts consistaient à regarder une séquence de texte de gauche à droite ou combinée d'une formation de gauche à droite et de droite à gauche. Les résultats de BERT qu'un modèle de langage formé de manière bidirectionnelle peut avoir un sens plus profond du contexte et du flux de langage que les modèles de langage à une seule direction. On a aussi le Transformer XL qui s'appuie sur le Transformateur d'origine et permet le traitement simultané de séquences d'entrées plus longues. Cela signifie que les séquences d'entrée n'ont pas besoin d'être divisées en longueurs fixes arbitraires car elles peuvent suivre les limites du langage naturel telles que les phrases et les paragraphes, cela aide à comprendre un contexte plus profond sur plusieurs phrases, paragraphes et textes potentiellement plus longs tels que des articles. On note aussi les X-Net Transformer qui

intègrent les idées de Transformer-XL, il surpasse de loin le BERT sur 20 tâches, souvent très largement, et obtient des résultats de pointe sur 18 tâches, notamment la réponse aux questions, l'inférence en langage naturel, l'analyse des sentiments et le classement des documents. On a aussi Open-AI Transformer, avec celle-ci, un générateur de texte qui égale le niveau de la presse a été créé.

Les nouvelles architectures ont facilité la formation de modèles sur des jeux de données auparavant considérés comme trop volumineux et coûteux en termes d'apprentissage. La plupart des gens ne pourraient pas utiliser ces jeux de données et il serait probablement toujours impossible à tout le monde de recycler leurs propres modèles, même si les nouvelles architectures le rendaient plus facile. En conséquence, cela signifie que les utilisateurs devaient mettre à disposition leurs modèles pré-formés pour les utiliser immédiatement ou pour les perfectionner et les ajuster au besoin.

Bibliographie

- AI, Google (Jeudi, 31 août 2017). “Transformer : A Novel Neural Network Architecture for Language Understanding”. In : URL : <https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html>.
- (Mardi, 27 Septembre 2016). “A Neural Network for Machine Translation, at Production Scale”. In : URL : <https://ai.googleblog.com/2016/09/a-neural-network-for-machine.html>.
- AI, Team Google (12 Juin 2017). “Attention Is All You Need”. In : URL : <https://arxiv.org/abs/1706.03762>.
- ALAMMAR, Jay (20 Février 2017). “The Illustrated Transformer”. In : URL : <http://jalammar.github.io/illustrated-transformer/>.
- BRIJESH (9Decembre 2018). “Replac your RNN and LSTM with Attention base Transformer model for NLP”. In : URL : <https://androidkt.com/attention-base-transformer-for-nlp/>.
- BROWNLEE, Jason (3Juillet 2017). “Gentle Introduction to the Adam Optimization Algorithm for Deep Learning”. In : URL : <https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/>.
- (12 Juillet 2017). “How to One Hot Encode Sequence Data in Python”. In : URL : <https://machinelearningmastery.com/how-to-one-hot-encode-sequence-data-in-python/>.
- (28 Janvier 2018). “Loss and Loss Functions for Training Deep Learning Neural Networks”. In : URL : <https://machinelearningmastery.com/loss-and-loss-functions-for-training-deep-learning-neural-networks/>.
- CHROMIAK, Michał (Mardi, 12 Septembre 2017). “The Transformer – Attention is all you need”. In : URL : <https://mchromiak.github.io/articles/2017/Sep/12/Transformer-Attention-is-all-you-need/#.XSC0muuiG73>.
- COLLIS, Jaron (16 Avril 2017). “Glossary of Deep Learning : Word Embedding”. In : URL : <https://medium.com/deeper-learning/glossary-of-deep-learning-word-embedding-f90c3cec34ca>.
- DAHAL, Paras. “Classification and Loss Evaluation - Softmax and Cross Entropy Losses”. In : (). URL : <https://deeplearnnotes.io/softmax-crossentropy>.
- DENOYES, Paul (10 Avril 2019). “BERT : Le "Transformer model" qui s'entraîne et qui représente”. In : URL : <https://lesdieuxducode.com/blog/2019/4/bert--le-transformer-model-qui-sentraigne-et-qui-represente>.
- DONGES, Niklas (28 Juin 2019). “INTRODUCTION À LA PNL”. In : URL : <https://builtin.com/data-science/introduction-nlp>.
- GEHRING, Michael Aulin (9 Mai 2017). “A novel approach to neural machine translation”. In : URL : <https://code.fb.com/ml-applications/a-novel-approach-to-neural-machine-translation/>.
- GEITGEY, Adam (22 Août 2016). “Machine Learning is Fun Part 5 : Language Translation with Deep Learning and the Magic of Sequences”. In : URL : <https://geitgey.com/jupyter/nn/intro.html>.

- //medium.com/@ageitgey/machine-learning-is-fun-part-5-language-translation-with-deep-learning-and-the-magic-of-sequences-2ace0acca0aa.
- GILL, NAVDEEP SINGH. "Overview of Artificial Intelligence and Natural Language Processing". In : (). URL : https://www.upwork.com/hiring/for-clients/artificial-intelligence-and-natural-language-processing-in-big-data/?utm_content=bufferfff6c&utm_medium=social&utm_source=twitter.com&utm_campaign=buffer.
- GRAVES, Alex (14 Novembre 2012). "Sequence Transduction with Recurrent Neural Networks". In : p. 1-9. URL : <https://arxiv.org/abs/1211.3711>.
- HORAN, Cathal (12 Mars 2019). "Ten trends in Deep learning NLP". In : URL : <https://blog.floydhub.com/ten-trends-in-deep-learning-nlp/>.
- HWANG, Yitaek (6 Octobre 2017). "<https://www.iotforall.com/transformer-vs-deepL-attention-based-machine-translation/>". In : URL : <https://www.iotforall.com/transformer-vs-deepL-attention-based-machine-translation/>.
- ICONIC. "Neural Machine Translation". In : (). URL : <https://lioniq.com/neural-machine-translation-today-b3beabf80ce4>.
- JIMMY LEI BA Jamie Ryan Kiros, Geoffrey E. Hinton (13 Janvier 2018). "Layer Normalization". In : p. 2-3. URL : <https://arxiv.org/abs/1607.06450>.
- JÉRÉMY, Clément Dimitri (17 Octobre 2018). "Introduction au NLP : le traitement de texte automatisé". In : URL : <https://blog.coddity.com/article/natural-language-processing/>.
- KADAM, Shirish (23 Decembre 2016). "Dependency Parsing in NLP". In : URL : <https://shirishkadam.com/2016/12/23/dependency-parsing-in-nlp/>.
- KEITAKURITA (13 Janvier 2018). "Weight Normalization and Layer Normalization Explained (Normalization in Deep Learning Part 2)". In : URL : <http://mlexplained.com/2018/01/13/weight-normalization-and-layer-normalization-explained-normalization-in-deep-learning-part-2/>.
- (29 decembre 2017). "Paper Dissected : "Attention is All You Need" Explained". In : URL : <http://mlexplained.com/2017/12/29/attention-is-all-you-need-explained/>.
- LIU, Jerry (7Octobre 2017). "Neural Machine Translation Today". In : URL : <https://lioniq.com/neural-machine-translation-today-b3beabf80ce4>.
- LYNN-EVANS, Samuel (27 Septembre 2018). "How to code The Transformer in Pytorch". In : URL : <https://towardsdatascience.com/how-to-code-the-transformer-in-pytorch-24db27c8f9ec>.
- MACHINE LEARNING CHEATSHEET MACHINE LEARNING CHEATSHEET, Membre du (2017). "Understand the Softmax Function in Minutes". In : URL : https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html.
- MISHRA, Prakhar (20 Février 2018). "Beam Search—A Search Strategy". In : URL : <https://hackernoon.com/beam-search-a-search-strategy-5d92fb7817f>.
- OLIVIER, Pauline. "Introduction au NLP (Partie II)". In : (). URL : <https://ekino.com/articles/introduction-au-nlp-partie-ii>.
- PAN, Hazel Mae (14 Novembre 2016). "How BLEU Measures Translation and Why It Matters". In : URL : <https://slator.com/technology/how-bleu-measures-translation-and-why-it-matters/>.

- RAJ, Desh (16 Septembre 2017). “Metrics for NLG evaluation”. In : URL : <https://medium.com/explorations-in-language-and-learning/metrics-for-nlg-evaluation-c89b6a781054>.
- RECHERCHE, L'équipe du papier de (Juillet 2002). “BLEU : a Method for Automatic Evaluation of Machine Translation”. In : p. 3-5.
- RESPENSABLE (28 Octobre 2018). “Why Natural Language Processing (NLP) is a core AI Technology”. In : URL : <https://witanworld.com/blog/2018/10/28/naturallanguageprocessing-nlp/>.
- RICARDOKLEINKLEIN (16 Novembre 2017[a]). “Attention is all you need's review”. In : URL : <https://ricardokleinklein.github.io/2017/11/16/Attention-is-all-you-need.html>.
- (16 Novembre 2017[b]). “Attention is all you need's review”. In : URL : <https://ricardokleinklein.github.io/2017/11/16/Attention-is-all-you-need.html>.
- (16 Novembre 2017[c]). “Attention is all you need's review”. In : URL : <https://ricardokleinklein.github.io/2017/11/16/Attention-is-all-you-need.html>.
- (26-27 juin 2014). “<http://www.statmt.org/wmt14/index.html>”. In : URL : <http://www.statmt.org/wmt14/index.html>.
- (3 Avril 2018). “The Annotated Transformer”. In : URL : <http://nlp.seas.harvard.edu/2018/04/03/attention.html>.
- SATHIYAKUGAN, Balakrishnan (24 Juillet 2018). “Learn Natural Language Processing from scratch”. In : URL : <https://blog.goodaudience.com/learn-natural-language-processing-from-scratch-7893314725ff>.
- SAWLA, Srishti (9 Mai 2018). “Introduction to Natural Language Processing”. In : URL : <https://medium.com/greyatom/introduction-tonatural-language-processing-78baac3c602b>.
- SHAFFY, Athif (8 Novembre 2017). “Vector Representations of Text for Machine Learning”. In : URL : <https://medium.com/@athif.shaffy/one-hot-encoding-of-text-b69124bef0a7>.
- SHIBUYA, Naoki (28 Octobre 2018). “Demystifying Cross-Entropy”. In : URL : <https://towardsdatascience.com/demystifying-cross-entropy-e80e3ad54a8>.
- TRACEY, Thomas (Aug 29, 2018). “Language Translation with RNNs”. In : URL : <https://towardsdatascience.com/language-translation-with-rnns-d84d43b40571>.
- UNIQTECH (30 Juin 2019). “Understand the Softmax Function in Minutes”. In : URL : <https://medium.com/data-science-bootcamp/understand-the-softmax-function-in-minutes-f3a59641e86d>.
- “Évaluation des résultats de la traduction : WER, SER et BLEU” (25 Juin 2012). In : URL : <http://wikimemoires.net/2012/06/evaluation-des-resultats-de-la-traduction-wer-ser-et-bleu/>.
- YSE, Diego Lopez (15 Janvier). “Your Guide to Natural Language Processing (NLP)”. In : URL : <https://towardsdatascience.com/your-guide-to-natural-language-processing-nlp-48ea2511f6e1>.
- ZHANG, Aston et al. (2019). *Dive into Deep Learning*. <http://www.d2l.ai>.

ZHANG, Leona (18 Septembre 2018). “Self-Attention Mechanisms in Natural Language Processing”. In : URL : <https://dzone.com/articles/self-attention-mechanisms-in-natural-language-proc>.