



المدرسة الوطنية للإعلام الآلي
(المعهد الوطني للتكوين في الإعلام الآلي سابقا)
École nationale Supérieure d'Informatique
ex. INI (Institut National de formation en Informatique)

Mémoire de fin d'études

Pour l'obtention du diplôme d'Ingénieur d'État en Informatique

Option : Systèmes Informatiques

Création d'un corpus de l'aphasie de Broca et développement d'un système Speech-to-speech de réhabilitation de la parole

Réalisé par :

BELGOUMRI Mohammed
Djameleddine
im_belgoumri@esi.dz

Encadré par :

Pr. SMAILI Kamel
smaili@loria.fr
Dr. LANGLOIS David
david.langlois@loria.fr
Dr. ZAKARIA Chahnez
c_zakaria@esi.dz

إهداء

إلى أبي وأمي، صاحببي الفضل علي بعد الله،
إلى أخي وأختي وجميع أهلي،
إلى شوقي، أعز أصدقائي،
إلى أولئك الذين أسأل عونهم إذا حدثت مشكلة،
وأشاركهم الاحتفال إذا حلت،
إليكم جميعا، أهدي هذا العمل. أنتم من شاركتموني تعبه وسهره وهمه،
وليس من أحد أحقر منكم أن يشاركوني فرحته.

شكر و عرفان

الحمد لله أولاً أن وفقني لهذ ما كنت لأبلغه لولا أن وفقني، وأعاني على إتمامه وما كنت لأتمه لولا أن أعايني. أحمدك اللهم ملء السماوات والأرض وملء مابينهما وملء ما شئت دون ذلك، لا أحصي ثناء عليك، أنت كما أثنيت على نفسك.

أشكر شakra جزيلاً الهيئة المشرفة على هذا المشروع، الدكتورة زكريا شهناز، البروفيسور سماعيلى كمال والدكتور Langlois David ، الذين لولا نصحهم وتوجيههم، ما كان هذا العمل قد نجح. أشكر صبركم على أخطائي وأشكر تصويبكم إياها، وأشكركم فوق كل شيء، على ما تعلمت منكم.

أشكر أيضاً والدي الذين حملوا قلق هذا العمل وتعبه بقدر ما حملتهما أنا. وأخي وأختي الذين أسندوني ساعة تعبي وأضحكوني ساعة حزني وغضبي.

الشكر كذلك لأولئك الذين ساعدوني ولم يخلوا علي بمشورة أو رأي، لا لشيء سوى كرم أنفسهم وطيبة معدنهم. نزيم ونهى وبلال ونسيم وزكريا وماسينيسا وبرهان ومليك وهيثم والدكتور إسماعيل عدون.

والشكر له آخراً كما له الشكر أولاً، هو الذي أنعم علي بهؤلاء جميعاً، وبهذا كله فالحمد له ملء ما أنعم، والحمد له ملء ما شاء

Résumé

L'aphasie est un trouble de langage qui résulte d'une lésion cérébrale (typiquement suite à un AVC). L'aphasie de Broca est une déficience de la production du langage causée par une lésion dans l'aire de Broca, une région du lobe frontal gauche du cerveau, responsable de la production de la parole. Une personne atteinte d'aphasie de Broca peut avoir des difficultés à articuler les mots et les phrases. Cependant, elle peut en général comprendre ce qui est dit. L'aphasie de Broca est associé à une diminution de la qualité de vie et à une augmentation du risque de dépression et de tentative de suicide.

La rééducation de la parole est le traitement le plus couramment prescrit aux personnes atteintes d'aphasie de Broca. En dépit de son efficacité, la rééducation de la parole est un traitement coûteux en termes de temps, argent et ressources humaines. Cela la rend indisponible à un grand nombre de personnes souffrant de l'aphasie de Broca.

L'utilisation des techniques basées sur le traitement automatique du langage pour améliorer la qualité de vie de ses individus est une voie d'exploration émergente qui a reçu beaucoup d'attention par les chercheurs pendant les années dernières.

Dans ce projet de fin d'études, nous nous intéressons à l'utilisation de la traduction automatique et la reconnaissance automatique de la parole pour automatiser une partie de la procédure de réhabilitation des personnes touchées par l'aphasie de Broca.

Dans ce but, nous menons une étude bibliographique dans laquelle nous introduisons l'aphasie de Broca, ses causes, ses effets, les problèmes avec les traitements classiques et les travaux existant sur la traduction automatique et la reconnaissance automatique de la parole.

Ensuite, nous proposons un système qui combine un modèle de traduction automatique avec un de reconnaissance automatique de la parole pour corriger la parole aphasique en français et nous réalisons le premier de ses modèles. Enfin, nous présentons les résultats de notre travail sous forme d'un corpus de reconnaissance automatique de la parole et d'un modèle de traduction d'un score BLEU de 79.61%.

Mots clés — Aphasie de Broca, Apprentissage automatique, Traitement automatique du langage, Traduction automatique, Reconnaissance automatique de la parole, Transformeur.

Abstract

Aphasia is a language disorder caused by brain damage (most commonly a stroke). Broca's aphasia is a form of aphasia that impairs language production. It is caused by an injury to Broca's Area, an area of the frontal lobe of the brain; responsible for language decoding. A person suffering from Broca's aphasia may find it difficult to articulate words and sentences. However, they generally can understand what is said to them. This form of aphasia is associated with a lower quality of life and a higher risk of depression and suicide.

Speech therapy is the most commonly prescribed remedy to people with Broca's aphasia. Despite its effectiveness, it remains an expensive, time-consuming, and effort-heavy process. This makes it inaccessible to a significant number of people with aphasia.

The use of natural language processing-based techniques to improve these people's quality of life is an emerging research avenue that has enjoyed the attention of many researchers in recent years.

In this graduation project, we are interested in the use of machine translation and automatic speech recognition to partially automate the rehabilitation of people with aphasia.

To this end, we conduct a bibliographic study in which introduce aphasia, its causes, consequences, the problems of classical treatment methods, and a literature review the existing works pertaining to machine translation and automatic speech recognition.

We then design a system that corrects french aphasic speech by combining a translation model with a speech recognition model, the former of which we implement. We finish by presenting the results of our work: a corpus for automatic speech recognition, and a translation model with a BLEU score of 79.61%.

Keywords — Broca aphasia, Machine learning, Natural language processing, Machine translation, Automatic speech recognition, Transformer.

ملخص

الحبسة إضطراب لغوي ناتج عن تلف في الدماغ، غالباً نتيجة سكتة دماغية. حبسة بروكا حبسة تنتج عن إصابة في منطقة بروكا، وهي منطقة في الفص الجبهي الأيسر للدماغ تعنى بإنتاج الكلام. قد يجد المصاب بحبسة بروكا صعوبة في تكوين الجمل والكلمات، إلا أنه عادةً يفهم ما يقال. ترتبط هذه الحبسة بتدني مسوى العيش وارتفاع خطر الاكتئاب والانتحار.

علاج النطق هو أكثر العلاجات وصفاً للمصابين بحبسة بروكا. رغم نجاعته، فهو يظل مكلفاً للوقت والمالي والجهد، ما يحول دون توفره لعدد كبير من يحتاجونه.

توظيف تقنيات معالجة اللغة الطبيعية لتحسين حياة المصابين بحبسة بروكا مجال بحث حظي باهتمام العديد من الباحثين في الأعوام الأخيرة.

في مشروع التخرج هذا، نهتم باستعمال الترجمة الآلية والتعرف الآلي على الكلام لتأدية جزء من علاج النطق لحبسة بروكا أوتوماتيكياً. من أجل ذلك، نعرض دراسة بيليوغرافية نعرف فيها بحبسة بروكا أسباباً ونتائج، ثم نتطرق لعيوب العلاجات المعتادة. وللأعمال التي سبق إنجازها في مجال الترجمة الآلية والتعرف الآلي على الكلام.

نأتي بعدها إلى تصميم نظام لتصحيح الكلام المحتبس باللغة الفرنسية يجمع بين نموذجين، أحدهما للتعرف الآلي على الكلام والآخر للترجمة الآلية ونعرض إنجاز هذا الأخير. نختم أخيراً بعرض نتائج هذا العمل متمثلة في مجموعة بيانات للتعرف الآلي على الكلام وننوضح للترجمة الآلية تقييمه بمقاييس BLEU يساوي 79.61% .

الكلمات المفتاحية – حبسة بروكا، تعلم الآلة، معالجة اللغة الطبيعية، ترجمة آلية، تعرف آلي على الكلام، شبكة عصبية غير ترتيبية.

Table des matières

Page de garde

Résumé i

Abstract ii

مُلْكُوم iii

Table des matières vi

Table des figures ix

Algorithmes et extraits de code x

Sigles et abréviations xi

Introduction générale 1

I État de l'art 5

1 Notions générales 6

 1.1 Aphasicie de Broca 6

 1.2 Traduction automatique 11

 1.3 Reconnaissance automatique de la parole 13

1.4	Conclusion	14
2	Apprentissage séquence-à-séquence	16
2.1	Énoncé du problème	16
2.2	Perceptrons multicouches	17
2.3	Architecture encodeur-décodeur	19
2.4	Réseaux de neurones récurrents	20
2.5	Réseau de neurones à convolutions	25
2.6	Transformateurs	28
2.7	Conclusion	32
3	Traduction automatique et reconnaissance automatique de la parole	34
3.1	Traduction automatique	34
3.2	Reconnaissance automatique de la parole	42
3.3	Conclusion	44
II	Contribution	45
4	Conception	46
4.1	Architecture générale de la solution	46
4.2	Reconnaissance automatique de la parole	47
4.3	Traduction automatique neuronale	49
4.4	Conclusion	60
5	Réalisation	61
5.1	Outils et technologies	61
5.2	Création des corpus	66

5.3	Création du modèle de traduction automatique	69
5.4	Interface utilisateur	74
5.5	Conclusion	75
6	Tests et résultats	76
6.1	Erreurs générées	76
6.2	Corpus créé	78
6.3	Entraînement du modèle	80
6.4	Réglage des hyper-paramètres	83
6.5	Entraînement avec les hyperparamètres optimaux	85
6.6	Conclusion	87
	Conclusion générale	88
	Bibliographie	90
A	Dépendances et bibliothèques	97

Table des figures

1.1	Cerveau de Victor Louis Leborgne avec la lésion encadrée.	7
1.2	Encéphale humain.	8
1.3	Division morphologique et fonctionnelle du cerveau.	8
1.4	Classification des syndromes aphasiques classiques.	9
1.5	Comparaison de la qualité de vie de communication chez les individus sains et ceux qui souffrent de l'aphasie de Broca.	11
1.6	Taxonomie des méthodes de traduction automatique.	12
1.7	Triangle de Vauquois.	13
1.8	Taxonomie des techniques d'ASR.	14
2.1	Architecture sous-jacente d'un MLP de profondeur 4.	17
2.2	Architecture encodeur-décodeur	20
2.3	RNN v.s FFN	20
2.4	Dépliement temporel d'un RNN sur une entrée de longueur 4.	22
2.5	Dépliement temporel d'un encodeur-décodeur récurrent.	22
2.6	Forme générale d'un RNN à portes.	23
2.7	Architecture interne d'un GRU	24
2.8	Architecture interne d'un LSTM	24
2.9	Couche convolutive unidimensionnelle.	26
2.10	Recurrent Continuous Translation Model.	27

2.11	Architecture de ByteNet.	27
2.12	Architecture de ConvS2S	27
2.13	L'architecture de transformeur.	29
2.14	Matrice d'encodage de la position pour $r = 10^4$ et $d = 512$	30
2.15	Schéma d'une couche attention multitête.	31
3.1	L'architecture de transformeur.	38
3.2	Exemple de décodage par beam search.	39
3.3	Architecture de GPT.	39
3.4	Architecture de BERT.	41
3.5	Architecture de BART	41
3.6	Affinement de BART pour la traduction.	42
3.7	Architecture de Wav2Vec.	43
3.8	Architecture de Whisper.	43
4.1	Architecture générale de la solution.	47
4.2	Déroulement de la partie ASR.	48
4.3	Déroulement de la partie NMT.	50
4.4	Organigramme de la création du corpus parallèle.	52
4.5	Les plongements lexicaux de quelques mots avec $d = 2$	54
4.6	Organigramme de la phase d'entraînement.	56
4.7	Entropie croisée d'un classifieur linéaire binaire.	57
5.1	Composantes de la bibliothèque <code>lightning</code>	64
5.2	Graphe de calcul du modèle définit dans la classe <code>Transformer</code> . .	72
5.3	Interface web de traduction.	74
6.1	Fréquences des catégories d'erreurs	76

6.2	Perplexité des phrases du corpus par rapport à différents modèles de langue.	79
6.3	Organigramme de la phase d’entraînement	81
6.4	Évolution des métriques au cours de l’entraînement.	82
6.5	Évolution des métriques au cours de l’entraînement.	83
6.6	Résultats de la recherche bayésienne des hyperparamètres.	84
6.7	Évolution du score BLEU au cours du réglage des hyperparamètres.	84
6.8	Importance des hyperparamètres et corrélation avec le score BLEU.	86
6.9	Évolution des métriques avec les hyperparamètres optimaux.	86

Table des matières

2.1	Passe d'un MLP	18
2.2	Passe d'un RNN	21
2.3	Encodeur-décodeur récurrent.	23
3.1	Byte pair encoding.	36
3.2	Score BLEU.	37
3.3	Décodage par beam search.	40
5.1	Génération des erreurs avec chatGPT.	67
5.2	Création du corpus parallèle synthétique.	68
5.3	Méthode d'initialisation d'un transformeur.	69
5.4	Creation de l'entraîneur.	71
5.5	Création et lancement d'une Sweep.	74
6.1	Génération des erreurs pour un mot	77
6.2	Calcul de la perplexité avec le modèle <code>gpt-fr-cased-base</code>	79

Sigles et abréviations

API	interface de programmation d'application
ASR	reconnaissance automatique de la parole
AVC	accident vasculaire cérébral
BART	bidirectional auto-regressive transformer
BERT	bidirectional encoder representations from transformers
BLEU	bilingual evaluation understudy
BPE	byte pair encoding
BPTT	rétro-propagation dans le temps
CLM	modélisation causale du langage
CNN	réseau de neurones à convolutions
DL	apprentissage profond
FFN	réseau de neurones feed-forward
GPT	generative pre-trained transformer
GRU	gated recurrent unit
IL	interlingue
LC	langage cible
LLM	grand modèle de langage
LS	langue source
LSTM	long short-term memory
ML	apprentissage automatique
MLM	modélisation masquée du langage
MLOps	Machine Learning Operations
MLP	perceptron multicouches
MT	traduction automatique
NLP	traitement automatique du langage
NMT	traduction automatique neuronale
NSP	prédiction de la prochaine phrase

RMBT traduction automatique à base de règle
RNN réseau de neurones récurrent

S2S séquence-à-séquence
SMT traduction automatique statistique

Introduction générale

L'accident vasculaire cérébral (AVC) est une condition médicale dans laquelle la circulation du sang dans une région du cerveau s'arrête brusquement (LAROUSSE, s. d.). La World Stroke Organization estime que plus de 101 millions personnes dans le monde sont les victimes d'un AVC. La même organisation indique que 12.2 millions AVC ont lieu chaque année. Ce nombre est susceptible d'augmenter avec le vieillissement de la population mondiale. Les complications d'un AVC peuvent être totales ou partielles, temporaires ou permanentes. Elles peuvent inclure la paralysie, l'amnésie et la difficulté à parler ou à comprendre la parole. Cette dernière complication est appelée *aphasie* (FEIGIN et al., 2022).

Contexte

L'aphasie est un trouble de communication qui complique près d'un tiers des cas d'AVC (FLOWERS et al., 2016). Il s'agit d'une perte partielle ou totale de la capacité de produire ou de comprendre le langage. Les cas d'aphasie sont classés en fonction de leur sévérité et des facultés affectées en *syndromes aphasiques*. L'aphasie de Broca est l'un de ces syndromes aphasiques (CHAPEY, 2008).

L'aphasie de Broca est une forme d'aphasie qui affecte la capacité de s'exprimer oralement ou par écrit. Elle résulte d'une lésion dans l'aire de Broca, une région du cerveau qui est responsable de la production de la parole. Les personnes qui souffrent d'une aphasie de Broca ont des difficultés à produire des mots, mais peuvent comprendre ce qui est dit (CHAPEY, 2008).

Il est possible de mitiger les effets de l'aphasie de Broca avec des traitements de réhabilitation. La majorité de ces traitements sont basés sur la rééducation de la parole. Cela nécessite plusieurs séances (généralement 1–3 par semaine) avec un orthophoniste. Le taux de succès de ces traitements après 18 mois est de 24% (da FONTOURA et al., 2012 ; LASKA et al., 2001 ; Z. LIU et al., 2021).

Problématique

Les difficultés de communication causées par l'aphasie en général et l'aphasie de Broca en particulier peuvent être très handicapantes pour la vie quotidienne. Les activités affectées incluent la communication avec les proches, la participation à des activités sociales,

l'exercice d'un emploi ou même la demande d'aide en cas d'urgence (HALLOWELL, 2017). Par conséquent, la qualité de vie des personnes atteintes de l'aphasie de Broca est significativement inférieure à celle des personnes en bonne santé (PALLAVI et al., 2018 ; ROSS & WERTZ, 2010).

Ces problèmes ont un impact négatif sur la santé mentale des personnes atteintes de l'aphasie de Broca. Elles sont plus susceptibles à l'isolation sociale, à la dépression et aux pensées suicidaires, des pensées dont la communication est inhibée par l'aphasie et l'isolation sociale (COSTANZA et al., 2021 ; MORRISON, 2016). Le résultat est plus qu'un doublement du taux de mortalité des personnes qui souffrent d'aphasie par rapport aux cas d'AVC sans aphasic (36% contre 16% dans les 18 mois suivant l'AVC) (LASKA et al., 2001).

De plus, la rééducation de la parole, le traitement de réhabilitation le plus courant et le plus efficace pour la plupart des syndromes aphasiques — dont l'aphasie de Broca —, est un processus long et coûteux et fatiguant pour les patients. Cela en fait un traitement inaccessible pour les personnes qui ont des difficultés économiques ou de mobilité. Or, les personnes âgées ont souvent de telles difficultés (JACOBS & ELLIS, 2021 ; Z. LIU et al., 2021).

Cette inaccessibilité du traitement de réhabilitation pour un groupe de personnes qui inclut une grande partie des personnes qui en ont besoin, est inacceptable vu les conséquences souvent catastrophiques de l'aphasie. Il est urgent de trouver et d'implémenter des solutions qui permettent de faciliter l'accès à la réhabilitation en réduisant le coût, le besoin du déplacement et la nécessité de supervision professionnelle.

Objectifs

L'application des techniques d'apprentissage automatique (ML, de l'anglais : machine learning) et du traitement automatique du langage (NLP, de l'anglais : natural language processing) à cette problématique est une piste de recherche qui commence à capturer l'attention des chercheurs (C. LI et al., 2022 ; MISRA et al., 2022 ; QIN et al., 2022 ; SMAÏLI et al., 2022).

L'objectif de ce travail est la création d'un système automatique de réhabilitation de la parole aphasique. La fonction de ce système est de corriger la parole de son utilisateur, c'est-à-dire qu'il prend en entrée la parole éventuellement erronée d'une personne aphasique et produit en sortie une version corrigé de cette parole (en parlant de système speech-to-speech). Pour le réaliser, nous nous intéressons particulièrement à la traduction automatique (MT, de l'anglais : machine translation) et à l'recognition automatique de la parole (ASR, de l'anglais : automatic speech recognition).

Organisation de ce mémoire

Ce document est la synthèse de notre travail de recherche sur la problématique présentée ci-dessus. Il est organisé en deux parties. La première est une étude bibliographique sur la littérature existante relative aux sujets abordés. La deuxième présente la contribution apportée par notre travail.

État de l'art

Dans cette partie, nous présentons les travaux qui ont été faits dans la direction que nous explorons. Elle est organisée en trois chapitres :

Chapitre 1 Notions générales : Dans ce chapitre, nous présentons en général les trois sujets qui sont au cœur de ce travail : l'aphasie de Broca, la MT et la ASR. Nous commençons par une présentation de l'aphasie de Broca dans laquelle nous décrivons ses symptômes, ses causes et ses conséquences. Nous passons ensuite à la réhabilitation de la parole où nous présentons son insuffisance actuelle par rapport à l'ampleur et la portée du problème. Nous terminons par introduire la MT et l'ASR en présentant leurs définitions et une taxonomie des approches existantes.

Chapitre 2 Apprentissage séquence-à-séquence : Ce chapitre sert à familiariser le lecteur avec l'apprentissage séquence-à-séquence (S2S), le cadre d'étude général dans lequel s'inscrivent la MT, l'ASR et la majorité des tâches de NLP. Nous commençons par énoncer le problème, puis nous présentons certaines des architectures neuronales qui ont été appliquées dans son cadre. Nous clôturons ce chapitre par une comparaison de ces architectures.

Chapitre 3 Traduction automatique et reconnaissance automatique de la parole : Ce chapitre part de l'étude générale du chapitre 1.4. Il détaille l'application du transformeur aux problèmes de MT et d'ASR. Nous y présentons les travaux les plus importants dans ces deux domaines.

Contribution

Dans cette partie, nous présentons notre contribution à la problématique de la réhabilitation de la parole aphasique. Elle est aussi organisée en trois chapitres :

Chapitre 4 Conception : Le premier chapitre de cette partie présente la conception de notre système. Nous commençons par présenter l'architecture générale de notre solution. Nous détaillons ensuite la partie ASR de cette architecture. Après cela, la partie MT est discutée. Pour chaque partie, nous décrivons la procédure d'acquisition de données. Pour la partie MT, la création du modèle est aussi présentée.

Chapitre 5 Réalisation : Ce chapitre est consacré à la discussion des choix techniques que nous avons faits. Nous présentons d'abord les outils, technologies et bibliothèques que nous avons utilisés. Nous détaillons ensuite l'implémentation de la collecte et organisation des données. Après cela, la création et l'entraînement du modèle MT sont présentés. Nous terminons par la présentation de l'interface utilisateur.

Chapitre 6 Tests et résultats : Le dernier chapitre de ce mémoire est une discussion des résultats obtenus (notamment dans la partie MT). D'abord, la qualité et les caractéristiques des données générées et du corpus créé sont discutées. Ensuite, nous présentons et commentons les résultats de l'entraînement. Finalement, nous terminons avec les résultats du réglage des hyperparamètres.

Première partie

État de l'art

Chapitre 1

Notions générales

Dans ce chapitre, nous traçons les grandes lignes de notre étude. Nous commençons par introduire les détails du problème principal que nous abordons, à savoir l'aphasie de Broca. Ensuite, nous présentons deux axes de recherche qui nous semblent pertinents pour résoudre ce problème. Il s'agit respectivement de la MT et de l'ASR. Dans les deux chapitres suivants, nous développons en détails ces deux approches en explorant la littérature scientifique sur ces sujets.

1.1 Aphasie de Broca

L'aphasie ; emprunté au Grec ancien $\alpha\phiασία$ qui veut dire “mutisme”, est un trouble de communication d'origine neurologique (LAROUSSE, s. d.). Elle affecte la capacité à comprendre le langage, s'y exprimer ou les deux. L'aphasie n'est pas causée par un trouble moteur, sensoriel, psychique ou intellectuel (CHAPEY, 2008). Sa cause principale est un AVC, mais elle peut également être le résultat d'une infection ou tumeur cérébrale, un traumatisme crânien, un trouble métabolique comme le diabète ou une maladie neurodégénérative comme l'Alzheimer (HALLOWELL, 2017).

1.1.1 Histoire et anatomie

Louis Victor Leborgne, né en 1809 à Moret-sur-Loing commence à perdre la capacité de parler à l'âge de 30 ans. Il est admis à l'hôpital de Bicêtre où il passera 21 ans pendant lesquels, il ne communiquera qu'en produisant le son “tan”, typiquement répété deux fois, si bien qu'on lui a donné le surnom “monsieur Tan tan” (MOHAMMED et al., 2018).

Le 11 avril 1861, monsieur Leborgne est examiné par Dr. Pierre Paul Broca pour une gangrène dans son pied droit. Dr. Broca s'intéresse au trouble linguistique dont souffre son patient (LORCH, 2011). Il fait l'observation que les facultés intellectuelles et motrices de monsieur Leborgne sont intactes, il en conclut qu'elles ne peuvent être à l'origine de son handicap. Dr. Broca donne le nom “aphémie” à ce type de situation (BROCA, 1861), il en écrit :

“Cette abolition de la parole, chez des individus qui ne sont ni paralysés ni idiots, constitue un symptôme assez singulier pour qu'il me paraisse utile de la désigner sous un nom spécial. Je lui donnerai donc le nom d'aphémie (α privatif; φημι, je parle, je prononce); car ce qui manque à ces malades, c'est seulement la faculté d'articuler les mots.”

— BROCA, 1861.

Dr. Broca prend ce constat comme confirmation de ce qu'il appelait “le principe de localisations cérébrales”. Il s'agit de l'idée que le cerveau fonctionne comme système à plusieurs composants plutôt qu'un monolithe et que les fonctions cognitives sont spatialement localisées (FODOR, 1983).

Quand monsieur Leborgne est décédé le 17 avril, Dr. Broca lui fait l'autopsie. En ouvrant le crâne, il observe une lésion dans le cortex inférieur gauche du lobe frontal (voir Figure 1.1). Il en déduit que (1) cette lésion était à l'origine de l'aphémie de monsieur Leborgne et que (2) la partie affectée du cerveau est responsable d'articuler des expressions dans le langage (BROCA, 1861 ; LORCH, 2011 ; MOHAMMED et al., 2018).

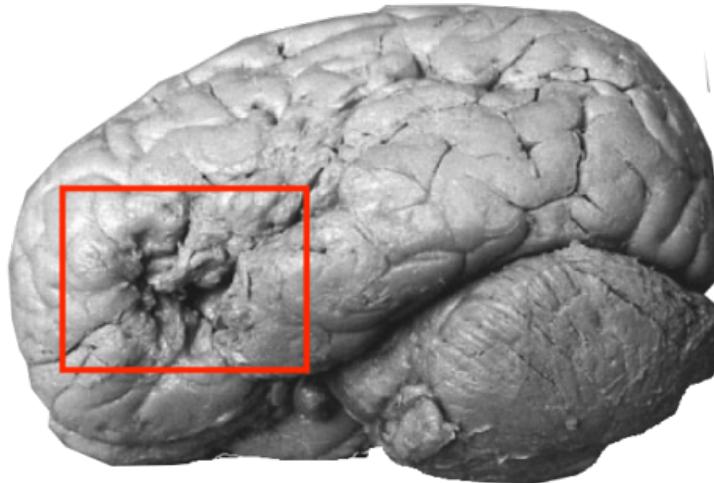


FIGURE 1.1 – Cerveau de Victor Louis Leborgne avec la lésion encadrée (LOPES, 2019).

Le trouble que Dr. Broca appelle “aphémie” est aujourd’hui connu sous le nom d’aphasie de Broca. Le cas de monsieur Leborgne est largement reconnu comme le premier cas enregistré d’aphasie en général et d’aphasie de Broca en particulier (MOHAMMED et al., 2018).

La quête de comprendre l’aphasie en général, et celle de Broca en particulier, commence avec le cerveau. Celui des humains est le système le plus complexe connu (SCIENCES et al., 1992). Avec le cervelet et le tronc cérébral, il forme l’encéphale (voir Figure 1.2). Le cerveau se charge du traitement des flux nerveux sensoriels et moteurs. Il est aussi le siège des hautes fonctions cognitives comme l’inférence logique, l’émotion et — crucialement pour notre étude — le traitement du langage (FODOR, 1983).

Le cerveau est composé de deux hémisphères ; chacun desquels se divise en lobes : frontal, temporal, pariétal et occipital (voir Figure 1.3a). La surface du cerveau s’appelle

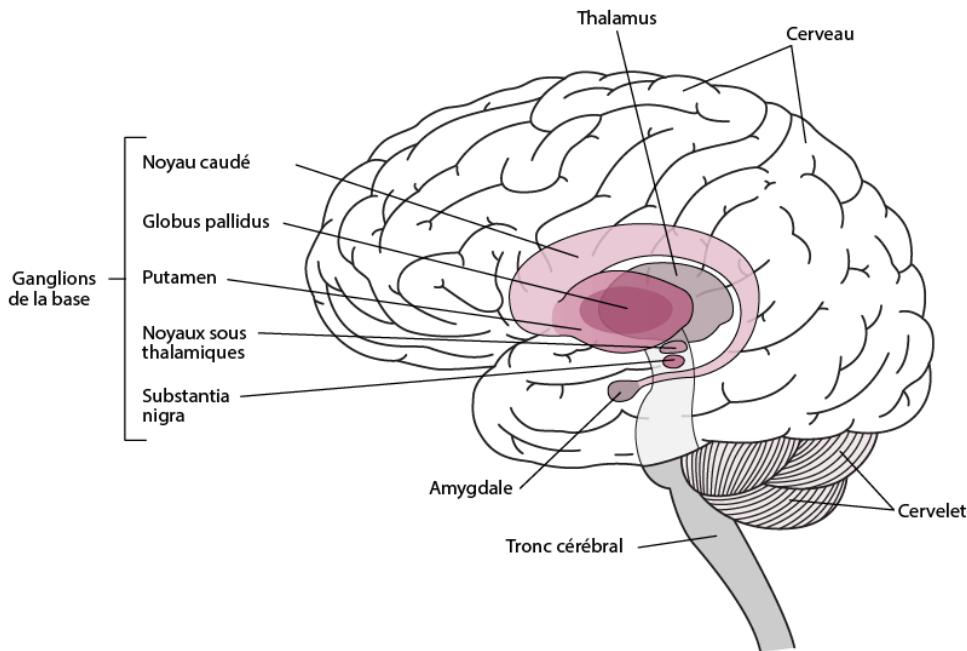
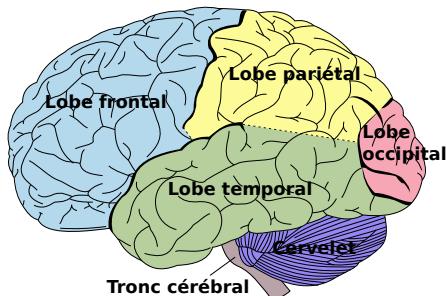
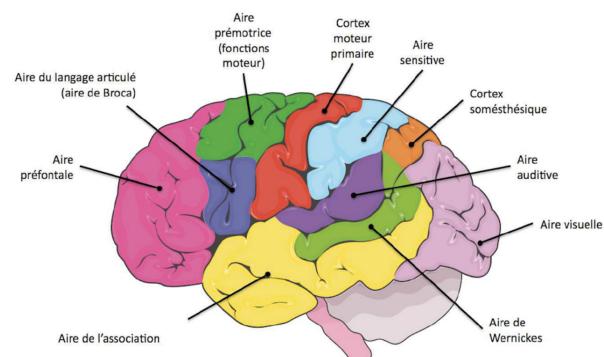


FIGURE 1.2 – Encéphale humain (« Noyaux gris centraux », s. d.).

le “cortex cérébral”. Il présente plusieurs circonvolutions qui augmentent considérablement sa surface. Le cortex cérébral est divisé en régions fonctionnelles que nous appelons “aires” (voir Figure 1.3b). Le travail de Dr. Broca sur le cas de M. Leborgne sont largement reconnus comme l’origine de cette division (FODOR, 1983).



(a) Lobes du cerveau (ART, 2013)



(b) Aires du cortex cérébral (JDIFOOL, 2006)

FIGURE 1.3 – Division morphologique et fonctionnelle du cerveau.

En particulier, la région du cerveau de M. Leborgne où Dr. Broca observe la lésion, correspond à l’aire qui porte son nom (aire de Broca). Ce dernier et celui de Wernicke jouent un rôle pivot pour l’aphasie (HALLOWELL, 2017).

1.1.2 Classification des syndromes aphasique

La définition que nous avons donnée de l'aphasie s'applique à une multitude de troubles (ou *syndromes*) dissimilaires en causes (région touchée du cerveau) et effets (conséquences pour la communication) (HALLOWELL, 2017). La table 1.1 et la figure 1.4 présentent une classification des syndromes aphasiques classiques. En particulier, notre sujet d'intérêt,

Syndrome Aphasique	Expressive / Réceptive	Localisation de la Lésion	Effet sur la Compréhension	Effet sur l'Expression
Aphasie de Wernicke	Réceptive	Aire de Wernicke (Brodmann 22 ¹)	Modéré à sévère	Modéré à sévère
Aphasie de Broca	Expressive	Aire de Broca (Brodmann 44, 45)	Léger à modéré	Modéré à sévère
Aphasie de Conduction	Les deux	Brodmann 40	Léger à modéré	Léger à modéré
Aphasie Globale	Expressive	Large, touche à plusieurs régions	Sévère	Sévère
Aphasie Transcorticale Sensorielle	Réceptive	Brodmann 37, 39	Modéré à sévère	Modéré à sévère
Aphasie Transcorticale Motrice	Expressive	Brodmann 6, 8–10, 46	Léger à modéré	Léger à modéré
Aphasie Transcorticale Mixte	Expressive	Lobe Frontal inférieur	Léger à modéré	Léger à modéré

TABLE 1.1 – Classification des syndromes aphasiques classiques (HALLOWELL, 2017)

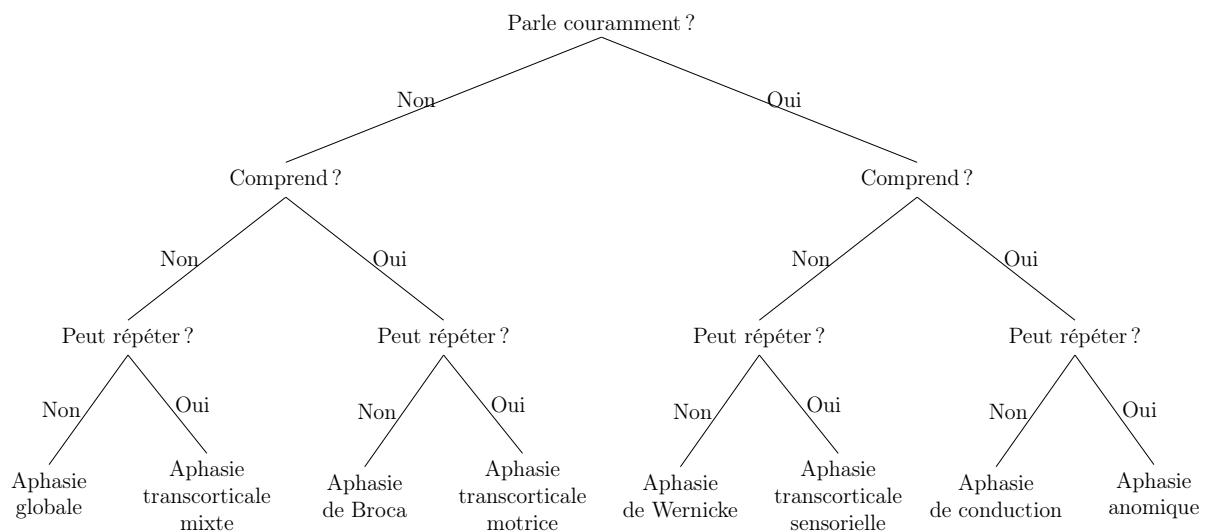


FIGURE 1.4 – Classification des syndromes aphasiques classiques (SREEDHARAN, 2018).

l'aphasie de Broca, est une aphasie expressive. Elle affecte surtout la production du langage. Sa compréhension est généralement préservée. De ce fait, les personnes atteintes de l'aphasie de Broca sont conscientes de leur handicap (CHAPEY, 2008).

1. BRODMANN, 2007.

1.1.3 Causes, prévalence et incidence

Les AVC sont la première cause d'aphasie (HALLOWELL, 2017). En effet, 30% des individus atteints d'un AVC développent une aphasic (FLOWERS et al., 2016). Parmi ces individus, 43% ont une aphasic de Broca (CNSA, 2015). Étant donné que 13 millions de personnes sont touchées par un AVC chaque année (SMAÏLI et al., 2022), cela représente environ 3.9 millions de cas d'aphasic par an (une incidence d'un peu moins de 0.05%). Inversement, 75% des cas d'aphasic sont causés par un AVC (CNSA, 2015).

Il est difficile d'estimer l'incidence et la prévalence globales de l'aphasic. Ceci est dû au manque de données dans la majorité des pays du monde. Cependant, le peu de données disponibles donnent des valeurs consistantes avec le 0.05% estimé ci-dessus. Selon l'association nationale de l'aphasic (« National Aphasia Association », s. d.), 2 millions d'Américains en souffrent, soit une prévalence de 0.6%. En France, ce chiffre est de l'ordre de 300 000 cas, 30 000 desquels sont nouveaux (CNSA, 2015). Ceci donne une prévalence de 0.44% et un taux d'incidence de 0.044%.

L'âge est un facteur de risque très important pour les AVC, il l'est donc également pour l'aphasic. En effet, l'âge moyen des individus français atteints de l'aphasic est 73 ans. 75% parmi eux sont âgés de plus de 65 ans dont 25% dépassent les 80 ans (CNSA, 2015).

1.1.4 Effet sur la qualité de vie

Il n'est pas surprenant que l'aphasic ait un impact négatif sur la qualité de vie des individus qui en souffrent. En effet, les individus atteints de l'aphasic ont une moins bonne qualité de vie que les individus en bonne santé selon les critères WHOQOL-BREF² et WPI³(voir Table 1.2).

Measure	Mean	Range	SD	Mean difference	95% C. I. ^a of difference	t(34)
<i>WHOQOL-BREF Transformed total</i>						
NBI participants	108.44	94.00–125.00	10.02			
Aphasic participants	96.11	68.00–124.00	14.05	12.23	4.07–20.60	3.03**
<i>WHOQOL-BREF overall QOL and general health rating</i>						
NBI participants	8.44	6.00–10.00	1.58			
Aphasic participants	7.22	4.00–10.00	1.52	1.22	0.17–2.27	2.37*
<i>PWI total</i>						
NBI participants	36.33	28.00–42.00	3.40			
Aphasic participants	31.72	22.00–41.00	5.90	4.61	1.35–2.87	2.87**

TABLE 1.2 – Comparaison de la qualité de vie chez les individus sains et ceux qui souffrent d'une aphasic (ROSS & WERTZ, 2010).

En particulier, l'aphasic de Broca a un effet négatif mesurable sur la qualité de vie des individus qu'elle affecte dans toute tâche qui nécessite la communication (PALLAVI et al., 2018). La Table 1.5 le montre bien pour les trois exemples d'activités sociales, confiances en soi et capacité à réaliser ses responsabilités quotidiennes.

2. <https://www.who.int/publications-detail-redirect/WHO-HIS-HSI-Rev.2012.03>

3. <https://www.acqol.com.au/instruments>

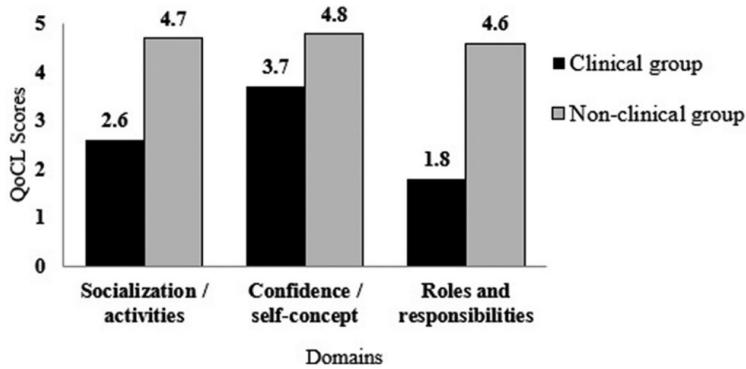


FIGURE 1.5 – Comparaison de la qualité de vie de communication chez les individus sains et ceux qui souffrent de l’aphasie de Broca (PALLAVI et al., 2018).

L’aphasie de Broca est également associée à une augmentation du risque d’hospitalisation et de décès (FLOWERS et al., 2016). Elle est aussi corrélée à un risque accru de maladies mentales, dépression et même tentative de suicide (COSTANZA et al., 2021 ; MORRISON, 2016). Ce dernier point est particulièrement grave en raison du fait que les personnes atteintes de l’aphasie de Broca sont généralement (1) âgées, (2) socialement isolées à cause de leur sentiment d’infériorité et (3) ont du mal à communiquer leur intention de suicide à cause de leur handicap. Il est donc urgent d’intervenir pour mitiger ces risques ainsi que la sévérité des effets de l’aphasie.

1.1.5 Traitement

Il n’existe pas de traitement général de l’aphasie de Broca. L’intervention thérapeutique est donc adaptée à chaque patient (ACHARYA & WROTON, 2022). Un point commun à la plupart des thérapies, est l’utilisation d’exercices orthophoniques pour la rééducation de la parole. Des exemples de tels exercices sont la répétition de phrases, la description d’images et la narration de récits (da FONTOURA et al., 2012). En dépit d’être la méthode la plus efficace dont on dispose, cette approche est très chronophage étant donné que la majorité des patients ne bénéficient que de 1–3 séances par semaine (da FONTOURA et al., 2012). Elle est également coûteuse, avec un coût moyen de plusieurs milliers de dollars par patient (JACOBS & ELLIS, 2021 ; Z. LIU et al., 2021), ce qui la rend très inaccessible. Ces lacunes sont inacceptables dans le contexte des effets dévastateurs de l’aphasie de Broca que nous avons décrits précédemment.

1.2 Traduction automatique

Nous avons établi que les difficultés d’accès aux traitements pour l’aphasie de Broca constituent un risque inadmissible en vue de ses effets potentiellement catastrophiques. Il est donc urgent de trouver des solutions pour y remédier. Une solution automatique à base de MT semble être une piste intéressante à explorer. Elle présente la possibilité de corriger les erreurs d’un individu atteint de l’aphasie de Broca sans avoir besoin d’un orthophoniste.

1.2.1 Généralités

La MT est une branche du NLP. Elle étudie l'utilisation des systèmes informatiques pour traduire le texte ou la parole d'une langue (appelée source) vers une autre (appelée cible) (CHAN, 2015). Dans cette section, nous introduisons la MT du texte pour donner un point de référence aux discussions des chapitres suivants.

1.2.2 Classification

Plusieurs méthodes de MT sont présentées dans la littérature. Ces méthodes peuvent être classées selon les outils mathématiques qu'elles utilisent. On distingue ainsi trois grandes familles de méthodes de MT (YANG et al., 2020) :

1. Des méthodes basées sur des connaissances linguistiques (règles de traduction).
2. Des méthodes basées sur les statistiques.
3. Des méthodes basées sur les réseaux de neurones.

On les appelle respectivement traduction automatique à base de règle (RMBT, de l'anglais : rule-based machine translation), traduction automatique statistique (SMT, de l'anglais : statistical machine translation), traduction automatique neuronale (NMT, de l'anglais : neural machine translation).

À l'intérieur de ces familles, les méthodes peuvent être distinguées en sous-familles. Ceci donne lieu à la hiérarchie représentée par la Figure 1.6

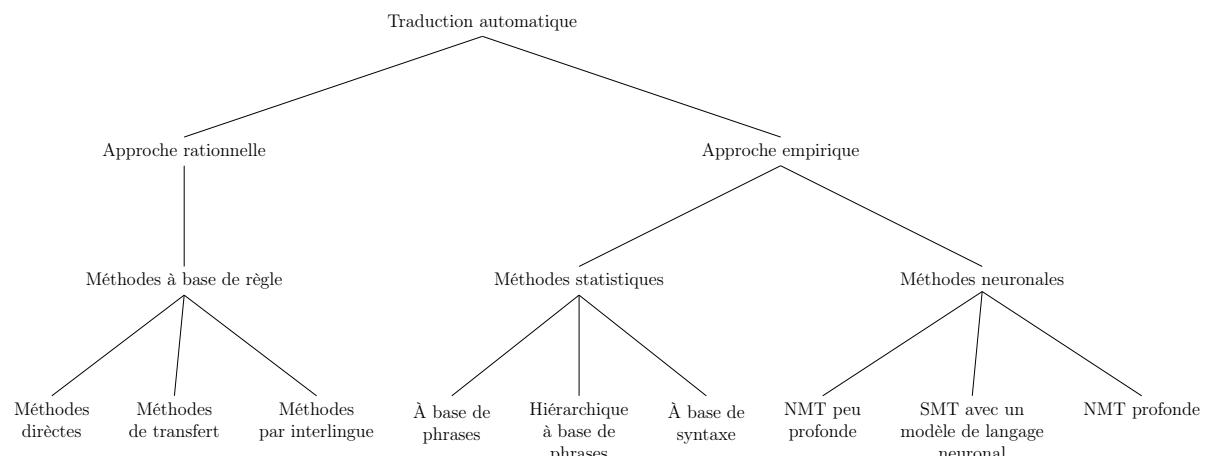


FIGURE 1.6 – Taxonomie des méthodes de traduction automatique (COSTA-JUSSÀ et al., 2016 ; YANG et al., 2020).

Une autre classification récurrente dans la littérature est celle du triangle de Vauquois (voir Figure 1.7). Elle se base sur l'utilisation ou non de représentations intermédiaires des langues. Si la phrase traduite est construite directement à partir de la phrase à traduire, il s'agit de traduction *directe*. Ce paradigme présente deux inconvénients majeurs. Le premier — et le plus évident — est la difficulté de passer directement d'une langue à une autre. Le deuxième est la scalabilité : pour maintenir la traduction entre n langues, $\frac{n(n-1)}{2}$ couples de fonctions de traduction sont nécessaires (CHAN, 2015).

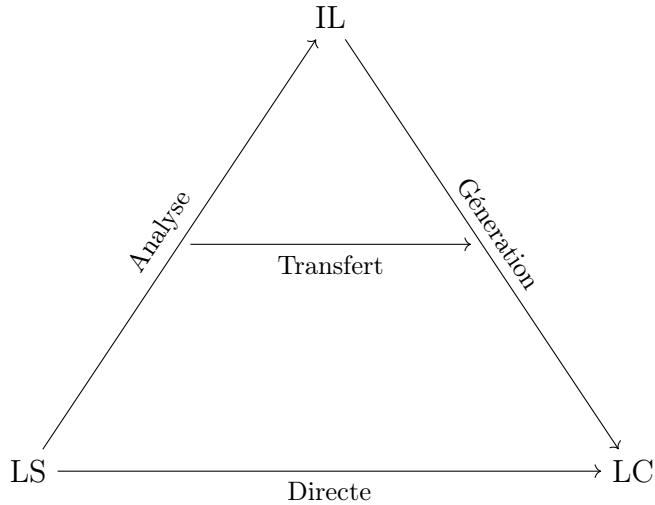


FIGURE 1.7 – Triangle de Vauquois (COSTA-JUSSÀ et al., 2016).

Les méthodes indirectes se distinguent en méthodes “par transfert”, qui utilisent une représentation intermédiaire dépendant de la langue (par exemple, arbre de syntaxe) et en méthodes “par interlingue”, qui utilisent la même représentation intermédiaire pour toutes les langues. Ce dernier paradigme résout le problème de scalabilité (seulement n couples de traductions sont nécessaires) au coût de construire une interlingue assez riche à représenter toute phrase dans toute langue (CHAN, 2015).

Dans ce travail, nous nous intéressons principalement aux sous arbre droit de la Figure 1.6. En effet, la majorité de notre investigation porte sur son dernier élément (la NMT profonde). Ceci est motivé par l’énorme succès dont l’apprentissage profond (DL, de l’anglais : deep learning) a fait preuve dans la dernière décennie (SEBASTIAN & MIRJALILI, 2017).

1.3 Reconnaissance automatique de la parole

L’ASR est une autre branche du NLP qui peut faciliter l’informatisation du traitement de l’aphasie de Broca. Elle permet de convertir la parole du patient en texte qui peut être corrigé avec la MT. Dans cette section, nous la présentons brièvement.

1.3.1 Généralités

La reconnaissance automatique de la parole (ou encore la transcription automatique) est une technique qui permet de convertir un signal audio en texte.

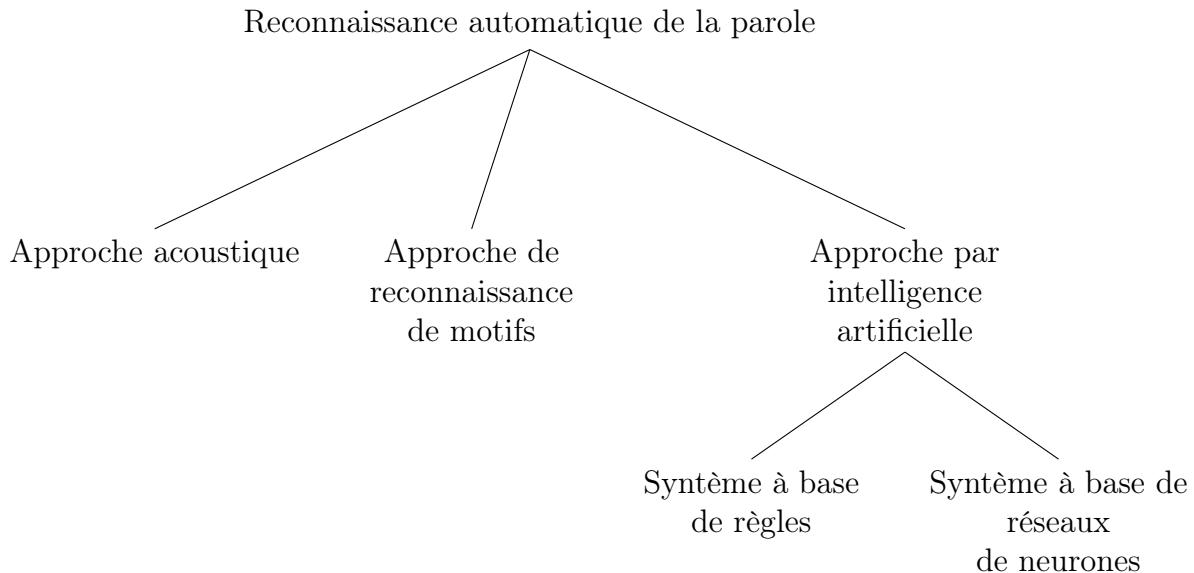


FIGURE 1.8 – Taxonomie des techniques d’ASR (VOLNY et al., 2012).

1.3.2 Classification

Les méthodes d’ASR peuvent être divisées en méthodes basées sur l’acoustique, méthodes basées sur la reconnaissance de motifs et méthodes basées sur l’intelligence artificielle (voir Figure 1.8).

Comme pour la MT, notre intérêt porte principalement sur les méthodes à base d’intelligence artificielle. Plus spécifiquement sur les méthodes qui utilisent le DL.

1.4 Conclusion

Dans ce chapitre, nous avons fourni une brève introduction aux sujets abordés dans ce mémoire. Nous avons introduit le problème central, ses spécificités, son contexte historique et scientifique et des chemins possibles vers une solution.

Nous avons commencé par présenter au lecteur les troubles linguistiques qui sont l’aphasie en général et l’aphasie de Broca en particulier. Nous l’avons abordé sous différents angles, notamment celui de l’histoire et de la neuroanatomie. Nous l’avons également mis dans le contexte des autres syndromes aphasiques. Une discussion des causes de l’aphasie de Broca a également été menée. Des statistiques sur l’incidence et la prévalence de l’aphasie de Broca, ainsi que des études sur ses conséquences ont été présentées pour aider le lecteur à apprécier l’ampleur du problème et sa gravité.

Nous avons terminé notre introduction à l’aphasie de Broca par une discussion des traitements typiquement déployés contre elle. En particulier, nous avons discuté le manque d’accessibilité et de scalabilité du traitement orthophonique. Un problème que nous avons jugé très grave à cause des conséquences dévastatrices de l’aphasie de Broca.

Après cela, nous avons introduit les techniques de NLP que nous jugeons pertinentes pour la résolution de notre problème : la MT et l'ASR. Pour la MT, nous avons commencé par une description de la tâche. Ensuite, nous avons avancé les classifications de ses méthodes qu'on trouve dans la littérature. Notre choix d'exploration s'est porté sur la classe que nous avons trouvée la plus prometteuse : la NMT.

Pour l'ASR, nous avons repris la même démarche. Après avoir décrit la tâche, nous avons illustré une classification des méthodes. Nous avons encore une fois choisi de nous concentrer sur les techniques à base de DL.

Dans le reste de ce document, nous étudions plus en détail la NMT et l'ASR. Le chapitre suivant aborde l'apprentissage S2S, le problème général dont ces deux tâches sont des sous-problèmes. Celui qui suit rentre dans le détail de l'application de la NMT et de l'ASR.

Chapitre 2

Apprentissage séquence-à-séquence

Les modèles S2S sont une famille d’algorithmes d’ML dont l’entrée et la sortie sont des séquences (MARTINS, 2018). Plusieurs tâches de ML, notamment en NLP, peuvent être formulées comme tâches d’apprentissage S2S. Parmi ces tâches, nous citons : la création de chatbots, la réponse aux questions et — crucialement pour ce travail — la MT et l’ASR (FATHI, 2021).

Dans ce chapitre, nous commençons par formuler le problème de modélisation de séquences. Ensuite, nous présentons les architectures neuronales les plus utilisées pour cette tâche. Enfin, nous concluons avec une étude comparative de celles-ci.

2.1 Énoncé du problème

Formellement, le problème de modélisation S2S est celui de calculer une fonction partielle $f : X^* \rightarrow Y^*$, où :

- X est un ensemble dit d’entrées.
- Y est un ensemble dit de sorties.
- Pour un ensemble A , $A^* = \bigcup_{n \in \mathbb{N}} A^n$ est l’ensemble de suites de longueur finie d’éléments de A .

f prend donc en entrée un $x = (x_1, x_2, \dots, x_n) \in X^n$ et renvoie un $y = (y_1, y_2, \dots, y_m) \in Y^m$. Dans le cas général, $n \neq m$ et aucune hypothèse d’alignement n’est admise. On prend souvent $X = \mathbb{R}^{d_e}$ et $Y = \mathbb{R}^{d_s}$ avec $d_e, d_s \in \mathbb{N}$. Dans ce cas, $x \in \mathbb{R}^{d_e n}$ et $y \in \mathbb{R}^{d_s m}$. Les indices peuvent représenter une succession temporelle ou un ordre plus abstrait comme celui des mots dans une phrase (MARTINS, 2018).

La majorité des outils mathématiques historiquement utilisés pour ce problème viennent de la théorie du traitement de signal numérique. Cependant, l’approche actuellement dominante, et celle qui a fait preuve de plus de succès, est de les combiner avec les réseaux de neurones.

2.2 Perceptrons multicouches

Les réseaux de neurones profonds sont parmi les modèles les plus expressifs en ML. Leur succès pratique est incomparable à ceux des modèles qui les ont précédés, que ce soit en termes de qualité des résultats ou de variété de domaines d'application (SEBASTIAN & MIRJALILI, 2017). De plus, grâce aux théorèmes dits d'approximation universelle, ce succès empirique est formellement assuré (CALIN, 2020).

2.2.1 Généralités

Dans cette section, nous introduisons les perceptrons multicouches (MLP, de l'anglais : multi-layer perceptorn), l'architecture neuronale la plus simple et la plus utilisée. Il s'agit d'une simple composition de couches affines avec des activations non affines (voir Définition 1).

Définition 1 (MLP).

Soient $k, w_0, w_1, \dots, w_{k+1} \in \mathbb{N}$, un réseau de neurones feed-forward de profondeur $k+1$, à w_0 entrées et w_{k+1} sorties, est défini par une fonction :

$$\begin{cases} \mathbb{R}^{w_0} \rightarrow \mathbb{R}^{w_{k+1}} \\ x \mapsto \varphi_{k+1} \circ A_{k+1} \circ \varphi_k \circ A_k \circ \dots \circ \varphi_1 \circ A_1(x) \end{cases} \quad (2.1)$$

où les A_i sont des fonctions affines $\mathbb{R}^{w_{i-1}} \rightarrow \mathbb{R}^{w_i}$ et les φ_i sont des fonctions quelconques, typiquement non affines $\mathbb{R}^{w_i} \rightarrow \mathbb{R}^{w_i}$, dites *d'activations*. La fonction $\varphi_i \circ A_i$ est appelée la i^{eme} couche du réseau (MUKHERJEE, 2021).

À un tel réseau de neurones, on peut associer un graphe orienté acyclique qu'on appelle son “architecture sous-jacente” (KEARNS & VAZIRANI, 1994). La Figure 2.1 illustre l'architecture d'un MLP de profondeur 4 avec $(w_0, w_1, w_2, w_3, w_4) = (4, 5, 7, 5, 4)$.

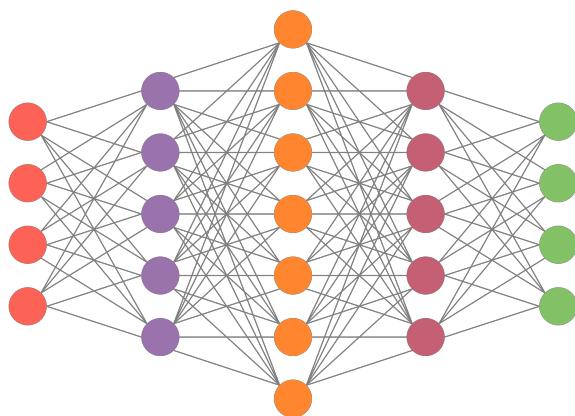


FIGURE 2.1 – Architecture sous-jacente d'un MLP de profondeur 4.

Deux MLP peuvent avoir la même architecture sous-jacente, en effet, cette dernière ne dépend que des dimensions de leurs couches respectives. De ce fait, une méthode

de trouver, pour une architecture et une fonction cible données, le meilleur MLP est nécessaire (MUKHERJEE, 2021).

Pour ce faire, nous exploitons le fait que les A_i sont des applications affines sur des espaces de dimensions finies. Nous pouvons donc les écrire comme combinaisons de produits matriciels et de translations. Le problème se réduit donc à régler¹ les paramètres des matrices en question. Cela nécessite une façon de quantifier la qualité d'approximation d'une fonction f par une autre \hat{f} . L'analyse fonctionnelle nous en donne plusieurs. Les équations (2.2) et (2.3) sont deux exemples récurrents de fonctions dites *de perte*.

$$L_1(f, \hat{f}) = \|f - \hat{f}\|_1 = \int |f - \hat{f}| \quad (2.2)$$

$$L_2(f, \hat{f}) = \|f - \hat{f}\|_2^2 = \int |f - \hat{f}|^2 \quad (2.3)$$

Ayant fixé une fonction de perte L , l'entraînement revient à un problème d'optimisation comme représenté par l'équation 2.4.

$$f^* = \underset{\hat{f}}{\operatorname{argmin}} L(f, \hat{f}) \quad (2.4)$$

Dans le cas particulier où L est différentiable, l'algorithme du gradient peut être utilisé pour trouver un minimum local. Les gradients sont calculés en utilisant une méthode de dérivation automatique comme la rétro-propagation.

2.2.2 Application à la modélisation de séquence

Les réseaux de neurones opèrent sur des vecteurs. Afin de les utiliser dans le contexte de la modélisation S2S, il faut utiliser une représentation vectorielle des entrées. Une telle représentation s'appelle un *plongement* (embedding en anglais). Le plongement peut être appris ou prédéfini (SEBASTIAN & MIRJALILI, 2017).

Dans le cas des MLP, la séquence d'entrée est d'abord décomposée en sous-séquences. Ensuite, les plongements de ces sous-séquences sont traités individuellement par le réseau de neurones, ce qui produit une séquence de vecteurs en sortie. (voir Algorithme 2.1).

```

1 def mlp_s2s(xs: Sequence, k: int) -> Sequence:
2     ys = [] # Initialiser la sortie
3     for x in blocks(xs, k): # Parcourir les blocs
4         y = mlp(x) # Traiter chaque bloc
5         ys.append(y) # Concatener les resultats
6     return ys

```

Extrait de code 2.1 – Passe d'un MLP

1. En ML, le terme “entraîner” est plutôt utilisé.

2.2.3 Avantages et inconvénients

Les MLP présentent deux avantages par rapport aux architectures discutées dans le reste de ce chapitre. Le premier est leur simplicité. Elle les rend plus simples à comprendre et à implémenter. Le deuxième est le fait qu'ils traitent indépendamment les sous-séquences. Cela rend très facile la tâche de les paralléliser, ce qui accélère considérablement leur entraînement.

Cependant, ce dernier point pose un grand problème. Comme ils traitent indépendamment les blocs de la séquence, les MLP ne peuvent pas modéliser les dépendances inter-blocs. Par conséquent, leur performance sur les séquences composées de plusieurs blocs est très médiocre. La solution de ce problème est d'augmenter la taille du bloc (et donc aussi la dimension d'entrée). Les MLP dépendent également d'une hypothèse d'alignement par blocs entre les deux séquences, une hypothèse invalide selon la Section 2.1.

2.3 Architecture encodeur–décodeur

Les lacunes des MLP sont en grande partie le résultat du traitement séparé des parties des séquences. Dans la production de l'élément (ou bloc) courant de la sortie, un MLP se base uniquement sur l'élément (ou bloc) correspondant de l'entrée. L'équation 2.5 l'illustre pour une entrée $x = (x_1, x_2, \dots, x_n)$ et une sortie $y = (y_1, y_2, \dots, y_m)$.

$$y_j = f(x_i, x_{i+1}, \dots, x_{i+\ell}) \quad 1 \leq j \leq m \quad (2.5)$$

En plus de l'hypothèse implicite de l'existence d'une telle correspondance, cela suppose que les éléments d'une séquence sont complètement indépendants les uns des autres. Cette dernière hypothèse n'est presque jamais vérifiée.

Une façon naturelle de combler ces lacunes est d'abandonner le traitement par bloc de l'entrée. Tout élément de la séquence de sortie est considéré comme fonction de la séquence d'entrée entière. L'équation 2.6 montre cette approche sur l'exemple précédent (STAHLBERG, 2020).

$$y_j = f(x) = f(x_1, x_2, \dots, x_n) \quad 1 \leq j \leq m \quad (2.6)$$

L'architecture encodeur–décodeur y parvient en combinant deux modules : un encodeur et un décodeur. L'encodeur consomme la suite donnée en entrée et produit un vecteur de dimension fixe qui la représente². Le décodeur consomme ce vecteur et produit la sortie (voir Figure 2.2). L'équation 2.7 le montre sur le même exemple.

$$\begin{aligned} c &= \text{encoder}(x) \\ y &= \text{decoder}(c) \end{aligned} \quad (2.7)$$

L'équation 2.7 ne dépend ni du fonctionnement interne de l'encodeur ni de celui du décodeur. Les deux modules peuvent avoir deux architectures quelconques, qui peuvent ou non être les mêmes (YANG et al., 2020). Dans le reste de ce chapitre, nous examinons les architectures communes en apprentissage S2S.

2. Ce vecteur est appelé un vecteur de contexte, vecteur de pensée ou encore un encodage.

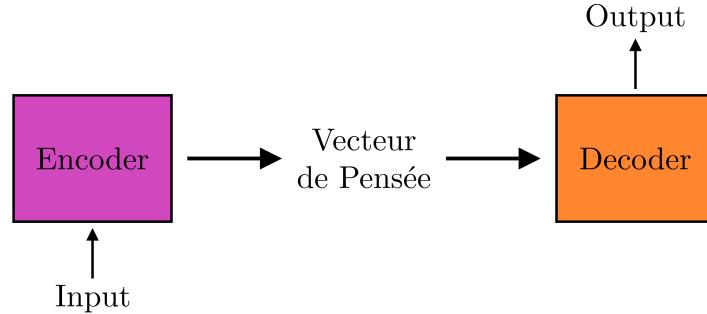


FIGURE 2.2 – Architecture encodeur-décodeur

2.4 Réseaux de neurones récurrents

L'un des principaux défauts que nous avons observés avec les MLP et qui nous ont poussés à introduire l'architecture encodeur-décodeur, est leur incapacité de représenter la dépendance entre les éléments d'une séquence. Une incapacité qui résulte de leur traitement indépendant des éléments.

Les réseaux de neurones récurrents (RNN, de l'anglais : recurrent neural network) tentent de résoudre ce problème en utilisant un état interne persistant. Chaque élément de la séquence modifie cet état lors de son traitement. Cela permet aux premiers éléments d'affecter le traitement des éléments qui les suivent, permettant ainsi à l'information de se propager vers le futur, ce qui donne lieu à une *mémoire*. Pour implémenter ce type

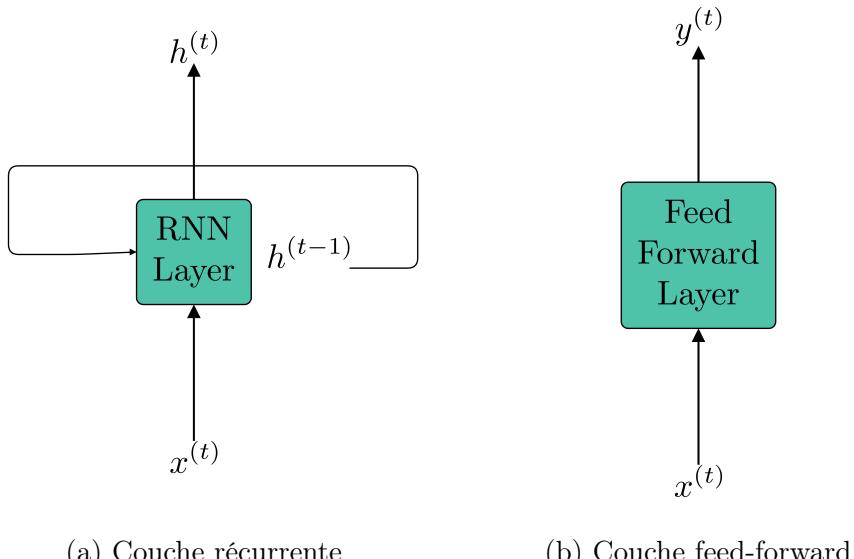


FIGURE 2.3 – RNN v.s FFN

de comportement, l'état d'un RNN est décalé d'une unité et réinjecté dans l'entrée (voir Figure 2.3a et Algorithme 2.2). Cela est très différent des MLP qui n'ont pas de boucle de rétroaction, il s'agit de réseaux de neurones feed-forward (FFN, de l'anglais : feed-forward

```

1 def rnn_pass(xs: Sequence, h=0) -> Vector:
2     for x in xs: # pour chaque element de la sequence
3         # traitement de l'element + mise a jour de l'état
4         h = process(x, h)
5     return h # retourne l'état final

```

Extrait de code 2.2 – Passe d'un RNN

network) (voir Figure 2.3b). Par conséquent, ils n'ont pas d'état ni de mémoire (FATHI, 2021).

2.4.1 Réseaux de neurones récurrents simples

Le RNN le plus simple à imaginer se réduit à une combinaison affine de l'état passé et de l'entrée. Il permet à l'information sur le passé de se propager sans contrôle particulier. Mathématiquement, une couche d'un tel RNN prend la forme suivante

$$h^{(t)} = \varphi(Uh^{(t-1)} + Wx^{(t)} + b) \quad 1 \leq t \leq n \quad (2.8)$$

où t est le temps³, $x^{(t)}, h^{(t)}$ sont respectivement l'entrée et la sortie à l'instant t , n est la longueur de la séquence et φ est la fonction d'activation (FATHI, 2021). Dans le cas où φ est l'identité, la transformée en z de l'équation 2.8 est donnée par :

$$H(z) = z(zI - U)^{-1}(WX(Z) + b) \quad (2.9)$$

il s'agit donc d'un système à réponse impulsionale infinie (FATHI, 2021).

Dépliement temporel et encodeur–décodeur récurrent

Le traitement d'une séquence x par un RNN (\mathcal{R}), est équivalent à son traitement par un FFN (\mathcal{F}) dont la profondeur est égale à la longueur de x ⁴. \mathcal{F} est donc appelé *dépliement temporel* de \mathcal{R} pour x . La Figure 2.4 montre le dépliement temporel de la Figure 2.3a pour une séquence de longueur 4 (LECUN et al., 2015).

Chaque état caché $h^{(i)}$ contient de l'information sur tous les $x^{(j)}$, $j \leq i$. En particulier, $h^{(n)}$ contient de l'information sur toute la séquence x . Un encodeur récurrent peut donc retourner son dernier état caché comme vecteur d'encodage (YANG et al., 2020).

De sa part, un décodeur récurrent peut conditionner sur l'encodage et passer ses états cachés à une couche supplémentaire qui les interprète comme plongements des éléments de la sortie y (FATHI, 2021). Un tel décodeur n'a pas besoin d'entrée séquentielle (voir Figure 2.5 et Algorithme 2.3).

3. Il peut être continu ou discret, réel ou abstrait.
4. Et dont toutes les couches sont identiques à \mathcal{R} .

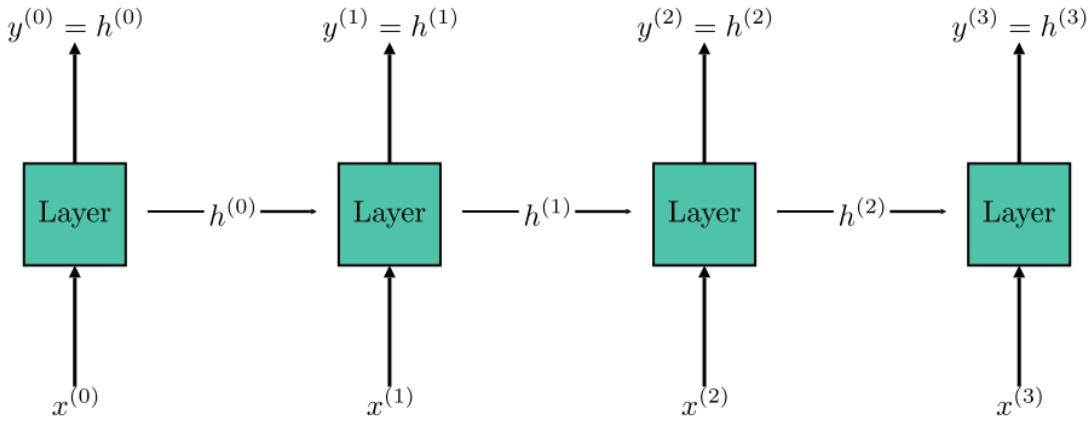


FIGURE 2.4 – Dépliement temporel d'un RNN sur une entrée de longueur 4.

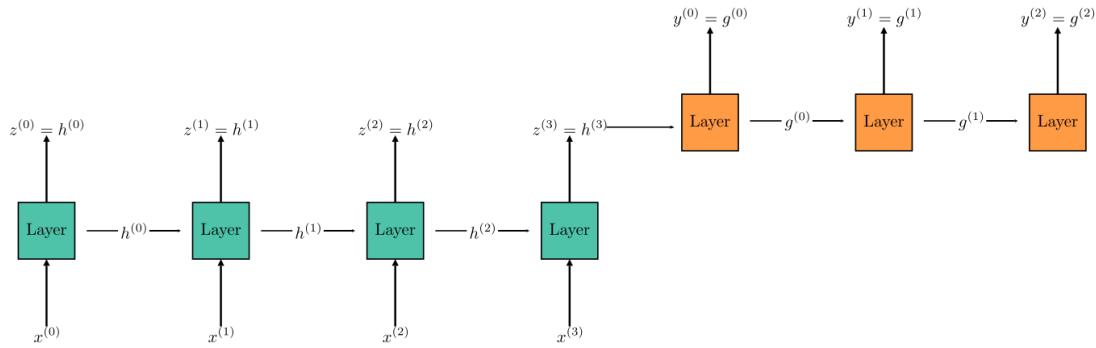


FIGURE 2.5 – Dépliement temporel d'un encodeur–décodeur récurrent sur une entrée de longueur 4 et une sortie de longueur 3.

Rétro-propagation dans le temps et mémoire à court terme

L'entraînement d'un RNN sur un exemple se fait en le dépliant, puis en entraînant le FFN qui résulte par rétro-propagation. L'algorithme résultant s'appelle la rétro-propagation dans le temps (BPTT, de l'anglais : back-propagation through time). Cela est problématique, car la taille de l'entrée n'est théoriquement pas bornée. Par conséquent, la profondeur effective d'un RNN ne l'est pas non plus (FATHI, 2021).

Or, dans l'entraînement d'un réseau de neurones trop profond, les modules des gradients peuvent atteindre des valeurs trop grandes ou trop petites. Il s'agit respectivement des problèmes de “l'explosion du gradient” et de “la disparition du gradient” (BASODI et al., 2020) Une conséquence de ce phénomène est que les RNN simples ont du mal à traiter les entrées pour lesquelles le dépliement temporel est profond, (i.e les longues entrées). Cette incapacité à modéliser les corrélations à long terme est appelée “mémoire à court terme” (BENGIO et al., 1994 ; INFORMATIK et al., 2003).

```

1 def rnn_s2s(xs: Sequence) -> Sequence:
2     encoder = RNN() # encodeur recurrent
3     decoder = RNN() # decodeur recurrent
4     ys = [] # sequence de sortie initialement vide
5     h = encoder(xs) # encode la sequence d'entree
6
7     while True:
8         # decoder sequentiellement
9         h = decoder(0, h)
10        ys.append(h)
11        if h == eos: # si on a genere le symbole de fin
12            # on arrete la generation
13            break
14
15    return ys

```

Extrait de code 2.3 – Encodeur-décodeur récurrent.

2.4.2 Réseaux de neurones à portes

Pour une grande partie des problèmes d'apprentissage S2S, les séquences peuvent être très longues. Le mémoire à court terme constitue donc un véritable obstacle pour l'utilisation des RNN simples en pratique. Une approche de le contourner qui a eu un énorme succès expérimental, est l'introduction d'un mécanisme de contrôle sur la boucle de rétroaction (voir Figure 2.6). Ce mécanisme est généralement implémenté avec des *portes*, des unités entraînables qui peuvent réguler le flux d'information dans la couche récurrente. On parle alors de *RNN à portes*. Dans cette section, nous explorons les deux variantes les plus utilisées de RNN à portes : le gated recurrent unit (GRU) et le long short-term memory (LSTM).

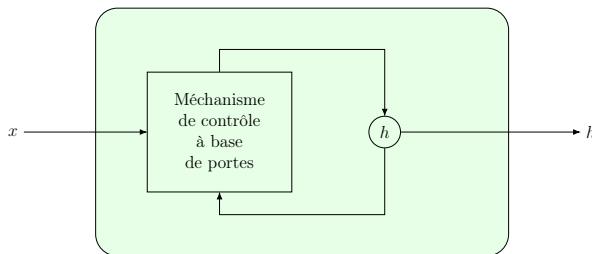


FIGURE 2.6 – Forme générale d'un RNN à portes.

Gated recurrent unit

Le GRU, introduit par (CHO et al., 2014), est une architecture récurrente à portes très simple (voir Figure 2.7). Elle utilise deux portes. La première est la porte de réinitialisation (r). Elle détermine le point auquel l'information sur le passé peut se propager (quand $r = 0$, pas de propagation et quand $r = 1$, propagation totale). Sa sortie s'appelle *l'état candidat* (\tilde{h}). La deuxième est la porte de mise à jour (z). Elle détermine les contributions respectives de l'état candidat et l'état passé (si $z = 0$, seul l'état candidat contribue et si $z = 1$, seul l'état courant contribue). Sa sortie est la sortie globale du GRU (CHO et al., 2014).

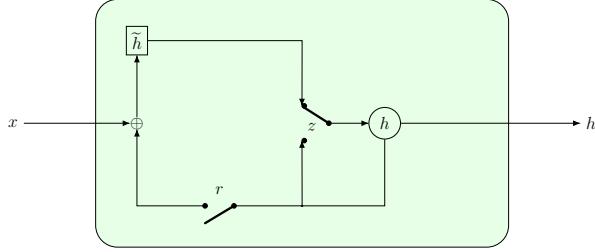


FIGURE 2.7 – Architecture interne d'un GRU (CHUNG et al., 2014)

Le fonctionnement des portes d'un GRU est simple. Leurs valeurs sont calculées à partir de l'entrée et de l'état courant par les équations 2.10 et 2.11, où σ est la fonction *sigmoïde*⁵ et \odot et le produit d'Hadamard⁶.

$$z^{(t)} = \sigma(W_z x^{(t)} + U_z h^{(t-1)} + b_z) \quad (2.10)$$

$$r^{(t)} = \sigma(W_r x^{(t)} + U_r h^{(t-1)} + b_r) \quad (2.11)$$

$$\tilde{h}^{(t)} = \varphi(W_h x^{(t)} + U_h (r^{(t)} \odot h^{(t-1)}) + b_h) \quad (2.12)$$

$$h^{(t)} = z^{(t)} \odot h^{(t-1)} + (1 - z^{(t)}) \odot \tilde{h}^{(t)} \quad (2.13)$$

L'état candidat est calculé à partir de l'entrée et l'état pondéré par la porte de réinitialisation par l'équation 2.12, où φ est la fonction d'activation. Finalement, l'état futur (la sortie) est la moyenne pondérée par z de l'état courant et l'état candidat (CHO et al., 2014). Notons que le GRU devient un RNN simple si les portes de réinitialisation et de mise à jour sont respectivement fixées à 1 et 0 (FATHI, 2021).

Long short-term memory

Il s'agit de l'une des premières architectures récurrentes à portes (CHUNG et al., 2014). Elle a été introduite par (HOCHREITER & SCHMIDHUBER, 1997). Un LSTM implémente trois portes : une porte d'entrée (i), une porte d'oubli (f) et une porte de sortie (o) (voir Figure 2.8).

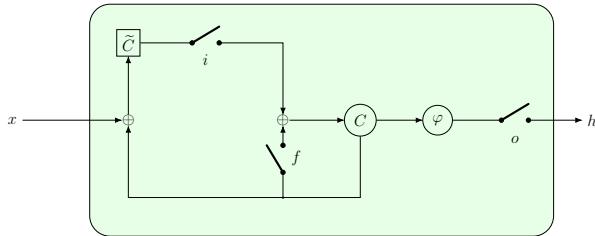


FIGURE 2.8 – Architecture interne d'un LSTM (CHUNG et al., 2014)

Le fonctionnement des portes est similaire à celui du GRU. Les équations 2.14–2.19 le montrent en détails⁷ (HOCHREITER & SCHMIDHUBER, 1997).

5. $\sigma : \mathbb{R} \rightarrow \mathbb{R}, x \mapsto \frac{1}{1+e^{-x}}$

6. Pour $u, v \in \mathbb{R}^n$, $u \odot v \in \mathbb{R}^n$ et $(u \odot v)_i = u_i v_i$

7. $\tanh : \mathbb{R} \rightarrow \mathbb{R}, x \mapsto \frac{e^x - e^{-x}}{e^x + e^{-x}}$

$$f^{(t)} = \sigma(W_f x^{(t)} + U_f h^{(t-1)} + b_f) \quad (2.14)$$

$$i^{(t)} = \sigma(W_i x^{(t)} + U_i h^{(t-1)} + b_i) \quad (2.15)$$

$$o^{(t)} = \sigma(W_o x^{(t)} + U_o h^{(t-1)} + b_o) \quad (2.16)$$

$$\tilde{c}^{(t)} = \tanh(W_c x^{(t)} + U_c h^{(t-1)} + b_c) \quad (2.17)$$

$$c^{(t)} = f^{(t)} \odot c^{(t-1)} + i^{(t)} \odot \tilde{c}^{(t)} \quad (2.18)$$

$$h^{(t)} = o^{(t)} \odot \varphi(c^{(t)}) \quad (2.19)$$

Évaluation des réseaux de neurones récurrents

Nous avons établi que les RNN simples souffrent de la mémoire à court terme (BENGIO et al., 1994 ; PASCANU et al., s. d.). En utilisant les portes pour contrôler le flux d'information, les GRU et LSTM résolvent le problème au prix d'architectures plus complexes et par conséquent plus lourdes à entraîner. Ils ont des performances similaires et largement meilleures que celle des RNN (CHUNG et al., 2014).

Cependant, toutes les architectures récurrentes présentent un problème fondamental : elles fonctionnent séquentiellement. Bien que cela les rende naturellement mieux adaptées à la modélisation de séquences, il les rend aussi quasi impossibles à paralléliser pour exploiter des architectures parallèles (GPU). Par conséquent, l'entraînement des RNN est extrêmement lent (STAHLBERG, 2020).

2.5 Réseau de neurones à convolutions

En dépit d'être une architecture naturelle pour le traitement des séquences, les RNN sont trop lents pour la majorité des cas d'utilisation pratiques. Cela est dû à leur nature séquentielle qui, à son tour, est due à leur utilisation de boucles de rétroaction (voir Section 2.4). Une architecture sans de telles boucles (i.e une architecture de FFN) est donc préférable.

Les réseaux de neurones à convolutions (CNN, convolutional neural network) sont une famille de FFN typiquement utilisés en traitement d'images. Ils ont été introduits par (FUKUSHIMA, 1980) et popularisés par (LECUN et al., 1989 ; LECUN et al., 1998). Les CNN atteignent des performances comparables à celles des RNN sans mémoire explicite. Ils y parviennent en exploitant un outil mathématique appelé produit de *convolution* (CALIN, 2020).

Principes mathématiques de fonctionnement

Le produit de convolution de deux signaux f et g est le signal $f * g$ donné par

$$u \mapsto \int_{\mathbb{R}} f(u-t)g(t)dt \quad (2.20)$$

il s'agit d'une opération commune en probabilité, analyse fonctionnelle, traitement de signaux et traitement d'images (BARBE & LEDOUX, 2012; OPPENHEIM & SCHAFER, 2013). À partir d'elle, une autre opération appelée *l'inter-corrélation* est définie. L'inter-corrélation de f et g est notée $f \star g$. Elle est définie par⁸ :

$$u \mapsto \int_{\mathbb{R}} f(t)g(t-u)dt = (f * g^-)(u) \quad (2.21)$$

Intuitivement, elle mesure la similarité entre les deux signaux en question. Les CNN utilisent l'inter-corrélation à la place des applications linéaires quelconques d'un MLP. Une couche de convolution unidimensionnelle calcule donc la fonction suivante

$$x \mapsto \varphi \left(b + \sum_{i=1}^n x \star w_i \right) \quad (2.22)$$

où x est l'entrée et les vecteurs w_i sont les paramètres entraînables de la couche, aussi appelés ses *noyaux* ou *masques* (voir Figure 2.9). La sortie de cette couche est une combinaison de tous les éléments de la séquence d'entrée. En composant suffisamment de telles couches, un CNN apprend une représentation de l'entrée beaucoup plus structurée qu'un MLP. On parle de représentation *hiérarchique*.

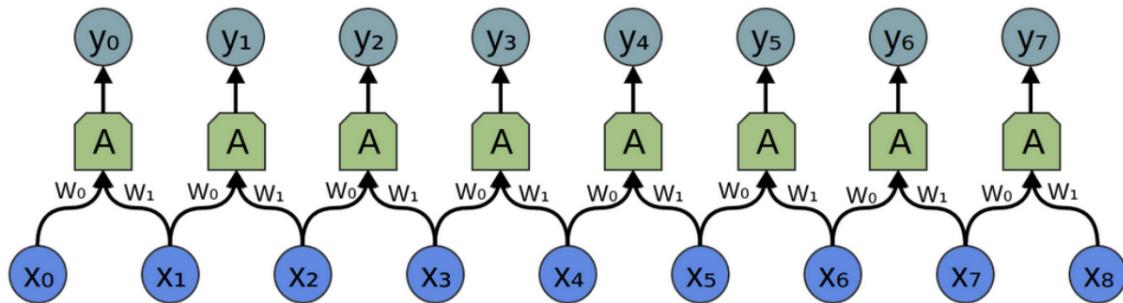


FIGURE 2.9 – Couche convulsive unidimensionnelle (OLAH, 2014).

Application à l'apprentissage S2S

Leur tendance naturelle à produire des représentations synthétiques des entrées, fait des CNN des encodeurs très puissants pour une grande variété de tâches, y compris des tâches de transduction de séquences. Plusieurs travaux ont combiné un encodeur convolutif avec un décodeur récurrent (voir Figure 2.10) (YANG et al., 2020).

Cependant, des architectures totalement convolutives pour l'apprentissage S2S existent également. ByteNet (voir Figure 2.11) et ConvS2S (voir Figure 2.12) sont deux exemples de telles architectures.

8. $g^- : t \mapsto g(-t)$.

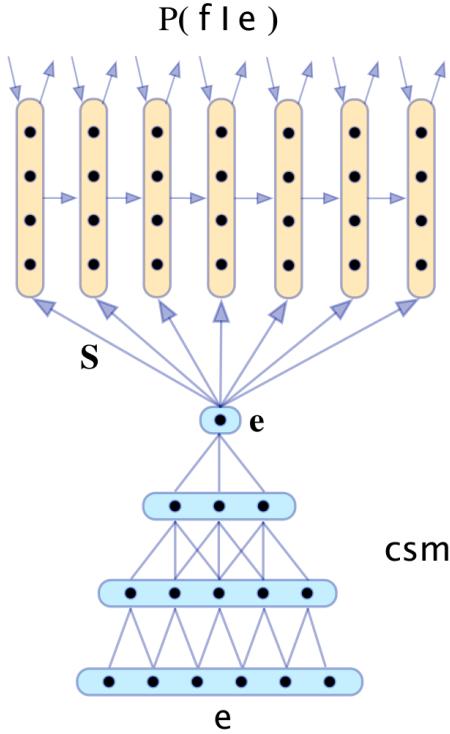


FIGURE 2.10 – Recurrent Continuous Translation Model (KALCHBRENNER & BLUNSMON, 2013).

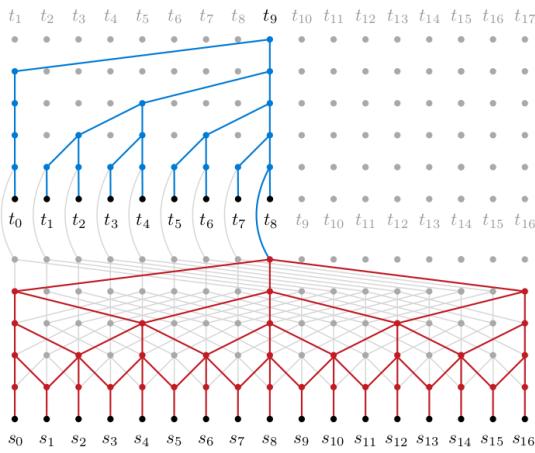


FIGURE 2.11 – Architecture de ByteNet (KALCHBRENNER et al., 2017).

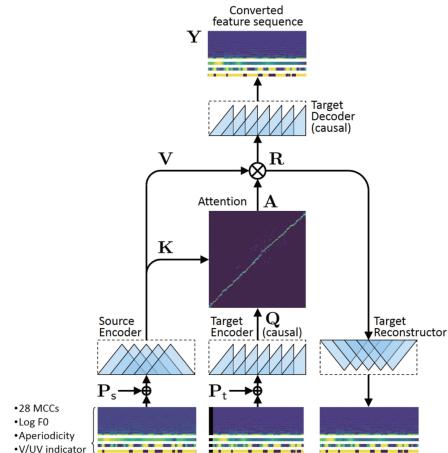


FIGURE 2.12 – Architecture de ConvS2S (KAMEOKA et al., 2020).

Avantages et inconvénients

Le produit de convolution (et par conséquent, inter-corrélation) est une opération très facile à paralléliser. Il est donc possible d'exploiter des architectures matérielles parallèles (GPU) pour accélérer l'entraînement des CNN. À cet effet, les CNN sont beaucoup plus efficaces que les RNN pour les longues entrées. Ils peuvent être employés dans des cas pratiques de tailles raisonnables (X. LI et al., 2016).

Or, la nature locale du produit de convolution rend difficile aux CNN d'apprendre les corrélations globales⁹. En effet, pour représenter toutes les relations dans une séquence de longueur n , un CNN dont les masques sont de taille $k < n$ doit avoir $\log_k n$ couches (VASWANI et al., 2017). Le traitement de séquences très longues nécessite alors des CNN trop profonds, ce qui donne lieu aux problèmes de disparition et d'explosion des gradients.

2.6 Transformeurs

Les architectures discutées dans les sections 2.2 à 2.5 (notamment les encodeurs-décodeurs à base de CNN et RNN) forment la colonne vertébrale des modèles classiques d'apprentissage S2S (YANG et al., 2020). Cependant, leur mise en place est inhibée par des problèmes de performance (voir sections 2.4, 2.5 et 2.6.5).

Le transformeur (aussi appelé réseau de neurones auto-attentif) est une architecture performante pour le traitement de séquences (SHIM & SUNG, 2022). Introduite par (VASWANI et al., 2017), elle est basée sur le mécanisme d'attention (LAROCHELLE & HINTON, 2010). Une opération mathématique parallélisable à l'instar du produit de convolution, mais dont la complexité ne dépend pas de la distance dans les séquences.

2.6.1 Architecture générale

Le transformeur a une architecture encodeur-décodeur (voir Figure 2.13). L'encodeur et le décodeur ont des architectures très similaires, comprenant chacun une couche d'encodage positionnel, une couche d'auto-attention et un MLP. Le décodeur se distingue de l'encodeur en ce qu'il a une couche d'attention croisée entre l'auto-attention et l'MLP. Chacune des couches susmentionnées est suivie d'une couche de normalisation (BA et al., 2016) qui reçoit également une connexion directe à l'entrée de la couche¹⁰. L'encodeur et le décodeur peuvent être empilés pour former un transformeur profond. VASWANI et al. proposent 6 couches d'encodeur et 6 couches de décodeur.

Étant donné une séquence d'entrée $x = (x_1, \dots, x_n)$, l'encodeur calcule un vecteur de pensée z qu'il passe au décodeur. Le décodeur prédit le prochain élément y_i de la séquence de sortie à partir de z et les éléments précédents (y_1, \dots, y_{i-1}) . Pour produire y_0 , le décodeur commence avec une séquence de sortie vide.

2.6.2 Encodage positionnel

La première couche de l'encodeur aussi bien que du décodeur est une couche d'encodage positionnel. Elle permet de calculer une représentation vectorielle de la séquence en

9. Dépendances entre des couples d'éléments éloignés dans la séquence

10. On parle de connexion résiduelle ou de connexion de saut (skip connection) (HE et al., 2016)

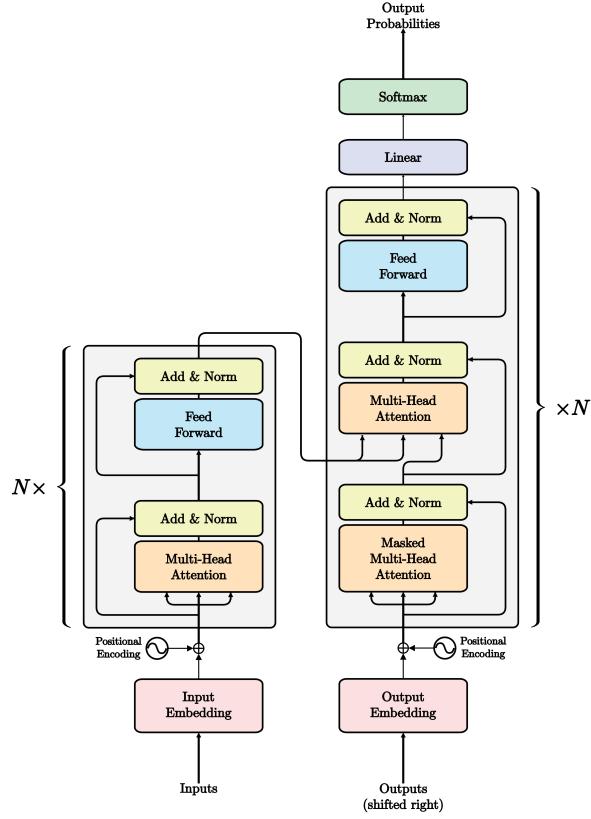


FIGURE 2.13 – Architecture de transformeur (VASWANI et al., 2017).

question. Elle est composée d'une couche de plongement et d'une couche d'encodage de la position.

La couche de plongement transforme chaque élément de la séquence en un vecteur de dimension fixe d , ainsi transformant une séquence de longueur n en une matrice de \mathbb{R}^{nd} . Elle peut avoir une architecture quelconque en fonction de la nature des données. Il peut s'agir d'une couche de plongement lexical dans le cas d'un texte, d'une couche de convolution dans le cas de l'audio ou d'une couche récurrente dans le cas d'une série chronologique.

Le transformeur est invariant aux permutations. La sortie de la couche de plongement est donc insuffisante pour représenter la séquence. Il lui faut ajouter une information sur l'ordre des éléments. C'est ce que fait la couche d'encodage de la position. Les auteurs de (Vaswani et al., 2017) proposent l'encodage suivant pour la position i :

$$\text{PE}_k(i) = \begin{cases} \sin\left(\frac{i}{r^{2k/d}}\right) & \text{si } k \text{ est pair} \\ \cos\left(\frac{i}{r^{2k/d}}\right) & \text{si } k \text{ est impair} \end{cases} \quad 1 \leq k \leq d \quad (2.23)$$

où d est la dimension de plongement et r est un hyperparamètre fixé à 10^4 par les auteurs. La figure 2.14 montre la matrice d'encodage des 256 premières positions avec $r = 10^4$ et $d = 512$.

La représentation finale de la séquence est la somme de la sortie de la couche de

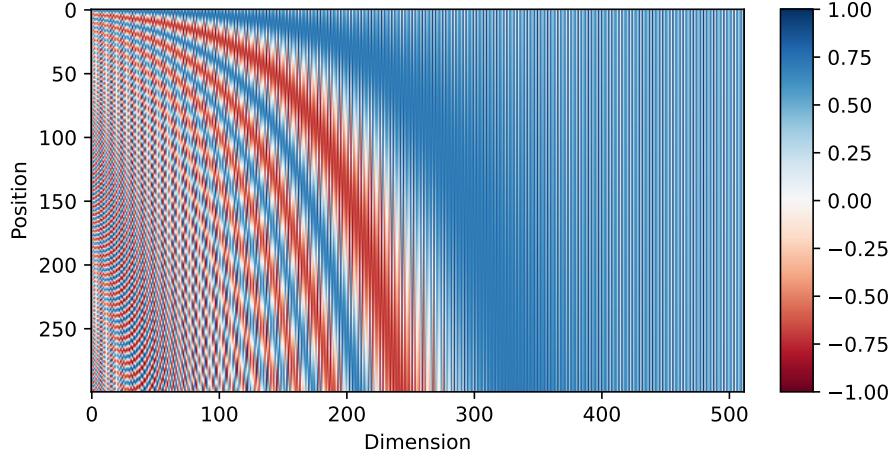


FIGURE 2.14 – Matrice d’encodage de la position pour $r = 10^4$ et $d = 512$.

plongement et celle de la couche d’encodage de la position. Une couche de plongement lexical peut être substituée à l’encodage donné par l’équation (2.23) avec des résultats similaires (VASWANI et al., 2017).

2.6.3 Couches attention

Le mécanisme d’attention est la partie centrale du transformeur qui réalise sa fonctionnalité. Tous les autres modules sont une forme de prétraitement de son entrée ou de post-traitement de sa sortie. Chaque couche de l’encodeur implémente un module d’attention, tandis que chaque couche du décodeur en implémente deux.

Plusieurs types de mécanismes d’attention ont été proposés dans la littérature, notamment l’attention additive (BAHDANAU et al., 2016) et l’attention multiplicative (LUONG et al., 2015). Le transformeur utilise une variante de l’attention multiplicative appelée “*scaled dot-product attention*” (VASWANI et al., 2017). Elle opère sur trois matrices Q, K, V ¹¹ de dimensions respectives d_Q, d_K, d_V , qui représentent chacune l’encodage d’une séquence¹². Le résultat est donné par l’équation (2.24).

$$\text{Attention}(Q, K, V) = \text{softmax} \left(\frac{QK^T}{\sqrt{d_K}} \right) V \quad (2.24)$$

Intuitivement, la *scaled dot-product attention* peut être vue comme une forme de recherche floue dans une base de données. Le terme QK^T est un produit scalaire entre les lignes de la requête Q et la clé K . Il représente la similarité entre les deux. La fonction softmax¹³ le normalise pour obtenir des poids positifs de somme 1. Enfin, le résultat est une somme des valeurs V pondérées par ces poids. C’est dans ce sens-là qu’il s’agit d’une recherche

11. De la terminologie anglophone : “*Query*”, “*Key*”, “*Value*”, empruntée aux systèmes de recherche de l’information.

12. d_Q, d_K, d_V sont les dimensions de plongement et l, m, n sont les longueurs des séquences. Notez que l’équation (2.24) impose les contraintes $d_Q = d_K$ et $m = n$, ce qui est vérifié pour les trois modules d’attention du transformeur.

13. softmax : $\mathbb{R}^p \rightarrow \mathbb{R}^p$, $x = (x_1, x_2, \dots, x_p) \mapsto \frac{e^x}{\sum_{i=1}^p e^{x_i}}$

floue : les valeurs correspondantes aux clés qui répondent le mieux à la requête sont sélectionnées, mais au lieu de renvoyer discrètement la meilleure valeur, toutes les valeurs contribuent dans la mesure de leur pertinence (« CS480/680 Lecture 19 : Attention and Transformer Networks », 2019). La division par $\sqrt{d_K}$ est une forme de régularisation. Elle permet d'éviter que le module de l'entrée du softmax devienne trop grand, ce qui rend son gradient trop petit ainsi ralentissant l'apprentissage¹⁴ (VASWANI et al., 2017).

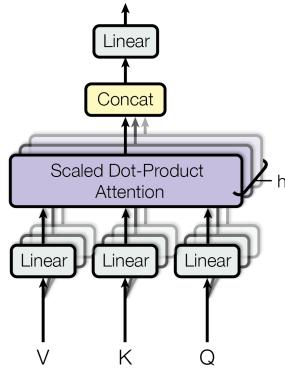


FIGURE 2.15 – Schéma d'une couche attention multi-tête (VASWANI et al., 2017).

Les auteurs de (VASWANI et al., 2017) ont observé une amélioration de la performance en utilisant plusieurs modules (ou têtes) de *scaled dot-product attention*, projetant les entrées différemment avant de les passer à chaque tête. Les sorties des têtes sont concaténées et passées à une couche linéaire qui produit la sortie finale. Les projections sont réalisées par des couches linéaires entraînables et le nombre h de têtes est un hyperparamètre fixé à 8 par les auteurs (voir la Figure 2.15).

La couche attention de l'encodeur et la première couche attention du décodeur sont des couches d'*auto-attention*, i.e leurs requêtes, clés et valeurs sont les mêmes. L'auto-attention du décodeur se distingue de celle de l'encodeur par la présence du *masque*. En produisant un élément de la séquence de sortie, le décodeur ne doit faire attention qu'aux éléments qui le précédent. Toutes les autres valeurs doivent avoir un poids nul. Pour ce faire, la matrice M

$$M_{ij} = \begin{cases} 0 & \text{si } j > i \\ -\infty & \text{sinon} \end{cases} = \begin{bmatrix} -\infty & -\infty & \cdots & -\infty \\ 0 & -\infty & \cdots & -\infty \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & -\infty \end{bmatrix} \quad (2.25)$$

est ajoutée à l'entrée de la fonction softmax.

La deuxième couche d'attention du décodeur est une couche d'*attention croisée*. Les requêtes viennent de la sortie et les clés et valeurs viennent de l'entrée. Cette couche calcule les dépendances entre l'entrée et la sortie, tandis que les deux autres couches calculent les dépendances à l'intérieur de l'entrée et de la sortie. C'est cette couche qui permet au décodeur de conditionner sa sortie sur l'entrée.

14. Le choix de la valeur de $\sqrt{d_K}$ n'est pas arbitraire. Sous l'hypothèse que les valeurs de Q et K sont centrées et réduites, les valeurs de QK^T sont centrées de variance d_K . La division par $\sqrt{d_K}$ les ramène à une variance de 1 (VASWANI et al., 2017).

2.6.4 Autres éléments de l'architecture

L'encodage positionnel et l'auto-attention sont les deux innovations clés du transformeur. Les autres éléments de l'architecture existent comme portes d'entrée et de sortie pour elles. Ces éléments sont les suivants :

- Les couches feed-forward : Il s'agit de MLP qui exploitent la représentation calculée par le mécanisme d'attention.
- Les couches de normalisation : Introduites par (BA et al., 2016), elles sont appliquées après chaque module. Elles assurent que leur sortie est centrée et réduite. Cela permet de stabiliser l'apprentissage.
- Les connexions résiduelles (HE et al., 2016) : Appliquées à chaque couche de normalisation, elles ont la forme LayerNorm ($x + \text{Module}(x)$). Elles permettent aux gradients de se propager plus facilement.
- La couche de sortie : Dans le cas où les éléments de la sortie sont des variables continues (par exemple pour amplitudes d'un signal audio), une couche linéaire est appliquée à la sortie du dernier module. Dans le cas opposé (par exemple pour les mots d'une phrase), la fonction softmax est aussi appliquée pour donner la loi de probabilité de la sortie.

2.6.5 Analyse comparative de la performance

Dans le début de cette section, nous affirmons la supériorité du transformeur aux autres architectures S2S. Ce constat repose sur la comparaison du Tableau 2.1 (VASWANI et al., 2017). On y voit que le transformeur est le seul à avoir une longueur de chemin constante, c'est-à-dire que cette longueur possède un majorant indépendant de la taille de l'entrée.

On note également que le transformeur exige un nombre d'opérations séquentielles constant, ce qui est le cas pour les autres architectures à l'exception des RNN. En effet, tous les réseaux feed-forward ont cette propriété.

Pour la complexité par couches, l'auto-attention est la plus efficace pour les courtes séquences (pour lesquelles $n \ll d$). Ces séquences sont les plus fréquentes en pratique pour les dimensions de représentation usuelles. Dans le cas de très longues séquences, elle est moins efficace que les autres architectures. Cependant, le transformeur donne toujours de meilleurs résultats pour ces séquences (SHIM & SUNG, 2022).

2.7 Conclusion

Dans ce chapitre, nous avons introduit l'apprentissage S2S, une tâche générale en ML dont la MT et l'ASR sont des cas particuliers (SZABO et al., s. d.). Nous avons défini le problème traité en modélisation S2S, expliqué les différentes architectures neuronales utilisées pour le résoudre et donné une comparaison de leurs performances théoriques.

Type du module	Complexité	Nombre d'Opérations Séquentielles	Longueur du Chemin Emprunté par le Gradient
Auto-Attention	$\mathcal{O}(n^2 \cdot d)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Recurrent	$\mathcal{O}(n \cdot d^2)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
Convolutif	$\mathcal{O}(k \cdot n \cdot d^2)$	$\mathcal{O}(1)$	$\mathcal{O}(\log_k n)$
MLP	$\mathcal{O}(n \cdot d^2)$	$\mathcal{O}(1)$	$+\infty$

TABLE 2.1 – Analyse comparative de la complexité des différents types d’architectures S2S. n est la longueur de la séquence, d est la dimension de la représentation vectorielle des éléments et k est la taille du noyau des convolutions (VASWANI et al., 2017).

Dans le chapitre suivant, nous étudions les deux exemples de MT et ASR dans la mesure où ils peuvent être appliqués à notre problématique.

Chapitre 3

Traduction automatique et reconnaissance automatique de la parole

Dans le premier chapitre, nous avons introduit la MT et l'ASR comme chemins possibles pour la réhabilitation de la parole chez les patients de l'aphasie de Broca. Ensuite, dans le chapitre précédent, nous avons présenté le problème général dont ces deux tâches sont des cas particuliers : celui de la modélisation S2S. Nous avons posé formellement le problème et présenté les architectures neuronales majeures qui ont été utilisées pour le résoudre en les comparant. Dans ce chapitre, nous abordons les aspects spécifiques de ces deux tâches et nous étudions l'application des architectures présentées (notamment le transformeur) dans leur contexte.

3.1 Traduction automatique

Étant donné un langage source L_S sur un vocabulaire Σ_S , un langage cible L_C sur un vocabulaire Σ_C , et une relation *d'équivalence*¹ \sim sur $L_S \cup L_C$ la MT de L_S en L_C consiste à trouver une fonction calculable $f : L_S \rightarrow L_C$ qui vérifie

$$\forall x \in L_S, \quad f(x) \sim x \tag{3.1}$$

la relation \sim donne un sens d'identité entre les phrases. Sa définition peut varier dans sa rigueur et sa précision. Par exemple, elle est mathématiquement définie dans le contexte de compilation² (HADJ, 2015). Elle a par contre une définition floue dans le contexte de la MT du langage naturel (CHAN, 2015). Notre contexte de MT pour la correction des erreurs³, peut être vu comme un cas intermédiaire (BRYANT et al., 2022).

1. Dans le sens mathématique du terme, c-à-d. une relation réflexive, symétrique et transitive.

2. Qui est bien un exemple de MT où L_S et L_C sont des langages de programmation et \sim est la relation d'équivalence sémantique.

3. Il s'agit là encore d'un exemple de MT. L_S est une copie de L_C avec un vocabulaire augmenté. En fonction de la complexité des erreurs, la relation \sim peut ressembler à l'égalité.

Ce flou dans la définition empêche l'application efficace de la RMBT dans ce contexte. La plupart des succès sont des exemples de NMT (YANG et al., 2020). Ces méthodes s'inscrivent facilement dans le cadre de l'apprentissage S2S tel que nous l'avons défini dans section 2.1. Il est donc naturel de considérer les modèles étudiés dans le chapitre 1.4 comme des candidats pour la MT. Plus précisément, l'architecture de transformeur est la plus prometteuse en vue de l'analyse comparative effectuée dans le chapitre 1.4 (voir Table 2.1). De ce constat, nous consacrons cette section à l'étude de l'utilisation des transformateurs pour la NMT.

3.1.1 Traduction automatique à base de transformeur

Le transformeur tel que nous l'avons présenté dans section 2.6 peut être utilisé pour la MT. En effet, (VASWANI et al., 2017) l'ont appliqué à cette même tâche. Ayant déjà introduit l'architecture et le principe de fonctionnement du transformeur, nous nous concentrerons sur les particularités de son application à la MT.

Plongement lexical

Le transformeur — comme tout réseau de neurones — n'opère que sur des vecteurs. Pour utiliser des phrases en entrée, il faut donc transformer ces phrases en vecteurs. La première étape consiste à transformer les mots en vecteurs. C'est ce qu'on appelle un *plongement lexical* (ALMEIDA & XEXÉO, 2019).

La première étape dans le calcul des plongements lexicaux est de diviser le texte en unités lexicales (tokens). Il peut s'agir de mots, de suites de mots ou d'unités plus petites qu'un mot. La sortie de cette étape est appelée un *vocabulaire*. Plusieurs techniques existent pour effectuer ce découpage. La plus simple est d'assimiler chaque mot à un token. Cette technique présente l'inconvénient de produire des vocabulaires très grands. Il est possible de limiter la taille du vocabulaire en ne gardant que les mots les plus fréquents, remplaçant les autres par un token spécial [UNK], mais cela a pour conséquence de perdre l'information sur les mots rares. Dans un contexte de correction d'erreurs, celles-ci sont intrinsèquement rares. La tokenisation à base de mots est inacceptable dans ce cas (RAI & BORAH, 2021).

D'autres techniques tentent de réduire la taille du vocabulaire en utilisant des tokens plus petits qu'un mot. On parle de techniques de tokenisation en sous-mots. L'une des plus connues est le byte pair encoding (BPE) (SENNRICH et al., 2016). Pour construire le vocabulaire, BPE commence par découper les mots en caractères (les bytes initiaux). Ensuite, il regroupe le couple de bytes les plus fréquents en un seul byte, remplaçant toutes leurs occurrences par ce dernier. Cette opération est répétée jusqu'à ce que le nombre de bytes soit égal au nombre de tokens désiré ou pour un nombre maximal d'itérations (voir Algorithme 3.1).

Après l'avoir construit, un plongement doit être associé au vocabulaire. L'application qui mappe un token à un vecteur peut avoir une implémentation arbitraire. Il peut s'agir d'un simple tableau de vecteurs (PASZKE et al., 2019), comme il peut s'agir d'un réseau

```

1 def get_stats(vocab):
2     pairs = defaultdict(int)
3     for word, freq in vocab.items():
4         symbols = word.split()
5         for i in range(len(symbols)-1):
6             pairs[symbols[i], symbols[i+1]] += freq
7     return pairs
8
9
10 def merge_vocab(pair, v_in):
11     v_out = {}
12     bigram = re.escape(" ".join(pair))
13     p = re.compile(r"(?<!\\S)" + bigram + r"(?!\\S)")
14     for word in v_in:
15         w_out = p.sub(" ".join(pair), word)
16         v_out[w_out] = v_in[word]
17     return v_out
18
19
20 vocab = {
21     "l o w </w>": 5,
22     "l o w e r </w>": 2,
23     "n e w e s t </w>": 6,
24     "w i d e s t </w>": 3
25 }
26 num_merges = 10
27
28 for _ in range(num_merges):
29     pairs = get_stats(vocab)
30     best = max(pairs, key=pairs.get)
31     vocab = merge_vocab(best, vocab)
32     print(best)

```

Extrait de code 3.1 – Byte pair encoding (SENNRICH et al., 2016).

de neurones (CHURCH, 2017). L’objectif est d’associer aux tokens similaires des vecteurs similaires⁴, on ramène ainsi, la relation mal définie de similarité entre les mots à une relation bien définie sur les vecteurs. Une fois qu’on a mis la phrase sous la forme d’une suite de vecteurs lexicaux, le transformeur peut la traiter (voir Section 2.6).

Évaluation et métriques

Plusieurs métriques peuvent être utilisées pour évaluer la qualité d’une traduction automatique. Les plus simples sont les métriques fondées sur la matrice de confusion (notamment la précision et le rappel). Ces métriques sont conçues pour quantifier la qualité des classifications simples. Cela les rend inadaptées à la MT. Par exemple, la précision de la traduction candidate “Le le le” par rapport à la référence “Le chat est sur le tapis” est égale à 1.

Des métriques plus adaptées à la MT ont été proposées. La plus connue parmi elles est bilingual evaluation understudy (BLEU) (PAPINENI et al., 2002). BLEU prend la précision

4. Qui ont un grand produit scalaire.

comme point de départ. Il l'améliore en apportant deux modifications : (1) il considère la précision sur les n -grammes⁵ plutôt que sur les mots individuels et (2) il prend en compte le nombre de fois où un n -gramme est présent dans la référence. Le résultat est appelé *la précision n -gramme modifiée*. Il est donné par la formule suivante⁶ :

$$p_n(\hat{y}, y) = \frac{\sum_{g \in G_n(\hat{y})} \min(\#(\hat{y}, g), \#(y, g))}{\sum_{g \in G_n(\hat{y})} \#(\hat{y}, g)} \quad (3.2)$$

BLEU est obtenu en prenant la moyenne géométrique des p_n multipliée par une pénalité exponentielle pour les phrases trop courtes (voir Algorithme 3.2).

```

1 def ngrams(sequence, n):
2     output = []
3     for i in range(len(sequence) - n + 1):
4         output.append(sequence[i:i + n])
5     return output
6
7
8 def modified_precision(candidate, truth, n):
9     candidate_ngrams = ngrams(candidate, n)
10    truth_ngrams = ngrams(truth, n)
11    count = 0
12    for ngram in candidate_ngrams:
13        if ngram in truth_ngrams:
14            count += 1
15        truth_ngrams.remove(ngram)
16    return count / len(candidate_ngrams)
17
18
19 def bleu(candidate, truth):
20     pns = []
21     for n in range(len(candidate)):
22         pns.append(modified_precision(candidate, truth, n + 1))
23     log_pns = [log(p_n) for p_n in pns]
24     brevity_penalty = 1
25     if len(candidate) > len(truth):
26         brevity_penalty = exp(1 - len(candidate) / len(truth))
27     return brevity_penalty * exp(sum(log_pns) / len(candidate))

```

Extrait de code 3.2 – Score BLEU.

Stratégie de décodage

La sortie de la dernière couche du transformeur est une loi de probabilité sur le vocabulaire de L_C (voir Figure 3.1). À fin d'obtenir une phrase, le décodeur est appelé autorégressivement avec la phrase vide comme grain. À chaque étape, il génère un mot en utilisant la loi de probabilité. Ce mot est ajouté à la phrase et le décodeur est appelé autorégressivement sur la phrase ainsi obtenue. Ce processus est répété jusqu'à ce que le symbole de fin de phrase soit généré (VASWANI et al., 2017).

5. séquences de n mots consécutifs.

6. $G_n(s)$ est l'ensemble des n -grammes de s et $\#(s, s')$ est le nombre d'occurrences de s' dans s .

Plusieurs méthodes existent pour générer un mot à partir de la loi de probabilité. La plus simple est de choisir le mot qui a la plus grande probabilité, on parle alors de décodage glouton, décodage argmax ou décodage par maximum a posteriori.

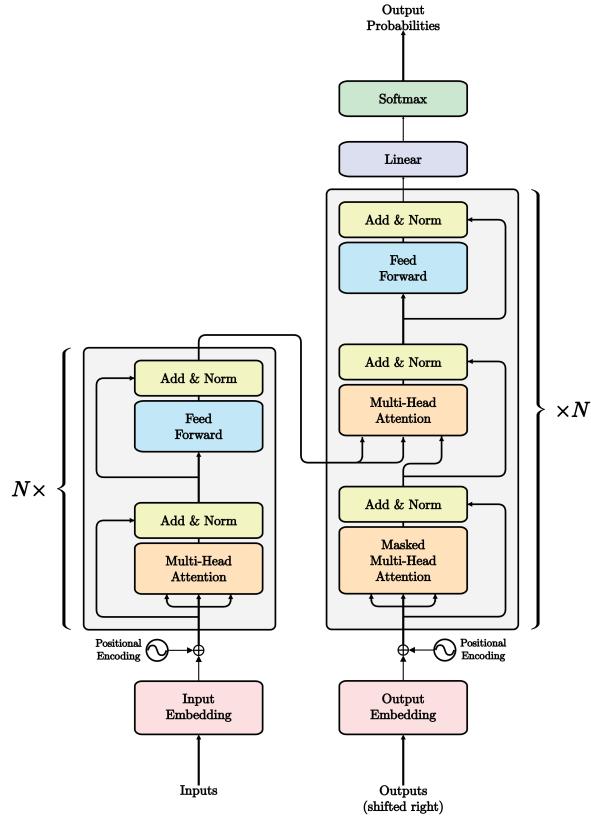


FIGURE 3.1 – Architecture de transformeur (VASWANI et al., 2017).

Cependant, cette méthode n'est pas optimale. L'alternative la plus courante est ce qu'on appelle le décodage par *beam search* (MEISTER et al., 2021). Le principe est de garder les b meilleurs candidats à chaque étape. À l'étape suivante, b nouveaux candidats sont générés à partir de chacun des b candidats gardés (ce qui donne b^2 candidats). Les b meilleurs candidats sont gardés et le processus est répété jusqu'à ce que le symbole de fin de phrase soit généré. (voir Algorithme 3.3 et Figure 3.2).

3.1.2 Generative pre-trained transformer

Generative pre-trained transformer (GPT) est un grand modèle de langage (LLM, de l'anglais : large language model) de OpenAI (RADFORD et al., 2018). Un modèle de langage est une loi de probabilité \mathbb{P} sur les phrases d'une langue (CHAN, 2015). Dans le cas de GPT, cette loi est calculée par un réseau de neurones. Plus précisément, il s'agit d'une *pile de décodeurs de transformeur*⁷ (voir Figure 3.3).

7. Une composition de N modules de décodeur. Dans (RADFORD et al., 2018), $N = 12$ (117M paramètres).

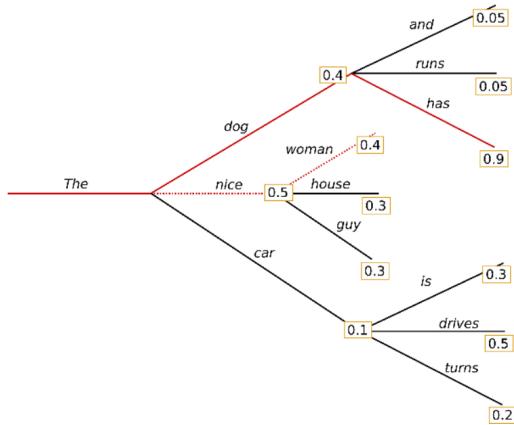


FIGURE 3.2 – Exemple de décodage par beam search (MEISTER et al., 2021).

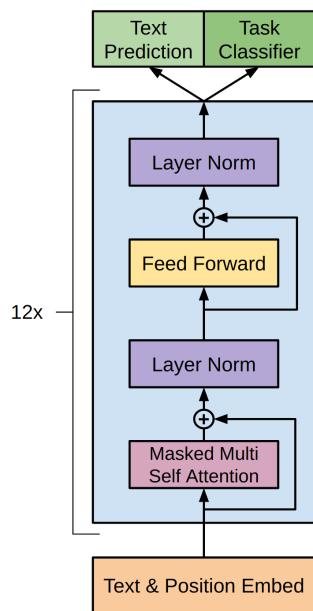


FIGURE 3.3 – Architecture de GPT (RADFORD et al., 2018).

L’entraînement de GPT passe par deux étapes. Dans un premier temps, il est pré-entraîné d’une façon autosupervisée sur un corpus de texte brut. Puis, il est affiné sur un corpus de texte annoté. Cette deuxième étape est un exemple d’apprentissage supervisé classique. La première est beaucoup plus intéressante. Pendant cette étape, le modèle est entraîné sur la tâche de modélisation causale du langage (CLM, de l’anglais : causal language modeling) i.e, la tâche de prédire, à partir d’une suite de tokens, le token suivant (RADFORD et al., 2018). Plus exactement, le modèle est entraîné pour maximiser la fonction :

$$L_1 = \sum_{i=k+1}^{n-k} \log \mathbb{P}(t_i | t_{i-k}, \dots, t_{i-1}; \theta) \quad (3.3)$$

où (t_1, t_2, \dots, t_n) est un corpus de taille n , k est un hyperparamètre qui détermine la taille de la fenêtre de contexte et θ représente les paramètres entraînables du modèle.

Le succès de GPT a motivé la création de plusieurs variantes. Notamment, GPT-2 (RADFORD et al., 2019) et GPT-3 (BROWN et al., 2020). La principale différence entre

```

1 def beam_search_decode(model, beam_width):
2     # Initialize the beam with a single blank sequence
3     beam = [((BOS,), 0)]
4
5     # Start exploring branches
6     while True:
7         # Generate all possible successors
8         successors = []
9         # for each sequence in the beam
10        for seq, score in beam:
11            successors.extend(
12                (seq + (next_word,), score + next_word.score)
13                for next_word in model(seq)
14            )
15        # Keep only the top b sequences
16        beam = sorted(successors, key=lambda x: x[1], reverse=True)
17        beam = beam[:beam_width]
18
19        # If all sequences in the beam end in <eos>, return the best one
20        if all(seq[-1] == EOS for seq, score in beam):
21            return beam[0][0]

```

Extrait de code 3.3 – Décodage par beam search (MEISTER et al., 2021).

ces variantes est la taille du modèle (1.5 B et 175B paramètres respectivement). Un autre modèle basé sur GPT (plus précisément sur GPT–2) qui est très pertinent pour notre travail est GPT–D. Il s’agit d’une copie de GPT–2 dont les paramètres ont été délibérément dégradés à fin de modéliser les erreurs linguistiques associées à la démence (C. LI et al., 2022).

3.1.3 Bidirectional encoder representations from transformers

Bidirectional encoder representations from transformers (BERT) est un LLM pré-entraîné de Google. Il s’agit d’une *pile d’encodeurs de transformeur* (voir Figure 3.4). BERT existe en deux versions : *base*, constitué de 12 couches d’encodeur et *large*, constitué de 24. Ils ont respectivement 110M et 340M de paramètres (DEVLIN et al., 2019).

Tout comme GPT, BERT est pré-entraîné d’une manière autosupervisée puis affiné sur une tâche spécifique. Pour le pré-entraînement de BERT, deux tâches sont utilisées. La première est la modélisation masquée du langage (MLM, de l’anglais : masked language modeling). Dans cette tâche, un pourcentage de tokens est aléatoirement remplacé par un token spécial [MASK]. Le modèle doit alors prédire les tokens cachés en prenant le reste de la phrase comme contexte. La deuxième tâche est la prédiction de la prochaine phrase (NSP, de l’anglais : next sentence prediction). Pour cette tâche, deux phrases sont concaténées et le modèle doit prédire si la deuxième phrase est la suite de la première (DEVLIN et al., 2019).

À son tour, BERT a motivé l’apparition de plusieurs variantes. En effet, il est l’un des modèles les plus réimplémentés dans la littérature à raison d’être open-source. Parmi les variantes, on peut citer : RoBERTa (Y. LIU et al., 2019), qui est identique à BERT sauf

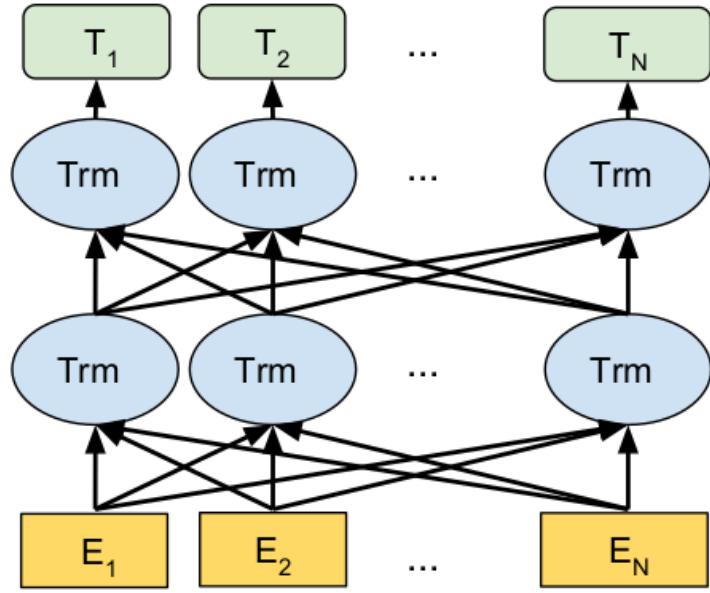


FIGURE 3.4 – Architecture de BERT (DEVLIN et al., 2019)

qu'il utilise un processus de pré-entraînement différent, CamemBERT (MARTIN et al., 2020), qui est une version française de BERT.

Plusieurs travaux dans la littérature ont utilisé BERT pour la NMT (CLINCHANT et al., 2019 ; ZHU et al., 2020). Certains l'ont même utilisé pour la représentation des phrases aphasiques pour la classification (QIN et al., 2022).

3.1.4 Bidirectional auto-regressive transformer

Bidirectional auto-regressive transformer (BART) est un LLM de Facebook AI. Contrairement à GPT et BERT, BART est un transformeur S2S. Il utilise BERT et GPT comme encodeur et décodeur respectivement (voir Figure 3.5).

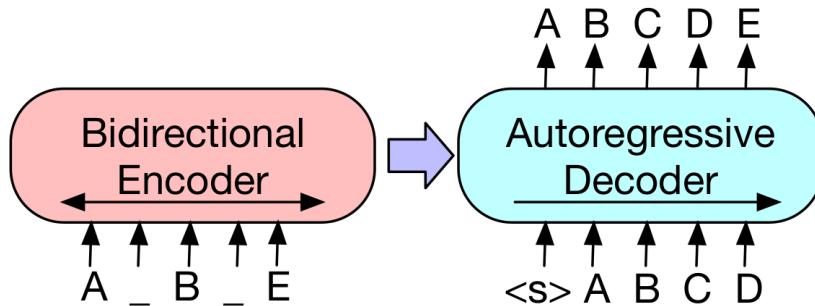


FIGURE 3.5 – Architecture de BART (LEWIS et al., 2019).

À l'instar de GPT et BERT, l'entraînement de BART est constitué d'une phase de pré-entraînement autosupervisé, suivie d'une phase d'affinement. BART est pré-entraîné sur la tâche d'auto-encodage du texte bruité. Similairement à la MLM, un extrait textuel

est modifié avant d'être passé à BART. Le modèle est alors chargé de reconstruire le texte original. Cette tâche est plus difficile que la MLM, car le but est de reconstruire la séquence plutôt que de prédire les tokens masqués, mais aussi, car l'ensemble des modifications possibles est plus grand que l'application d'un masque. Il inclut également la suppression d'un token (sans le remplacer par [MASK]), le remplacement d'une suite de token par [MASK] et l'application d'une permutation de phrases (LEWIS et al., 2019).

L'affinement dépend de la tâche cible. BART peut être affiné pour les tâches de classification (comme BERT), génération de texte (comme GPT) ou traduction (voir Figure 3.6).

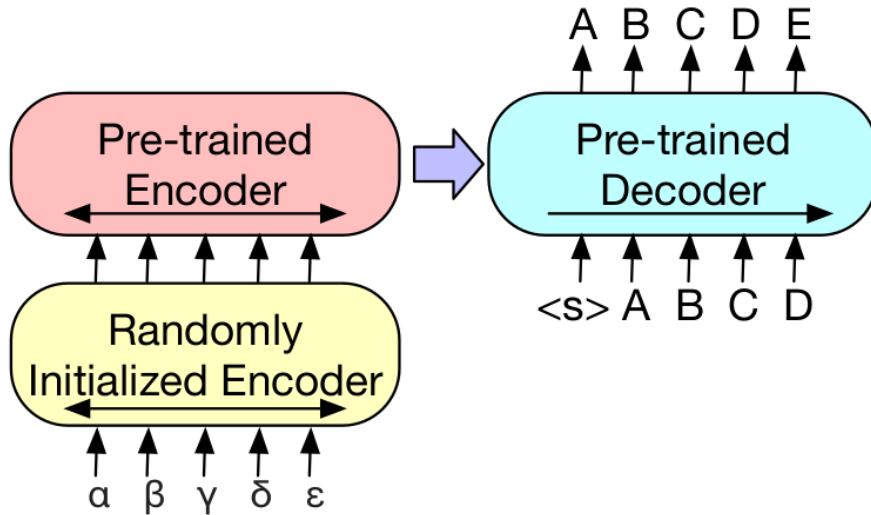


FIGURE 3.6 – Affinement de BART pour la traduction (LEWIS et al., 2019).

3.2 Reconnaissance automatique de la parole

L'ASR est un scénario d'apprentissage S2S où l'ensemble de départ est celui des signaux numériques audio $s \in \mathbb{R}^*$ ⁸, et l'ensemble d'arrivée est un langage L sur un vocabulaire Σ . Étant une tâche d'apprentissage S2S, l'ASR se fie très naturellement au traitement par transformateurs. En effet, il est plus facile de les appliquer à cette tâche, car l'entrée est déjà un vecteur. Plusieurs travaux ont exploré l'application des transformateurs à l'ASR. Deux travaux en particulier ont eu un succès retentissant. Il s'agit de (SCHNEIDER et al., 2019) et de (RADFORD et al., 2022).

3.2.1 Wav2Vec

Wav2Vec (SCHNEIDER et al., 2019) est un modèle de ASR proposé par Facebook AI. Son architecture est composée d'un CNN qui extrait des caractéristiques de l'entrée audio, suivie d'un transformateur qui effectue le traitement séquentiel (voir Figure 3.7).

8. On rappelle que $A^* := \bigcup_{n \in \mathbb{N}} A^n$.

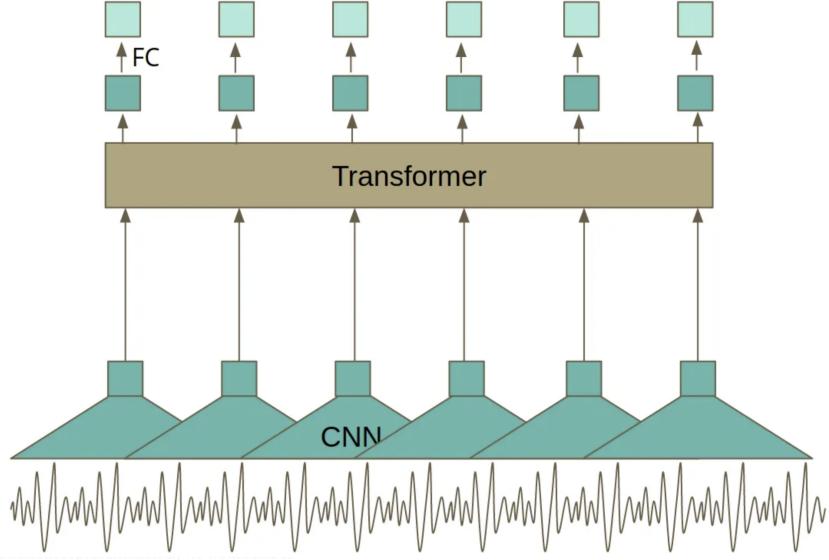


FIGURE 3.7 – Architecture de Wav2Vec (SUS, 2021).

3.2.2 Whisper

Whisper (RADFORD et al., 2022) est un modèle de ASR proposé par OpenAI. À l'instar de Wav2Vec, son architecture comporte un CNN et un transformeur. Cependant, contrairement à Wav2Vec, l'entrée n'est pas la forme temporelle du signal audio. Il s'agit plutôt de représentation spectrale (voir Figure 3.8). Whisper utilise un encodage positionnel sinusoïdal pour l'audio (similaire à (VASWANI et al., 2017)) et un encodage positionnel appris pour le texte (voir section 2.6).

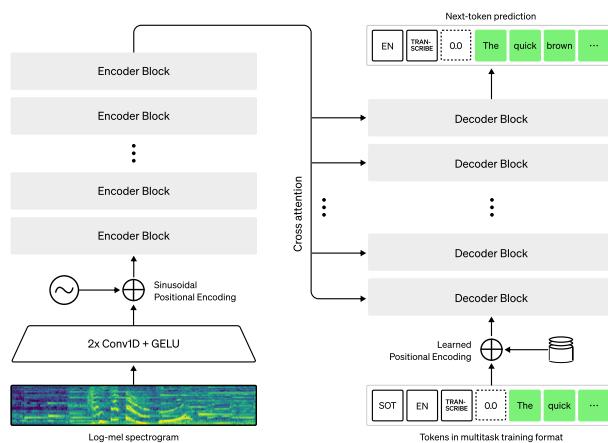


FIGURE 3.8 – Architecture de Whisper (RADFORD et al., 2022).

3.3 Conclusion

Dans ce chapitre, nous avons présenté l'utilisation des transformateurs pour la MT et l'ASR. Nous avons commencé par présenter les problèmes qu'il s'agit de résoudre. Ensuite, nous avons expliqué les modifications (s'il y en a) apportées au transformeur pour l'adapter à ces deux tâches. Enfin, nous avons présenté les travaux qui ont été réalisés dans ce domaine.

Deuxième partie

Contribution

Chapitre 4

Conception

Dans les chapitres précédents, nous avons effectué une étude bibliographique sur l’aphasie de Broca et les méthodes de NLP qui peuvent être utilisées pour la traiter (particulièrement les modèles S2S). Cela nous a permis de développer une idée claire d’un système S2S pour la réhabilitation de la parole aphasique.

Dans ce chapitre, nous allons présenter les détails de la conception de notre système. Nous commençons par décrire la démarche suivie pour le concevoir. Puis, nous présentons l’architecture générale du système, que nous détaillons par la suite.

4.1 Architecture générale de la solution

Pour résoudre le problème de la réhabilitation de la parole aphasique, nous proposons un système dont l’architecture générale est illustrée dans la Figure 4.1. Ce système est composé de deux parties principales : (a) le sous-système d’ASR qui permet de transcrire la parole aphasique en texte (partie gauche) et (b) le sous-système de NMT qui permet de traduire le texte transcrit en parole saine (partie droite).

Pour la partie ASR, nous avons commencé par la collecte de données existantes d’AphasiaBank. Ces données n’étant pas suffisantes, nous avons collecté des données supplémentaires sur internet. Ces dernières ont été transcrrites automatiquement à l’aide de Whisper. Les transcriptions automatiques ont été filtrées manuellement pour corriger les erreurs de transcription. Le résultat de cette étape est combiné avec les données d’AphasiaBank pour former un corpus de données de parole aphasique. Ce corpus peut être utilisé pour entraîner un modèle ASR.

Une fois les deux modèles entraînés, ils peuvent être enchainés pour former un système “speech-to-speech” (la phase d’exploitation dans la figure 4.1). Dans la suite de ce chapitre, nous reprenons dans le détail les différentes étapes de cette architecture, que nous avons abordées brièvement dans cette section.

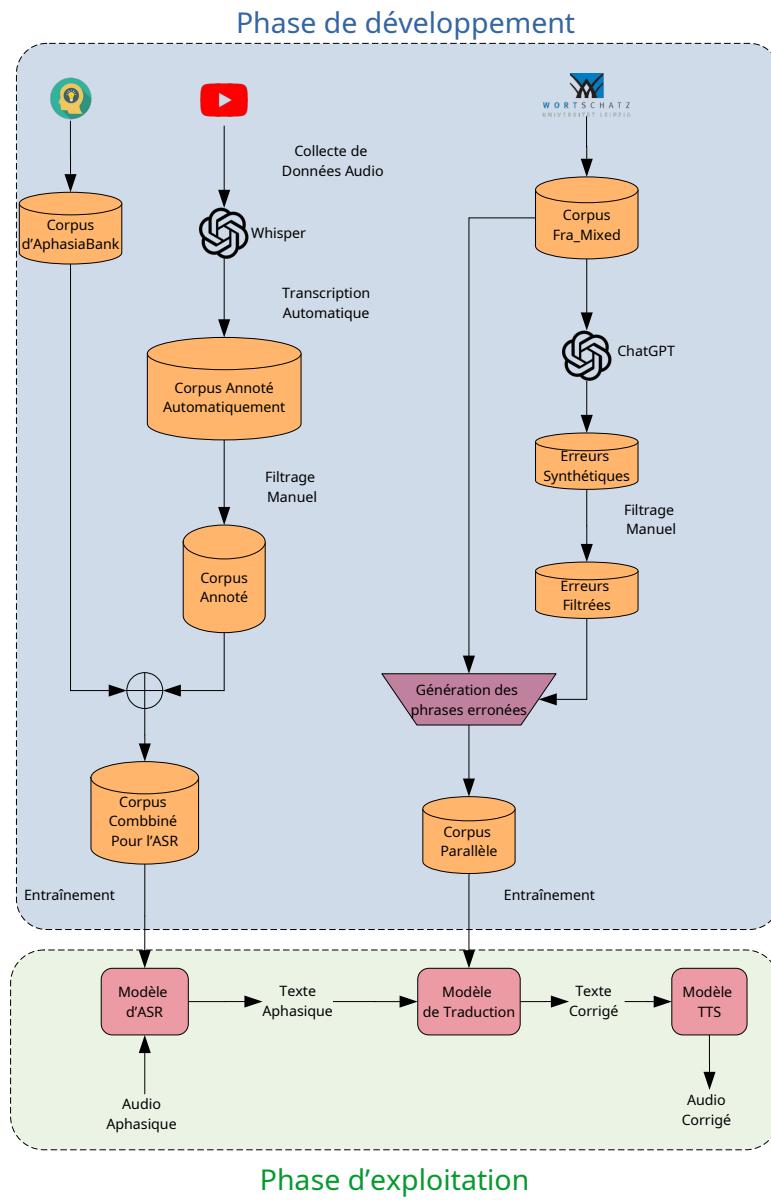


FIGURE 4.1 – Architecture générale de la solution.

4.2 Reconnaissance automatique de la parole

Le modèle d'ASR permet de transformer la parole aphasiique en texte dans le but de l'utiliser comme entrée pour le modèle de traduction. Les étapes de sa construction sont présentées dans la Figure 4.2. Dans cette section, nous allons détailler ces étapes. Nous notons que la dernière condition de la Figure 4.1 n'a pas été satisfaite. En effet, nous n'avons pas pu entraîner un modèle ASR à cause du manque de données.

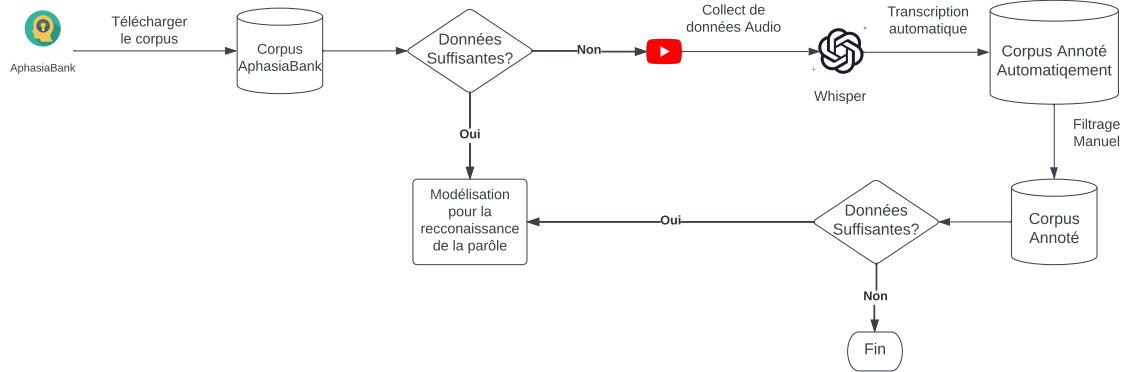


FIGURE 4.2 – Déroulement de la partie ASR.

4.2.1 Préparation des données

La première étape de tout projet de ML est la préparation des données. Dans ce cas, le jeu de données doit être constitué de couples de morceaux d’audio et de leurs transcriptions textuelles.

Corpus AphasiaBank

Notre choix s'est porté sur le corpus *AphasiaBank* (MACWHINNEY et al., 2011). Il fait partie du projet *TalkBank* (MACWHINNEY, 2007), une collection de bases de données créées pour l'étude du langage. AphasiaBank contient plusieurs vidéos d'entre-teints entre des chercheurs et des personnes souffrant de l'aphasie de Broca. Ses vidéos sont accompagnées par des transcriptions textuelles faites par des experts dans un format particulier. La qualité des transcriptions est donc excellente.



Cependant, le volume de données sur l'aphasie de Broca est très limité. En effet, un seul des 11 exemples disponibles en français est un exemple de l'aphasie de Broca. Il s'agit d'une vidéo de 12 min 03 s, qui contient 3000 mots. Il faut noter que la moitié de ces mots sont prononcés par le chercheur. La durée effective de la parole aphasique est donc de 6 min. Cela est très loin d'être suffisant pour entraîner un modèle profond. Pour ce but, il est nécessaire de collecter des données supplémentaires.

Collecte de données supplémentaires

Les données d'AphasiaBank étant insuffisantes, d'autres sources sont nécessaires. Plusieurs enregistrements de personnes souffrant de l'aphasie de Broca sont disponibles sur internet (YouTube, Vimeo, ...). La qualité de ces enregistrements est très variable et largement inférieure à celle d'AphasiaBank. Cependant, leur ajout au corpus est nécessaire pour augmenter sa taille.

Notre recherche nous a permis d'obtenir 22 enregistrements de personnes souffrant de l'aphasie de Broca d'une durée totale de 48 min 44 s. Des statistiques sur la répartition démographique de ces enregistrements sont présentées dans le tableau 4.1. On y observe

	Nombre	Durée
Hommes	7	13 min 45 s
Femmes	31	34 min 2 s
Groupe	2	9 min 57 s

TABLE 4.1 – Répartition des enregistrements collectés par genre.

que les enregistrements sont assez diverses en comparaisons à l'unique enregistrement trouvé sur AphasiaBank.

Transcription et filtrage des données

Les 22 enregistrements collectés sont transcrits à l'aide de Whisper (RADFORD et al., 2022). Cela permet d'avoir des transcriptions textuelles de qualité. Cependant, les transcriptions obtenues contiennent des erreurs (particulièrement pour les prononciations aphasiques).

L'étape suivante est de filtrer à la main les transcriptions obtenues. Cela permet de corriger les erreurs de transcription et de réintroduire les prononciations aphasiques éliminées par Whisper. La sortie de cette étape est un corpus (parole/écrit).

Les données d'AphasiaBank sont déjà transcrrites, mais cela est fait dans un format particulier. Il est donc nécessaire de réécrire les transcriptions en français standard. L'exemple d'AphasiaBank et donc ajouté au corpus, ce qui fait un total de 1 h 0 min 47 s de parole aphasique. Cela est encore insuffisant pour entraîner un modèle profond. Cependant, il peut servir aux chercheurs qui souhaitent travailler sur l'aphasie de Broca. On rend donc disponible ce corpus.

4.2.2 Création et entraînement du modèle

Les données collectées n'étant pas suffisantes, il n'est pas possible d'entraîner un modèle de DL. Nous avons donc décidé de mettre en pause le développement de ce modèle jusqu'à ce que des données supplémentaires soient disponibles. Dans le but de faciliter l'accès à de telles données, nous avons mis notre corpus à la disposition des chercheurs. Pour le reste de ce projet, nous nous focalisons sur le modèle de traduction.

4.3 Traduction automatique neuronale

La deuxième partie de notre système (et celle qui réalise sa fonction principale) est le modèle de traduction. Ce modèle corrige la parole aphasique pour la rendre plus compré-

hensible. La procédure de sa création est décrite dans la Figure 4.3. Dans cette section, nous allons détailler les étapes de sa construction.

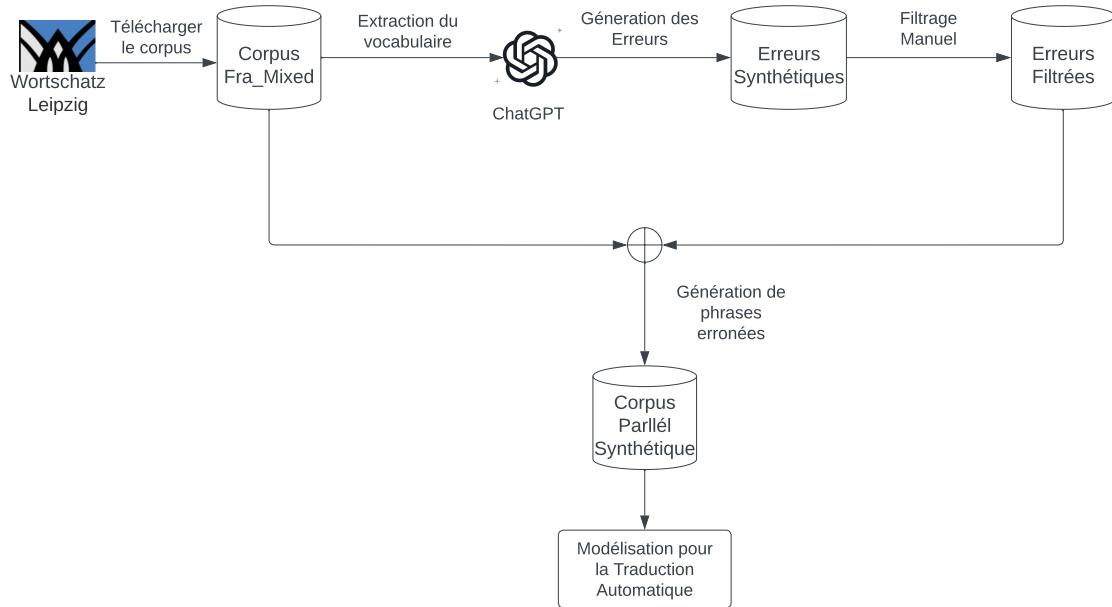


FIGURE 4.3 – Déroulement de la partie NMT.

4.3.1 Création d'un corpus parallèle

L’entraînement d’un modèle de traduction nécessite la présence d’un *corpus parallèle*, c’est-à-dire un corpus contenant les mêmes phrases dans plusieurs langues (dans notre cas, français et français aphasique). Cependant, un tel corpus n’existe pas. Il est donc nécessaire de le créer. La création d’un tel corpus nécessite la collecte d’un corpus de parole aphasique de taille suffisante. Ce corpus doit être ensuite traité par des experts pour corriger les erreurs dedans. Or, nous avons déjà établi qu’un corpus de parole aphasique de taille suffisante n’existe pas. Ce chemin donc est infranchissable dans ce moment.

L’alternative proposée dans (SMAÏLI et al., 2022) — et celle que nous prenons — est de créer un corpus synthétique. c.-à.-d., partir d’un corpus de parole normale et le modifier pour introduire des erreurs similaires à celles trouvées dans la parole aphasique.

Corpus de parole normale

Nous avons utilisé le corpus `fra_mixed100k` de *Leipzig Corpora Collection* (GOLDHAHN et al., 2012). Il s’agit d’un corpus de 100000 phrases françaises collectées de plusieurs sites web. Ces phrases sont diverses dans leur contenu, longueur, vocabulaire et structure grammaticale.

Extraction du vocabulaire et sélection des mots à modifier

Les erreurs causées par l'aphasie de Broca ne sont pas déterministes. Un individu qui en souffre ne se trompe pas sur tous les mots, ni de la même manière sur le même mot. Cependant, les erreurs ne sont pas uniformément réparties sur le vocabulaire. Naturellement, un tel individu à tendance à se tromper plus sur les mots “*difficiles*”. Il est aussi plus susceptible de se tromper sur les mots qu'il utilise le plus souvent. On peut donc s'attendre à un biais pour les mots fréquents et difficiles.

Pour simuler ce biais dans notre corpus, nous avons extrait le vocabulaire du corpus. Puis, nous avons sélectionné les mots “*difficiles*”. Nous avons considéré un mot “*difficile*” s'il est long (plus de 2 syllabes). Nous avons considéré ses mots dans l'ordre de leur fréquence dans le corpus (les 1000 premiers).

Génération et filtrage des erreurs synthétiques

Les mots sélectionnés à l'étape précédente sont donnés à chatGPT qui est chargé de générer 10 erreurs pour chaque mot dans le style d'un individu souffrant de l'aphasie de Broca. Le résultat de cette opération est une liste de 10000 couples (mot, erreur).

Ces couples sont filtrés manuellement pour supprimer les erreurs trop similaires au mot original (par exemple celle qui en diffèrent uniquement par la suppression d'une lettre) et les erreurs dissimilaires à celle produite par un individu aphasique. Cela a donné une moyenne de 5 erreurs retenues par mot.

Génération des phrases erronées

Le corpus parallèle est créé à partir du corpus original C et de l'ensemble des erreurs filtrés L de la façon suivante (voir Figure 4.4) :

1. Sélectionner de C les phrases qui contiennent au moins un mot présent dans L .
2. Pour chaque phrase sélectionnée, générer des variantes erronées en remplaçant les mots présents dans L par les erreurs correspondantes (si plusieurs mots existent, prendre toutes les combinaisons possibles).
3. Insérer les phrases générées dans le corpus parallèle avec la phrase de départ comme traduction.

Pour illustrer le processus de modification des phrases, prenons l'exemple suivant de la phrase “Maintenant que j'ai suffisamment d'argent, je peux m'acheter cet appareil photo.” avec

$$L = \begin{cases} \text{suffisamment} & \rightarrow \text{fussamment} \\ \text{suffisamment} & \rightarrow \text{suffimment} \\ \text{appareil} & \rightarrow \text{apairel} \\ \text{appareil} & \rightarrow \text{paparel} \\ \text{appareil} & \rightarrow \text{pareil} \end{cases}$$

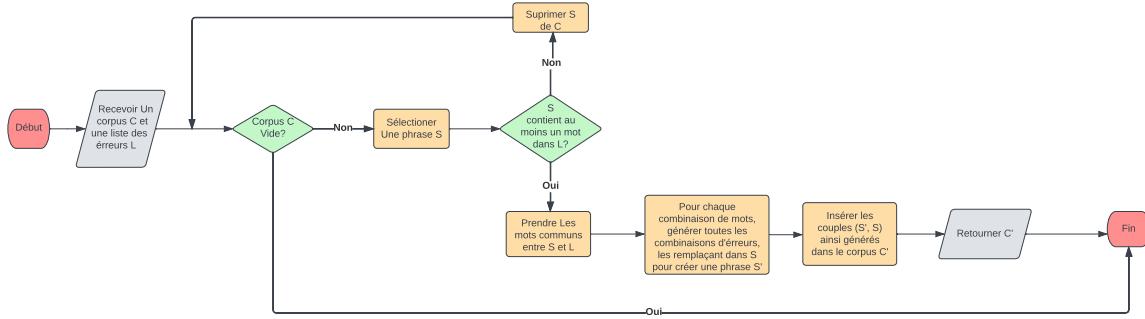


FIGURE 4.4 – Organigramme de la création du corpus parallèle.

Pour cet exemple, nous obtenons 11 phrases modifiées : 2 où uniquement le mot “suffisamment” est modifié, 3 où uniquement le mot “appareil” est modifié et 6 où les deux mots sont modifiés.

Le corpus ainsi créé compte 282689 couples de phrases. Cela est bien suffisant pour entraîner un réseau de neurones. L’étape de modélisation peut donc être entamée.

4.3.2 Création du modèle

Après avoir construit le corpus, nous pouvons passer à la création du modèle de traduction. Dans cette section, nous allons détailler les étapes de sa construction. La procédure d’entraînement est discutée dans la section suivante.

Division du corpus

Le corpus est divisé en trois parties : entraînement, validation et test. La partie entraînement est utilisée pour optimiser les paramètres du modèle. Cela peut conduire au phénomène de sur-apprentissage (où le modèle est bien ajusté aux données d’entraînement, mais ne généralise pas bien). Pour détecter ce phénomène, nous utilisons la partie validation pour évaluer le modèle pendant l’entraînement. La partie test est réservée pour l’évaluation finale du modèle (après l’entraînement).

Tokenisation

La création du modèle implique plusieurs choix techniques. L’un des plus importants parmi ces choix est celui du tokeniseur. Dans Section 3.1.1, nous avons introduit le tokeniseur BPE. Nous avons décidé d’utiliser ce tokeniseur basé sur BPE qui s’appelle “WordPiece”. Il a été introduit par Google dans (DEVLIN et al., 2019).

Comme BPE, WordPiece part d’un vocabulaire réduit à l’alphabet du corpus puis, il l’élargit en fusionnant itérativement les tokens adjacents. Contrairement à BPE, la

fusion n'est pas faite sur la base de la fréquence, mais sur celle de la fonction d'évaluation suivante :

$$\text{score}(x, y) = \frac{\mathbb{P}(x, y)}{\mathbb{P}(x)\mathbb{P}(y)} \quad (4.1)$$

où x et y sont deux tokens dans le vocabulaire à une itération donnée.

La division de la fréquence du couple par le produit de celles de ses composants a pour but de défavoriser la fusion des tokens fréquents (qui sont souvent des mots). Cela empêche la création de tokens plus longs qu'un mot et permet de conserver les tokens qui représentent des affixes communs (comme "im-" et "-able").

Nous avons opté pour une taille de vocabulaire de 5000 tokens pour la source et la cible. Des tokens spéciaux sont ajoutés au vocabulaire pour représenter la structure de la phrase. Ces tokens sont :

- [BOS] : début de la phrase ;
- [EOS] : fin de la phrase ;
- [UNK] : token inconnu (qui n'existe pas dans le vocabulaire) ;
- [PAD] : token de remplissage (utilisé pour ramener les phrases à la même longueur dans le cas de l'entraînement par lots).

À la sortie du tokeniseur, une opération de *numérisation* est effectuée. Il s'agit d'une application bijective entre les tokens et les entiers (que nous appelons "indices"). Cela permet de représenter les phrases par des suites d'entiers de longueur variable. Chose qui facilite leur représentation vectorielle. Le fait qu'elle soit bijective permet de reconstruire les phrases à partir de ces suites d'entiers produites par le modèle.

Représentation vectorielle des mots

Comme discuté dans la Section 3.1.1, un plongement lexical est appliqué après la tokenisation. En effet, les indices retournés par le tokeniseur peuvent en principe être utilisés tels quels. Cela présente en outre deux problèmes majeurs. Le premier est que la plage des indices est aussi grande que la taille du vocabulaire. Cela garantit que les mots de grands indices écrasent les autres lors de l'entraînement. Le deuxième problème est que les indices ne sont pas structurés selon une quelconque relation sémantique. Bien que d'être le plus souvent basés sur la fréquence des tokens, les indices ne contiennent pas d'information sur leur distribution conjointe.

Un plongement lexical est une application $\Sigma \rightarrow \mathbb{R}^d$ où Σ est le vocabulaire (qui passe le plus souvent par les indices) et d est la dimension du plongement (que nous avons fixé à 64)¹. Cela permet de remédier aux problèmes cités ci-dessus. Le premier problème est résolu, car la norme des vecteurs de plongement peut être contrôlée. Le deuxième est réglé, car les plongements, étant des vecteurs, présentent une structure vectorielle. Il est possible de définir la similarité entre deux plongements en utilisant le produit scalaire. Cela permet de capturer les relations sémantiques entre les tokens en s'assurant que les tokens corrélés sont représentés par des vecteurs similaires (voir Figure 4.5).

1. Il est commun dans la littérature de trouver des plongements qui sont des puissances de 2 (PASZKE et al., 2019 ; VASWANI et al., 2017). 64 nous semble être un bon compromis entre la dimension par défaut de 512 et la taille du vocabulaire.

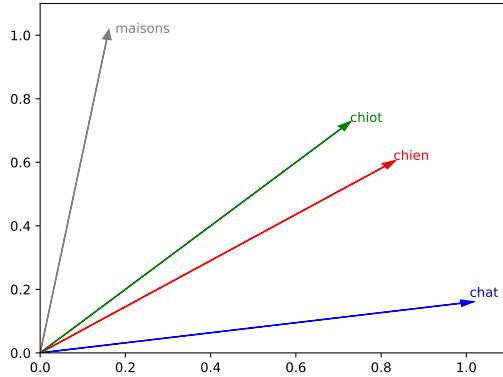


FIGURE 4.5 – Les plongements lexicaux de quelques mots avec $d = 2$.

Plusieurs algorithmes existent pour calculer les plongements lexicaux (voir Section 3.1.1). Dans ce travail, nous utilisons une couche de plongement lexical simple. Il contient un tableau de $|\Sigma|$ vecteurs de dimension d . Elle est donc paramétrée par les composantes de ces vecteurs. Cela donne une matrice de paramètres W dont les lignes sont les vecteurs de plongement :

$$W = \left\{ \begin{array}{c} \overbrace{\begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1N} \\ w_{21} & w_{22} & \cdots & w_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ w_{d1} & w_{d2} & \cdots & w_{dN} \end{bmatrix}}^{d=64} \\ |\Sigma| = 5000 \end{array} \right\} \quad (4.2)$$

qui a donc $|\Sigma|d = 500064 = 320000$ paramètres. Pour calculer le plongement d'un token x d'indice i , il suffit de prendre la ligne $i+1$ de W . La matrice W peut être optimisée comme n'importe quelle autre matrice de paramètres. Cela permet d'apprendre les plongements lexicaux à partir des données (PASZKE et al., 2019).

Architecture du modèle

Suite à l'étude bibliographique du Chapitre 2.7, nous avons choisi d'utiliser un transformeur comme architecture de base pour notre modèle. Nous avons opté pour le transformeur de base décrit dans (VASWANI et al., 2017).

Notre modèle est composé de 3 couches d'encodeurs et de 3 couches de décodeurs. Chaque couche a 4 têtes d'attention. La dimension de toutes les couches est de 64 et la fonction ReLU est utilisée comme fonction d'activation². Pour l'encodage positionnel, nous avons choisi de l'apprendre plutôt que d'utiliser un encodage sinusoïdal comme celui décrit dans (VASWANI et al., 2017).

2. ReLU : $\mathbb{R} \rightarrow \mathbb{R}, x \mapsto \max(0, x)$.

4.3.3 Entraînement

L’entraînement du modèle est un problème d’optimisation. Étant donné une fonction qui mesure la dissimilarité entre la sortie du modèle et la cible, le but est de trouver les valeurs des paramètres qui minimisent cette fonction.

Algorithmes d’optimisation

Plusieurs algorithmes d’optimisation sont utilisés pour entraîner les modèles de DL. La majorité d’entre eux sont basés sur l’algorithme du gradient. C’est le cas de l’algorithme d’optimisation Adam (KINGMA & BA, 2017) que nous avons utilisé. Il s’agit d’un algorithme itératif qui met à jour les paramètres du modèle à chaque itération. Son comportement est contrôlé par plusieurs hyperparamètres, mais le seul que nous avons modifié est le taux d’apprentissage η que nous avons initialisé à 3×10^{-4} , une valeur récurrente dans la littérature (ISLAM et al., 2022 ; LU et al., 2023).

Pour accélérer l’entraînement, il est fait d’une manière *stochastique*. Cela signifie que la fonction de perte est estimée sur un sous-ensemble des données d’entraînement qu’on appelle un *lot*. La taille du lot peut avoir un grand impact sur la performance du modèle. Dans notre cas, nous avons utilisé des lots de 256 paires de phrases³.

Contre-mesures au sur-apprentissage

Le sur-apprentissage est un problème courant dans l’entraînement des modèles de DL. Pour mitiger ce risque, nous avons utilisé plusieurs techniques. L’une d’entre elles est le *dropout*. Il s’agit d’une technique de régularisation qui consiste à ignorer aléatoirement une fraction p_{drop} des valeurs d’une couche.

Une autre méthode de régularisation est la majoration de la norme des vecteurs de plongement. Cela permet de contraindre les paramètres des couches de plongement lexical et, par conséquent, de réduire sa puissance de représentation.

Réglage des hyperparamètres

Les hyperparamètres sont les paramètres du modèle qui ne sont pas appris durant l’entraînement. Dans notre cas, ces paramètres incluent les dimensions de plongement, la taille du vocabulaire, le nombre de couches du transformeur, le nombre de têtes d’attention, le *dropout* et la taille des lots d’entraînement. Leurs valeurs sont souvent fixées par l’utilisateur. Cependant, elles peuvent avoir un impact significatif sur les performances du modèle. Elles sont donc souvent choisies en explorant systématiquement l’espace des possibilités. Cette exploration peut se faire d’une manière exhaustive ou aléatoire.

Le problème de réglage des hyperparamètres est aussi un problème d’optimisation. Or,

3. La plus grande taille de lot compatible avec la mémoire de la carte graphique utilisée.

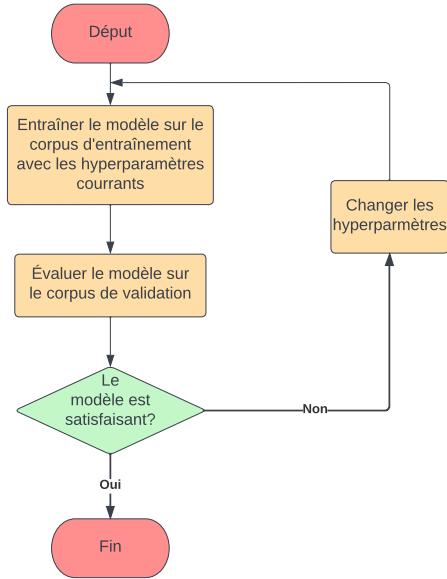


FIGURE 4.6 – Organigramme de la phase d’entraînement.

il est souvent de nature discrète ou mixte, car les hyperparamètres ne sont pas nécessairement continus. Cela rend l’optimisation beaucoup plus difficile. La fonction objectif est généralement calculée à partir du jeu de données de validation.

Le processus d’entraînement dans sa totalité est illustré par la Figure 4.6. On note sur la figure que le réglage des hyperparamètres n’est effectué que si le modèle entraîné avec la combinaison initiale d’hyperparamètres n’est pas satisfaisant. Sinon, la condition d’arrêt est atteinte et le modèle est utilisé pour la phase d’évaluation.

Il est important de noter que le corpus de test est particulièrement important dans le cas où le réglage des hyperparamètres est effectué. Dans ce cas, il est possible que le sur-apprentissage se produise simultanément sur le corpus d’entraînement et celui de validation. Le corpus de test est donc la seule manière d’obtenir une estimation non biaisée de la performance du modèle.

4.3.4 Évaluation et métriques

En ML, l’évaluation quantitative de la performance d’un modèle est une étape incontournable. Elle permet d’obtenir des mesures objectives de la qualité du modèle. Cela est d’autant plus important que le modèle est destiné à être déployé dans un système critique. Pour ce faire, il faut définir des métriques qui peuvent, à partir des prédictions du modèle et des valeurs réelles, fournir un nombre (ou une liste de nombres) qui représente la qualité du modèle. La MT étant un exemple de problème de classification en grande dimension⁴ (YANG et al., 2020), les métriques utilisées sont celles de la classification.

4. Où les classes sont les tokens du vocabulaire de la langue cible.

Entropie croisée

L'entropie croisée est une mesure de la dissimilarité de deux lois de probabilité (VASILEV et al., 2019). Pour deux lois de probabilité p et q sur le même espace probabilisable (Ω, \mathcal{A}) , leur entropie croisée $H(p, q)$ est définie par :

$$H(p, q) = H(p) + D_{KL}(p \parallel q) \quad (4.3)$$

où $H(p)$ est l'entropie de p et $D_{KL}(p \parallel q)$ est la divergence de Kullback-Leibler de p par rapport à q (BISHOP, 2006). Dans le cas de lois de probabilité discrètes, l'équation 4.3 devient :

$$H(p, q) = \mathbb{E}_{X \sim p} [-\log q(x)] = - \sum_{x \in \Omega} p(x) \log q(x) \quad (4.4)$$

où nous avons, par abus de notation, noté $p(x)$ et $q(x)$ les probabilités des événements $\{x\}$ par les lois p et q respectivement.

Quand l'entropie croisée est utilisée dans le cadre d'un problème de classification, on prend souvent comme p la loi de probabilité *cible*, c'est-à-dire celle des valeurs réelles. Dans notre cas, il s'agit de la loi du prochain token sachant la phrase source et les tokens précédents. On prend généralement une *loi de Dirac*⁵, on parle dans ce cas d'encodage *one-hot*. Pour la loi q , la sortie du modèle (typiquement avec une fonction softmax) est prise. L'entropie croisée devient donc une mesure de la déviation de la sortie du modèle de la vérité terrain. Sa minimisation est donc un objectif naturel pour l'entraînement du modèle.

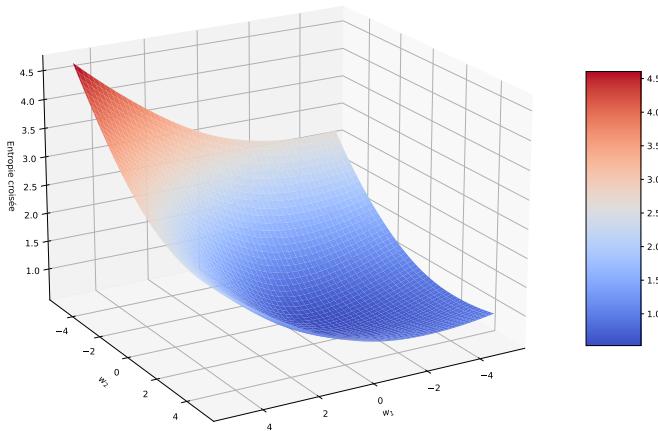


FIGURE 4.7 – Entropie croisée d'un classifieur linéaire binaire.

L'entropie croisée présente un avantage important par rapport aux autres métriques de classification : elle est *dérivable* par rapport aux paramètres du modèle. La Figure 4.7

5. Pour un espace probabilisable (Ω, \mathcal{A}) et $a \in \Omega$, la loi de Dirac $\delta_a : \mathcal{A} \rightarrow \{0, 1\}$ est définie par :

$$\delta_a(X) = \begin{cases} 1 & \text{si } a \in X \\ 0 & \text{sinon} \end{cases}$$

l'illustre sur le cas d'un modèle de régression logistique en deux dimensions. On y voit que la surface qui représente l'entropie croisée est lisse. Elle est également convexe dans ce cas, mais cela n'est pas vrai pour un réseau de neurones.

Le fait d'être dérivable (contrairement à la majorité des alternatives) rend possible sa minimisation par un algorithme de gradient comme Adam. C'est pour cette raison que l'entropie croisée est la fonction de perte la plus utilisée en classification. Pour cette même raison, elle est utilisée dans notre système pour l'entraînement du modèle.

Cela étant dit, l'entropie croisée a le défaut de ne pas être interprétable. Comme elle prend ses valeurs dans l'intervalle $[0, +\infty]$, elle ne donne aucune information sur la différence relative entre deux modèles. Elle permet de déterminer si un modèle \mathcal{M} est plus performant qu'un autre modèle \mathcal{M}' , mais elle ne nous dit rien sur le degré auquel \mathcal{M} dépasse \mathcal{M}' . Pour cette raison, des métriques normalisées sont nécessaires pour comparer des modèles.

Exactitude, précision, rappel et mesure F_β

Une façon universelle d'évaluer la performance d'un modèle de classification est de regarder sa *matrice de confusion*. Il s'agit d'une matrice carrée M dont les lignes sont indexées par les classes réelles et les colonnes par les classes prédites. L'élément à la position (i, j) de la matrice est le nombre d'exemples de la classe i attribués par le modèle à la classe j . Un modèle parfait à donc une matrice diagonale pour matrice de confusion (SEBASTIAN & MIRJALILI, 2017).

Il est moins facile d'interpréter la matrice de confusion d'un modèle imparfait. Pour cette raison, il est utile d'en extraire des nombres directement interprétables. Il est possible de construire ces nombres à partir de la matrice de confusion en posant 3 questions assez naturelles :

- (1) Quelle est la proportion d'exemples correctement classés ? C'est-à-dire, quelle est la proportion d'exemples pour lesquels la classe prédite est la même que la classe réelle ?
- (2) Quelle proportion des exemples de la classe i est attribuée à la classe i ?
- (3) Quelle proportion des exemples attribués à la classe j est réellement de la classe j ?

La réponse à la première question est donnée par l'*exactitude* du modèle. Il s'agit d'un nombre entre 0 et 1 qui mesure la similarité entre la matrice de confusion du modèle et la matrice diagonale du modèle parfait. L'exactitude est définie par :

$$\text{exactitude} = \frac{\text{tr } M}{\text{sum}(M)} = \frac{\sum_{i=1}^n M_{ii}}{\sum_{i=1}^n \sum_{j=1}^n M_{ij}} \quad (4.5)$$

elle donne une mesure de la performance globale du modèle, indépendamment des classes. Cela n'est pas toujours souhaitable, un exemple d'un cas problématique est celui d'un modèle qui prédit toujours la classe majoritaire. L'exactitude de ce modèle est minorée par la fréquence de cette classe. Dans le cas où les classes sont très déséquilibrées, un

modèle sans aucune capacité à décerner les classes minoritaires peut avoir une exactitude élevée en attribuant simplement toutes les classes à l'une des classes majoritaires. Cela suggère un besoin de mesures spécifiques aux classes, ce qui est ce que nous obtenons en abordant les questions (2) et (3) (SEBASTIAN & MIRJALILI, 2017).

Les réponses des deux questions restantes sont parfaitement symétriques (comme le sont les questions elles-mêmes). Elles sont données respectivement par le *rappel* et la *précision* du modèle. Le rappel de la classe i est défini par :

$$\text{rappel}_i = \frac{M_{ii}}{\sum_{j=1}^n M_{ij}} \quad (4.6)$$

il nous permet de savoir, sachant la classe d'un exemple, la probabilité que le modèle le classe correctement. Une valeur élevée de rappel suggère que le modèle tend à bien identifier les éléments de la classe i (on dit qu'il a peu de *faux négatifs* pour cette classe). La précision de la classe j est définie par :

$$\text{précision}_j = \frac{M_{jj}}{\sum_{i=1}^n M_{ij}} \quad (4.7)$$

elle nous permet de savoir, sachant la prédiction du modèle, la probabilité qu'elle soit correcte. Une grande précision suggère que le modèle tend à ne pas attribuer la classe j à tort (on dit qu'il a peu de *faux positifs* pour cette classe). Il est possible d'obtenir à partir de la précision et du rappel des mesures globales comme l'exactitude. Il suffit pour cela de prendre la moyenne des précisions ou des rappels sur toutes les classes. Cela présente l'avantage de traiter équitablement toutes les classes (SEBASTIAN & MIRJALILI, 2017).

La mesure F_β permet d'agréger le rappel et la précision en une seule mesure qui contient l'information sur les faux positifs et les faux négatifs. Elle est paramétrée par un réel positif β qui peut être interprété comme le degré d'importance des faux positifs comparé aux faux négatifs. Un $\beta = 2$ signifie que les faux positifs sont pénalisés 2 fois plus que les faux négatifs (van RIJSBERGEN, 2002). Le score F_β est défini comme la *moyenne harmonique* de la précision et le rappel pondérée par $\alpha = \frac{1}{1+\beta^2}$:

$$F_\beta = \frac{1}{\frac{\alpha}{P} + \frac{1-\alpha}{R}} \quad (4.8)$$

où P et R sont respectivement la précision et le rappel du modèle. La valeur la plus courante de β est 1, on parle alors simplement de F_1 -mesure.

Score BLEU

Toutes les métriques discutées jusqu'à présent sont des métriques de classification. Elles ne sont pas spécifiques à la tâche de traduction. En effet, elles mesurent la performance du modèle sur le niveau des tokens individuels. Elles ne nous apprennent rien sur sa performance au niveau des phrases entières.

Pour combler cette lacune, des métriques comme BLEU (voir Section 3.1.1) ont été conçues. Dans notre travail, nous avons choisi de l'utiliser comme objectif pour le réglage

des hyperparamètres pendant la validation. Nous l'avons calculé sur les sorties du modèle avec *forçage de l'enseignement*. Cela signifie que la sortie est produite à partir de l'entrée et d'une version décalée d'elle-même avec un masque causal (voir Section 2.6).

4.4 Conclusion

Dans ce chapitre, nous avons présenté les détails de la conception de la solution que nous avons proposé. Il s'agit d'un système à deux composantes principales : un modèle d'ASR et un modèle de traduction. Pour chacune de ces composantes, nous avons expliqué la procédure suivie pour l'acquisition et la préparation des données.

Dans le cas du modèle d'ASR, nous avons trouvé que les données ne sont pas suffisantes pour entraîner un modèle de qualité. Nous nous sommes donc contentés de la création d'un corpus de données qui peut être exploité par des futurs travaux.

Pour le cas de la traduction, nous avons créé un corpus synthétique de taille suffisante pour entraîner un modèle. Nous avons présenté les différentes décisions que nous avons prises pour la création et l'entraînement de ce modèle. Ces décisions incluent le choix de l'architecture et de l'algorithme d'entraînement, les valeurs des hyperparamètres, les métriques à mesurer et la possibilité d'optimisation des hyperparamètres.

Dans le chapitre suivant, nous présentons notre réalisation de la solution conformément à la conception proposée.

Chapitre 5

Réalisation

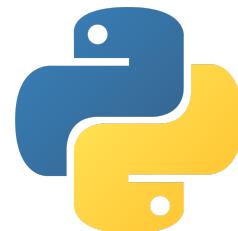
Dans le chapitre précédent, nous avons présenté les détails fonctionnels de la conception de notre solution. Le présent chapitre est consacré à la réalisation de cette dernière. Nous commençons par présenter les outils et technologies utilisés pour sa mise en place. Ensuite, nous présentons dans le détail les points importants de l'implémentation des différentes étapes de cette solution.

5.1 Outils et technologies

La création d'un système aussi sophistiqué que le nôtre est une tâche assez complexe. Elle comporte plusieurs étapes dont certaines nécessitent d'entraîner des réseaux de neurones ou de faire appel à des réseaux de neurones pré-entraînés par le biais d'interfaces de programmation d'application (API, de l'anglais : application programming interface). Pour faciliter certaines de ces étapes, nous avons exploité plusieurs outils et technologies open-source. Cette section est consacrée à la présentation de ces derniers.

5.1.1 Python

Python est un langage de programmation open-source interprété, orienté objet et multiparadigme. Introduit en 1991 par Guido van Rossum (« About Python », 2022), Python est aujourd’hui l’un des langages de programmation les plus populaires au monde. Le (« Stack Overflow Developer Survey », 2022) l’a classé comme la 4^e technologie la plus populaire au monde (48.07%) et la 6^e technologie la plus aimée (67.34%). La documentation officielle de Python le décrit comme suit :



“Python est un langage de programmation puissant et facile à apprendre. Il dispose de structures de données de haut niveau et permet une approche simple, mais efficace de la programmation orientée objet. Parce que sa syntaxe est élégante, que son typage est dynamique et qu'il est interprété, Python est un

langage idéal pour l'écriture de scripts et le développement rapide d'applications dans de nombreux domaines et sur la plupart des plateformes.”

— « Le tutoriel Python », 2023

La popularité de Python fait qu'il y a une grande communauté de développeurs qui contribuent à son écosystème. En effet, [le répertoire officiel de Python](#) contient plus de 450 000 packages. Tous les programmes écrits dans le cadre de ce projet sont écrits en Python.

5.1.2 Jupyter

Jupyter est un environnement d'exécution interactif pour Python (et plusieurs autres langages). Il permet de créer des documents appelés “notebooks” qui combinent le code exécutable avec une documentation textuelle. Cela est réalisé en divisant le document en unités appelées des “cellules”. Une cellule est associée à un langage qui peut être écrit dedans. Si ce langage est un langage de programmation (Python, R ou Julia), la cellule est exécutable. Si ce langage est Markdown, elle ne l'est pas.



5.1.3 Google Colaboratory, Kaggle et Paperspace Gradient

Colaboratory est un service fourni par Google. Il donne à ses utilisateurs l'accès à un environnement d'exécution Jupyter hébergé sur le cloud. Il est possible d'y créer et d'y exécuter des notebooks sans installation ni configuration. Il offre également (et gratuitement) le choix entre les exécuter sur un CPU ou un GPU. La machine virtuelle qui héberge cet environnement est réinitialisée après 12 h. Cette limite peut être étendue en utilisant un compte payant.



Kaggle est une plateforme en ligne qui permet d'organiser des compétitions en data science. Dans ce cadre, elle donne accès à un environnement similaire à celui trouvé sur Google Colaboratory. Les GPU qu'elle offre ont 16 Go de mémoire. Le temps de calcul sur Kaggle est limité à 30 heures par semaine.



Paperspace est une plateforme du calcul cloud dédiée au ML et à l'intelligence artificielle. Elle permet de créer des machines virtuelles avec des configurations personnalisées. Paperspace possède une grande variété d'architectures matérielles. Des CPU et des GPU y sont disponibles, mais aussi des TPU et des IPU. Il est possible de personnaliser le nombre de CPU, GPU, TPU et IPU à inclure dans une machine, ainsi que le type et les caractéristiques des CPU et GPU.

Gradient est un service offert par Paperspace. Similairement à Google Colaboratory et Kaggle, il permet d'accéder à un environnement Jupyter hébergé sur le cloud. Cependant, il offre plus de flexibilité que ces derniers. Il permet de personnaliser d'une manière similaire aux machines virtuelles de Paperspace.



5.1.4 Anaconda

Anaconda est une distribution logicielle open-source qui regroupe Python, R et une multitude de leurs packages consacrés au calcul scientifique et aux data science. Anaconda permet de créer des environnements virtuels pour Python pour isoler les dépendances entre les projets. Chaque environnement possède sa propre installation de Python et de ses packages. Un gestionnaire de packages appelé "conda" est fourni comme alternative à pip.



5.1.5 CUDA

CUDA est une technologie de calcul parallèle développée par NVIDIA. Elle permet d'utiliser les GPU de NVIDIA pour accélérer les calculs parallélisables. CUDA est programmable en C++. Cependant, il existe des bibliothèques qui permettent de l'utiliser avec d'autres langages (y compris Python).



5.1.6 PyTorch

PyTorch est une bibliothèque open-source de DL pour Python. Elle permet de créer et d'entraîner rapidement et facilement des réseaux de neurones. Elle est développée par Facebook AI, mais elle rejoint la fondation Linux en 2022.

PyTorch possède plusieurs modules. Les plus importants sont :

- **torch.Tensor** : ce module fournit une implémentation du tenseur, un tableau multidimensionnel qui peut être manipulé mathématiquement d'une façon qui généralise les scalaires, les vecteurs et les matrices.
- **torch.nn** : ce module contient une implémentation des réseaux de neurones. La classe la plus importante dans ce module est **torch.nn.Module**. Il s'agit d'une classe abstraite qui doit être héritée pour créer un réseau de neurones. Des implémentations pour les architectures les plus courantes sont fournies dans ce module (CNN, RNN, transformeur, etc.).



- `torch.optim` : ce module offre un ensemble d’algorithmes d’optimisation.
- `torch.autograd` : ce module fournit une implémentation de la dérivation automatique (backward). Il permet de calculer les gradients des paramètres en construisant un graphe de calcul puis en le traversant pour évaluer les dérivées.

Toutes les opérations de PyTorch peuvent être exécutées sur un GPU en utilisant CUDA. Cela rend très rapide l’entraînement des réseaux de neurones.

L’écosystème de PyTorch est très riche. Plusieurs packages sont construits là-dessus par la communauté ainsi que par l’équipe de PyTorch. Parmi les packages `torchtext` et `torchdata` nous sont les plus utiles. Le premier nous offre des objets utiles pour le traitement de texte en langage naturel (tokeniseurs, jeux de données populaires, modèles de langage, etc.). Le second implémente les fonctionnalités de manipulation de données. Les deux classes les plus importantes dans ce module sont `torchdata.Dataset` et `torchdata.DataLoader`.

5.1.7 Lightning

Lightning.AI est une plateforme pour créer, entraîner et déployer des modèles de DL. La bibliothèque `lightning` distribuée contient 3 packages (voir Figure 5.1) :

- `lightning.pytorch`
- `lightning.fabric`
- `lightning.applications`

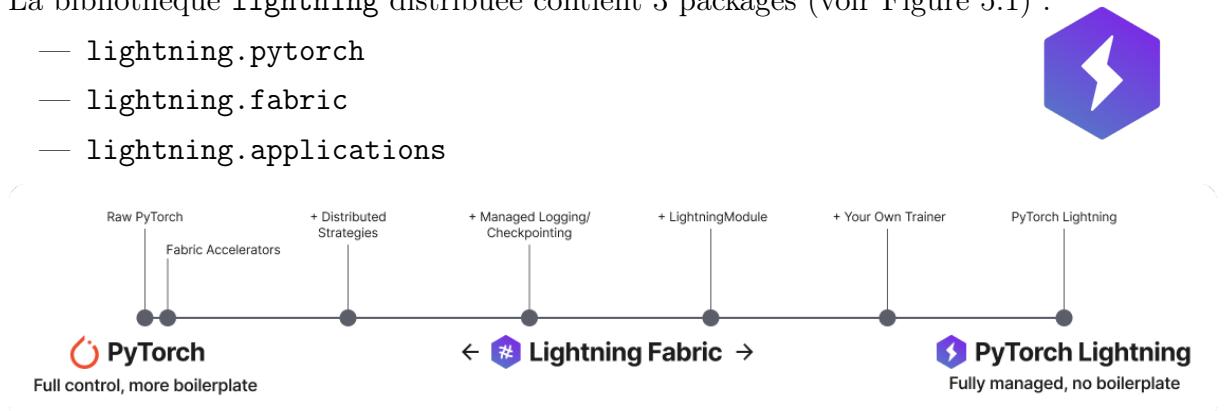


FIGURE 5.1 – Composantes de la bibliothèque `lightning` (FALCON & THE PYTORCH LIGHTNING TEAM, 2019).

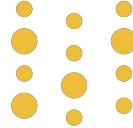
`lightning.pytorch` (aussi appelé `pytorch-lightning`) est le seul que nous avons utilisé. Il permet d’étendre PyTorch pour faciliter la création et l’entraînement des modèles. Les fonctionnalités qu’il offre incluent les rappels de fonctions et la journalisation.

Lightning.AI distribue un quatrième package en dehors de la bibliothèque `lightning` : `torchmetrics`. Ce dernier implémente plus de 90 métriques d’évaluation, dont l’exactitude et le score BLEU. Nous avons fait usage de ce package pour calculer ces métriques.

5.1.8 Weights & Biases

Weights & Biases est une plateforme de Machine Learning Operations (MLOps). Son but est de faciliter le processus de développement, de journalisation des expériences, de gestion des versions de jeux de données et de collaboration entre les membres d'une équipe.

l'API de Weights & Biases est accessible via un module Python `wandb` ou via une interface en ligne de commande du même nom. Elle offre la possibilité de journaliser les métriques d'évaluation, les hyperparamètres, les graphes de calcul, etc. Elle est intégrée avec `lightning` à travers la classe `lightning.pytorch.loggers.WandbLogger`.



5.1.9 Hugging Face

Hugging Face est une entreprise qui se spécialise en intelligence artificielle. Elle distribue gratuitement plusieurs bibliothèques open-source et des modèles pré-entraînés.

Parmi ces bibliothèques, nous avons utilisé `tokenizers` qui permet de créer des tokeniseurs et `evaluate` qui implémente plusieurs métriques d'évaluation (dont la perplexité). Nous l'avons utilisé plutôt que `torchmetrics`, car elle permet d'utiliser les modèles pré-entraînés dans Hugging Face Hub.



5.1.10 Open AI

Open AI est une entreprise de recherche en intelligence artificielle. C'est cette entreprise qui a créé les modèles GPT et Whisper discutés dans le chapitre 2.7.

Nous avons fait appel à l'API d'Open AI pour utiliser chatGPT, une version de GPT-3 qui est entraînée sur des conversations. Nous l'avons utilisé pour générer les erreurs synthétiques. Nous avons également utilisé Whisper pour la transcription des vidéos collectées, mais nous l'avons utilisé localement, car il est open-source.



5.1.11 Pynecone

Pynecone est une bibliothèque qui permet de développer des sites web entièrement en Python (sans besoin d'écrire du code HTML, CSS ou JavaScript). Nous l'avons utilisé pour créer une interface web pour le modèle de correction d'erreurs.

5.1.12 Autres bibliothèques

La Table 5.1 donne une liste des autres bibliothèques dont nous avons fait un usage secondaire. Une liste complète des bibliothèques Python installées dans notre environnement est donnée par l'Annexe 6.6.

Bibliothèque	Description et utilisation
<code>pytorch_memlab</code>	Diagnostique de l'utilisation du mémoire GPU par CUDA
<code>PyHyphen</code>	Division des mots en liste de syllabes
<code>torchview</code>	Dessin des graphes de calcul des modules de PyTorch
<code>PyYAML</code>	Chargement et écriture des fichiers <code>yaml</code> en Python
<code>beautifulsoup4</code>	Analyse syntaxique des pages HTML
<code>Requests</code>	Utilisation du protocole HTTP
<code>scikit_learn</code>	Division du jeu de données
<code>pandas</code>	Division du jeu de données
<code>python-dotenv</code>	Chargement des variables d'environnement
<code>tqdm</code>	Visualisation de l'état d'avancement d'une boucle

TABLE 5.1 – Bibliothèques auxiliaires.

5.2 Crédit des corpus

Dans cette section, nous donnons les détails de l'implémentation de la première étape de notre solution, à savoir la création des corpus. Nous commençons par le corpus de la partie ASR. Ensuite, nous passons à celui de la partie MT.

5.2.1 Reconnaissance automatique de la parole

Pour la création du corpus de la partie ASR, nous avons commencé par repérer les vidéos pertinentes sur YouTube et Vimeo. Le résultat de cette opération est une liste de 3 vidéos qui n'ont pas déjà été transcris.

Après cela, nous avons divisé chaque vidéo en plusieurs segments en fonction de la personne qui parle. Nous avons organisé ces segments en un fichier `url.yaml` dont un extrait est donné ci-dessous.

```
allo-docteurs:  
  url: https://www.youtube.com/watch?v=rqoSKafN3aw  
  is_broca: yes  
  segments:  
    jean-dominique:  
      - [2:00, 5:40]  
      - [8:20, 10:02]  
      - [16:36, 18:00]
```

```

        - [20:36, 22:48]
        - [23:17, 23:39]
    raquel:
        - [10:20, 11:20]
        - [11:37, 11:52]
        - [12:24, 13:50]
hug:
    url: https://www.youtube.com/watch?v=d4Cybxw3sHk
    is_broca: yes
    segments:
        marie:
            - [34, 1:07]
            - [6:42, 6:47]
        jean-pierre:
            - [7:57, 8:57]

```

Une fois préparé, ce fichier est passé à un script Python qui se charge de télécharger les vidéos (en utilisant `youtube-dl`) et de les découper en segments. Ces segments sont ensuite transcrits en utilisant l’interface ligne de commande de Whisper.

5.2.2 Traduction automatique

Pour la création du corpus de la partie MT, nous avons interrogé l’API d’Open AI, plus précisément, le modèle `gpt-3.5-turbo` qui est celui derrière chatGPT (voir Extrait de code 5.1). Nous avons passé 3 paramètres à chatGPT :

- `messages` : une liste d’interactions décrivant l’histoire de la conversation. Chaque interaction est un dictionnaire contenant les clés `role` et `content`. La clé `role` représente l’identité de l’interlocuteur. Elle admet 3 valeurs : `user`, `bot` et `system`. Le message `system` est utilisé pour configurer le comportement de chatGPT pour le reste de la conversation.
- `timeout` : le temps maximal que chatGPT peut prendre pour générer une réponse (en secondes).
- `n` : le nombre de réponses à générer.

Cette opération peut échouer à cause des limitations de l’API d’Open AI. Si le nombre de requêtes dépasse le quota autorisé, l’API renvoie une erreur. Dans ce cas, nous attrapons l’erreur et nous réessayons après une seconde. Dans le cas contraire, nous passons au mot suivant. À la fin de la boucle, les erreurs sont enregistrées dans un fichier `errors.yaml`.

```

1 import os
2 import time
3
4 import openai
5 from yaml import Dumper, dump
6
7 openai.api_key = os.getenv("openai")
8
9 system_prompt = (
10     "You are a linguistics researcher. "

```

```

11     + "you are trying to understand the speaking patterns "
12     + "of people with aphasia. "
13     + "You will be given a french word. "
14     + "You must try to modify it in the same way "
15     + "a french speaker with aphasia would."
16     + "Your response should be a single word, "
17     + "with no punctuation, nor line breaks."
18 )
19
20 num_errors_per_word = 10
21 errors = []
22 for word in words:
23     prompt = (
24         f'Deforme le mot "{word}" a la facon dont le ferait ,
25         + "un aphasique de Broca. Repond avec un seul mot."
26     )
27
28     success = False
29     ntries = 0
30     while not success:
31         try:
32             responses = openai.ChatCompletion.create(
33                 model="gpt-3.5-turbo",
34                 messages=[
35                     {"role": "system", "content": system_prompt},
36                     {"role": "user", "content": prompt},
37                 ],
38                 timeout=100,
39                 n=num_errors_per_word,
40             )
41         except openai.error.RateLimitError:
42             ntries += 1
43             print(f"Retry {ntries}")
44             time.sleep(1)
45             continue
46         success = True
47         errors[word] = list(
48             response_object["message"]["content"]
49             for response_object in responses["choices"]
50         )
51
52 with open("errors.yaml", "x", encoding="utf-8") as f:
53     dump(errors, f, Dumper, allow_unicode=True, sort_keys=False)

```

Extrait de code 5.1 – Génération des erreurs avec chatGPT.

Le fichier `errors.yaml` est combiné avec le corpus pour produire un corpus parallèle synthétique. Cette procédure est illustrée par l'extrait de code 5.2. Pour chaque phrase, les mots modifiables sont mis dans une liste `corruptable_words`. À partir de cette liste, une deuxième liste `corruptions` est générée qui contient toutes les combinaisons de modifications possibles. Ces modifications sont donc appliquées à la phrase d'origine pour produire des phrases corrompues. Les phrases corrompues (différentes de la phrase d'origine) sont ajoutées au corpus parallèle qui est retourné.

```

1 from itertools import product
2
3

```

```

4 def get_corrupted_sentences(corpus, errors):
5     result = {}
6     for sentence in corpus:
7         variants = []
8         corruptable_words = [
9             word
10            for word in errors # The keys of the errors dictionary
11            if word in sentence.split(" ") # Only those in the sentence
12        ]
13        corruptions = list(
14            product( # Cartesian product of all possible corruptions
15                *[
16                    [(word, word)] # Identity corruption
17                    +
18                    [(
19                        word, error) for error in errors[word]
20                    ] # Modification corruptions
21                    for word in corruptable_words
22                ]
23            )
24        )
25        for corruption in corruptions:
26            splt = sentence.split(" ")
27            for word, error in corruption:
28                splt[splt.index(word)] = error
29            if splt != sentence.split():
30                variants.append(" ".join(splt))
31        result[sentence] = set(variants)
32    return result

```

Extrait de code 5.2 – Création du corpus parallèle synthétique.

Il est important de souligner que les erreurs sont divisées en 3 parties (entraînement, validation et test) avant d'être combinées avec le corpus. Cela a pour but d'éviter le sur-apprentissage à cause de fuites de données, c'est-à-dire que les règles de modification soient les mêmes dans les 3 parties même si les phrases sont différentes.

5.3 Création du modèle de traduction automatique

La section précédente fournit une vue de la procédure de création d'un corpus parallèle pour la MT. Dans cette section, nous exploitons ce corpus pour créer et entraîner un modèle de NMT.

5.3.1 Création du modèle

Comme discuté dans Section 5.1, nous avons choisi d'utiliser PyTorch et PyTorch Lightning pour la partie DL. La classe `Transformer` qui représente notre modèle hérite donc de la classe `lightning.pytorch.LightningModule` qui elle-même hérite de la classe `torch.nn.Module`. L'Extrait de code 5.3 montre la méthode d'initialisation de cette classe.

```

1 def __init__(
2     self,

```

```

3     d_model: int,
4     nhead: int,
5     source_vocab_size: int,
6     target_vocab_size: int,
7     source_pad_idx: int,
8     num_encoder_layers: int,
9     num_decoder_layers: int,
10    max_len: int,
11    dim_feedforward: int,
12    dropout: float,
13    lr: float,
14  ):
15     super().__init__()
16
17     self.input_embedding = nn.Embedding(source_vocab_size, d_model)
18     self.input_position_embedding = nn.Embedding(max_len, d_model)
19
20     self.output_embedding = nn.Embedding(target_vocab_size, d_model)
21     self.output_position_embedding = nn.Embedding(max_len, d_model)
22
23     self.transformer = nn.Transformer(
24         d_model,
25         nhead,
26         num_encoder_layers,
27         num_decoder_layers,
28         dim_feedforward,
29         dropout,
30     )
31
32     self.linear = nn.Linear(d_model, target_vocab_size)
33     self.dropout = nn.Dropout(dropout)
34     self.source_pad_idx = source_pad_idx
35
36     self.lr = lr

```

Extrait de code 5.3 – Méthode d’initialisation d’un transformeur.

La méthode `__init__` prend en argument les hyperparamètres du modèle (`d_model`, `nhead`, `num_encoder_layers`, `num_decoder_layers`, `dim_feedforward` et `dropout`) ainsi que des paramètres de configuration de l’entraînement et de l’inférence (`source_vocab_size`, `target_vocab_size`, `source_pad_idx`, `max_len` et `lr`).

L’appelle à la méthode `super().__init__()` a l’effet de construire par défaut un `LightningModule`. Les lignes qui suivent cette instruction permettent d’initialiser ce module en définissant les couches qui le composent. Parmi ces couches, la plus importante est `self.transformer`, un objet de la classe `nn.Transformer`. Tous les autres modules sont des couches de prétraitement (encodage positionnel) ou de post-traitement (projection linéaire). La Figure 5.2 montre le graphe de calcul du modèle résultant.

En utilisant les valeurs proposées dans le Chapitre 3.3 pour les hyperparamètres :

$$\begin{aligned}d_{\text{model}} &= d_{\text{FFN}} = 64 \\N_{\text{head}} &= 4 \\N_{\text{encoder}} &= N_{\text{decoder}} = 3\end{aligned}$$

nous obtenons un modèle dont la Table 5.2 résume les paramètres par couche.

	Name	Type	Params
0	input_embedding	Embedding	320 K
1	input_position_embedding	Embedding	6.4 K
2	output_embedding	Embedding	320 K
3	output_position_embedding	Embedding	6.4 K
4	transformer	Transformer	201 K
5	transformer.encoder	TransformerEncoder	75.8 K
6	transformer.decoder	TransformerDecoder	126 K
7	linear	Linear	325 K
8	dropout	Dropout	0

1.2 M Paramètres entraînables
 0 Paramètres non entraînables
 1.2 M Paramètres totaux
 4.719 Taille totale estimée (en Mo)

TABLE 5.2 – Résumé du modèle.

5.3.2 Entraînement

Pour entraîner le modèle, un objet *entraîneur* de la classe `lightning.pytorch.Trainer` est utilisé. Pour configurer sans comportement, plusieurs paramètres sont utilisés (voir Extrait de code 5.4

- `max_epochs` : le nombre maximal des passes sur le corpus d’entraînement.
- `deterministic` : un indicateur booléen, s’il est mis à `vrai`, des algorithmes déterministes sont utilisés pour la reproductibilité.
- `logger` : le journaliseur à utiliser pour garder trace des métriques. `WandbLogger` utilise *Weights & Biases* pour la journalisation.
- `callbacks` : une liste de rappels de fonctions à effectuer à la fin de chaque époque. `early_stopping` implémente l’arrêt précoce et `checkpoint` permet de sauvegarder le meilleur modèle.
- `gradient_clip_val` : un majorant sur la norme du vecteur gradient. Son but est d’éviter l’explosion des gradients.

Pour lancer l’entraînement, il suffit d’appeler la méthode `fit` de l’entraîneur comme suit : `trainer.fit(model, datamodule=dm)`.

```

1 trainer = Trainer(
2     max_epochs=epochs,
3     deterministic=True,
4     logger=WandbLogger(project="project-name")
5     callbacks=[early_stopping, checkpoint],
6     gradient_clip_val=1.0,
7 )

```

Extrait de code 5.4 – Creation de l’entraîneur.

Le paramètre `datamodule` passé à la méthode `Trainer.fit` est un objet de classe `lightning.pytorch.LightningDataModule`. Il s’agit d’une classe d’utilité qui implé-

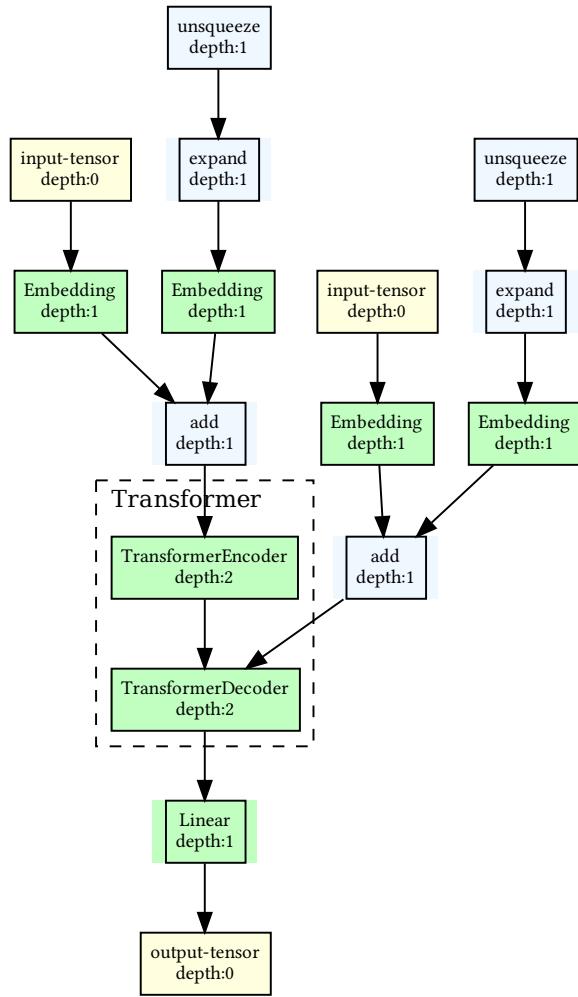


FIGURE 5.2 – Graphe de calcul du modèle défini dans la classe `Transformer` (profondeur = 2).

mente des fonctions de chargement de données. Ses 3 méthodes `train_dataloader`, `val_dataloader` et `test_dataloader` retournent respectivement les chargeurs d’entraînement, de validation et de test. L’entraîneur se charge de les appeler au moment opportun.

5.3.3 Réglage des hyperparamètres

Pour cette tâche, nous avons utilisé *Sweeps*, un outil fourni par Weights & Biases. Pour l’utiliser, il faut créer une configuration qui définit l’espace et la stratégie de recherche pour les hyperparamètres. Trois stratégies de recherche sont offertes :

- Grid : une recherche exhaustive sur l’espace des possibilités. Cette méthode garantit l’optimalité du résultat, mais elle est trop coûteuse et ne marche qu’avec un espace

de recherche fini.

- Random : une recherche aléatoire sur l'espace de possibilité. Cette méthode ne donne pas de garanties sur la qualité des hyperparamètres qu'elle produit. Cependant, elle est beaucoup plus rapide que la recherche exhaustive (car il est possible de définir le nombre des combinaisons à explorer) et elle peut être utilisée sur un espace infini.
- Bayesian : une méthode d'amélioration itérative. Sa première itération est identique à la recherche aléatoire, mais après elle utilise l'information sur le dernier essai pour informer les choix des paramètres du suivant. Elle converge généralement vers la solution optimale.

Pour définir l'espace de recherche, un dictionnaire ou un fichier `yaml` est utilisé. L'objet de configuration possède 3 clés obligatoires :

- `method` : la stratégie de recherche.
- `metric` : la métrique à optimiser. Il faut fournir le nom de la métrique, le mode d'optimisation (`maximize` ou `minimize`) et éventuellement une cible à atteindre.
- `parameters` : un dictionnaire qui contient la définition de l'espace des paramètres. Chaque paramètre est défini par un couple clé-valeur. La clé est le nom du paramètre et la valeur est un dictionnaire qui contient les informations suivantes :
 - Si la valeur du paramètre est fixée, il suffit de fournir sa valeur pour la clé `value`.
 - Si le paramètre a un nombre fini de valeurs possibles, il faut fournir la liste de ces valeurs pour la clé `values` et éventuellement sa loi de probabilité pour la clé `probabilities`. Si cette dernière n'est pas fournie, la loi uniforme est utilisée.
 - Si le paramètre est continu, il faut fournir une loi de probabilité pour la clé `distribution` en donnant le nom de la loi et ses paramètres.

Ci-dessous un exemple de configuration pour une recherche bayésienne. La métrique à maximiser est `val/bleu_score` et sa valeur cible est de 99. Les paramètres `lr` et `dropout` sont continus et suivent une loi log-uniforme sur les intervalles $[10^{-5}, 10^{-1}]$ et $[0.1, 0.5]$ respectivement. Le paramètre `batch_size` est fixé à 256.

```
method: bayes
metric:
  name: val/bleu_score
  goal: maximize
  target: 99
parameters:
  lr:
    distribution: log_uniform_values
    min: 1.e-5
    max: 1.e-1
  dropout:
    distribution: log_uniform_values
```

```

min: 0.1
max: 0.5
batch_size:
value: 256

```

Une fois la configuration définie, il suffit de créer une Sweep en appelant la méthode `wandb.sweep`, puis de lancer la recherche en appelant la méthode `wandb.agent`. Cette méthode prend en paramètre l’identifiant de la Sweep, une fonction qui définit l’entraînement et le nombre d’essais à effectuer (voir Extrait de code 5.5).

```

1 with open("config/sweep.yaml") as cfg:
2     sweep_cfg = load(cfg, Loader)
3
4 sweep_id = wandb.sweep(sweep_cfg, project="project-name")
5 wandb.agent(sweep_id, train, count=10)

```

Extrait de code 5.5 – Création et lancement d’une Sweep.

5.4 Interface utilisateur

Dans l’absence d’un système d’ASR, il est impossible de créer un système text-to-speech complet. Une autre interface est donc nécessaire pour utiliser le modèle. Pour cette fin, nous avons développé une interface web de traduction.

Bumblebee

Entrer un texte aphasique et cliquer sur corriger.



FIGURE 5.3 – Interface web de traduction.

La Figure 5.3 montre l’interface que nous avons développée. Nous l’avons nommée *Bumblebee*¹. L’utilisateur peut saisir un texte aphasique dans la zone de texte en haut et cliquer sur le bouton *Corriger*. La correction s’affiche dans la zone de texte en bas. La figure montre un exemple de correction pour la phrase : “*est-ce que cela fera une différence ?*” où le mot “*différence*” est remplacé par “*difféce*”.

1. Une référence à un personnage de la série *Transformers* qui souffre de mutisme, mais qui reprend la parole.

5.5 Conclusion

Dans ce chapitre, nous avons présenté les détails de la réalisation de notre solution. Nous y avons listé les outils, technologies et bibliothèques logicielles utilisés pour sa mise en place. Ensuite, nous nous sommes étalés sur les étapes menées pour accomplir cette tâche. Dans le chapitre suivant, nous présentons les résultats obtenus et nous les discutons.

Chapitre 6

Tests et résultats

Dans le chapitre précédent, nous avons présenté les détails de la réalisation de notre solution. Le présent chapitre est consacré à la présentation des résultats obtenus. Dans ce but, nous présentons les différents tests que nous avons effectués, les résultats obtenus et la signification de ces derniers.

6.1 Erreurs générées

Parmi les erreurs générées par chatGPT, celles qui ressemblent le plus à des erreurs humaines ont été manuellement sélectionnées. Le résultat de cette sélection est une liste de 217 mots avec une moyenne de 5 erreurs retenues par mot (1104 erreurs en termes absolus). Ces erreurs ont été analysées et classées en 4 catégories :

- des suppressions : de lettres ou de syllabes,
- des ajouts : de lettres ou de syllabes,
- des substitutions : de lettres ou de syllabes,
- des transpositions : de lettres ou de syllabes.

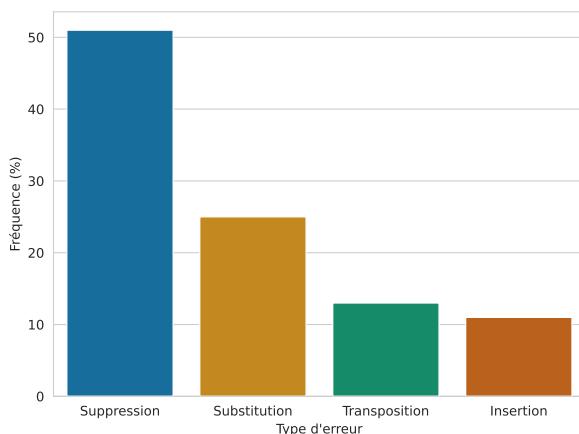


FIGURE 6.1 – Fréquences des catégories d’erreurs

Les fréquences de ces erreurs (pour les 32 premiers mots qui ont 327 modifications) sont présentées dans la Figure 6.1. Sur la base de ces fréquences, nous avons créé une fonction qui génère, pour un mot donné, des erreurs qui suivent les mêmes fréquences (voir le code 6.1).

```

1 def corrupt_word(
2     word,
3     p_remove=0.51,
4     p_substitute=0.25,
5     p_transpose=0.13,
6     p_insert=0.11,
7     p_skip=0.5,
8     all_syllables=None,
9 ):
10    from random import seed, randint, random
11    from hyphen import Hyphenator
12
13    hyphenator = Hyphenator("fr_FR")
14    syls = hyphenator.syllables(word)
15
16    # skip words that are too short
17    if len(syls) < 3:
18        return word
19
20    # skip all words with probability p_skip
21    if random() < p_skip:
22        return word
23
24    # remove a syllable with probability p_remove
25    if random() < p_remove:
26        idx = randint(0, len(syls) - 1)
27        del syls[idx]
28
29    # substitute a syllable with probability p_substitute
30    if random() < p_substitute:
31        idx1 = randint(0, len(syls) - 1)
32        syls[idx] = choice(all_syllables)
33
34    # transpose two syllables with probability p_transpose
35    if random() < p_transpose:
36        idx1 = randint(0, len(syls) - 1)
37        idx2 = randint(0, len(syls) - 1)
38        syls[idx1], syls[idx2] = syls[idx2], syls[idx1]
39
40    # insert a syllable with probability p_insert
41    if random() < p_insert:
42        idx = randint(0, len(syls) - 1)
43        syls.insert(idx, choice(all_syllables))
44    return ''.join(syls)

```

Extrait de code 6.1 – Génération des erreurs pour un mot

Les erreurs générées par cette fonction sont similaires aux erreurs générées par chatGPT (par exemple, maintenant → temain | tenant, entendu → enten | tendu | tenendu.). Cependant, certaines parmi elles ne sont pas prononçables (par exemple, maintenant → nantmain, simplement → mentple). Pour cette raison, nous avons décidé de ne pas les utiliser dans le corpus. Cela étant dit, il nous paraît intéressant d'explorer des méthodes de

filtrage de ces erreurs. Si réussies, elles permettent de générer des erreurs plus rapidement et plus facilement que par chatGPT.

6.2 Corpus créé

Après avoir construit le corpus parallèle en suivant la démarche décrite dans Sections 4.1 et 5.2.2, il est nécessaire d'évaluer sa qualité. Pour cela, nous avons choisi deux métriques : le score BLEU (voir Section 3.1.1) et la perplexité.

6.2.1 Perplexité

La perplexité est une mesure de la qualité d'un modèle de langue par rapport à un corpus. Si la qualité du modèle est connue (et bonne), la perplexité est une mesure de la qualité du corpus. La perplexité du corpus \mathcal{C} par rapport au modèle de langue \mathcal{M} est définie comme la moyenne géométrique des probabilités des phrases du corpus (voir l'équation 6.1).

$$\text{perplexité}(\mathcal{C}, \mathcal{M}) = \prod_{s \in \mathcal{C}} \mathcal{M}(s)^{-\frac{1}{|\mathcal{C}|}} \quad (6.1)$$

En prenant le logarithme de la perplexité, on retrouve l'entropie croisée de la loi de probabilité donnée par \mathcal{M} et celle induite par \mathcal{C} ¹. Elle mesure ainsi la dissimilarité entre ces deux lois (voir Section 3.1.1). La perplexité des phrases générées (pour simuler l'aphasie de Broca) est une mesure de la dissimilarité entre ses phrases et le français courant.

	french	aphasia
gpt-2	392.07	566.07
gpt-2-large	182.20	417.30
gpt-fr-cased-small	147.84	1357.31
gpt-fr-cased-base	123.93	1023.12

TABLE 6.1 – Perplexité des phrases du corpus par rapport à différents modèles de langue.

Pour calculer la perplexité, nous avons utilisé quatre modèles de langage basés sur GPT-2 et hébergés sur Hugging Face Hub :

- **gpt2** : ce modèle compte 124 M de paramètres et a été entraîné sur 40 Go de texte en plusieurs langues.
- **gpt2-large** : ce modèle compte 774 M de paramètres et a été entraîné sur le même corpus que **gpt2**.
- **gpt-fr-cased-small** : ce modèle compte 124 M de paramètres et a été entraîné sur un corpus de textes en français d'une taille non précisée.
- **gpt-fr-cased-base** : ce modèle compte 1.017 G de paramètres et a été entraîné sur le même corpus que **gpt-fr-cased-small**.

1. À une constante multiplicative dépendante de la base du logarithme près

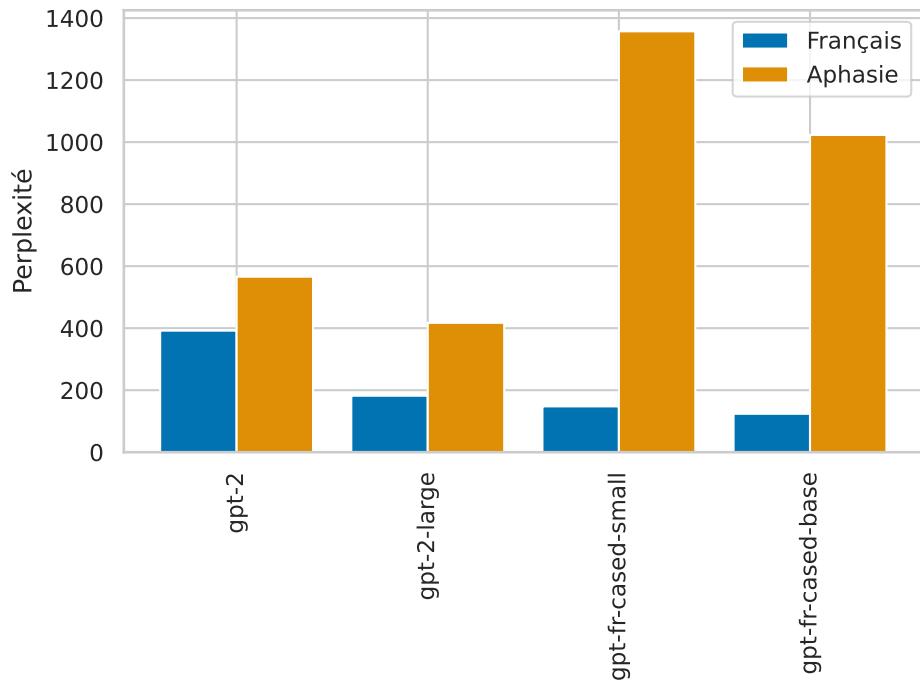


FIGURE 6.2 – Perplexité des phrases du corpus par rapport à différents modèles de langue.

Le code utilisé pour calculer la perplexité est donné par l’Extrait de code 6.2. Ce code a été exécuté sur Google Colaboratory avec une instance GPU munie de 16 Go de mémoire vive.

```

1 import pandas as pd
2 import evaluate
3
4 # Load perplexity metric
5 perplexity = evaluate.load("perplexity", module_type="measurement")
6
7 # Load data
8 df = pd.read_csv("val.csv")
9 fr = df.french.tolist()
10 aph = df.aphasia.tolist()
11
12 # Compute perplexities
13 fr = perplexity.compute(
14     model_id='asi/gpt-fr-cased-base',
15     add_start_token=False,
16     data=fr
17 )
18 aph = perplexity.compute(
19     model_id='asi/gpt-fr-cased-base',
20     add_start_token=False,
21     data=aph
22 )
23
24 print(f'{fr["mean_perplexity"]:.2f}, {aph["mean_perplexity"]:.2f}')

```

Extrait de code 6.2 – Calcul de la perplexité avec le modèle gpt-fr-cased-base.

Les résultats sont donnés par Figure 6.2 et Table 6.1. On y observe que tous les modèles attribuent une perplexité plus faible aux phrases en français qu'à celles qui simulent l'aphasie de Broca. On note également que cette différence est plus prononcée pour les modèles entraînés sur des textes en français.

Les différences données par ces derniers sont compatibles avec les résultats obtenus par (GHUMMAN, 2021) avec des transcriptions de patients aphasiques et un groupe de contrôle. Il est raisonnable d'en conclure que les phrases générées sont similaires à celles produites dans le cas d'une aphasie de Broca.

6.2.2 Score BLEU

Le problème avec l'utilisation de la perplexité pour mesurer la différence entre les phrases générées et le langage ordinaire, est qu'elle ne prend pas en compte la relation entre une phrase aphasique et la phrase correcte qui lui correspond. Elle traite les deux corpus comme étant indépendants. BLEU est une métrique qui prend en compte cette relation (voire Section 3.1.1). En prenant la moyenne des scores BLEU de chaque couple de phrases dans le corpus parallèle, nous obtenons un seul nombre qui mesure la similarité entre les deux parties du corpus.

La fonction `bleu_score` de `torchmetrics` prend en entrée une liste de phrases qui représentent les traductions candidates (dans notre cas, les phrases aphasiques) et une liste de listes de phrases qui représentent les traductions de référence (dans notre cas, les phrases en français). Un score BLEU de 62.72% a été obtenu sur le corpus de validation.

6.3 Entraînement du modèle

Tous les tests présentés dans le reste de ce chapitre ont été effectués sur un ordinateur portable équipé d'un processeur **Intel Core i9-8950HK** et d'une carte graphique **NVIDIA GeForce GTX 1050 Ti**. Cette machine dispose de 32 Go de mémoire vive et de 4 Go de mémoire vidéo. Le système d'exploitation est **Ubuntu 20.04 LTS**, la version de Python utilisée est **3.10.6** et celle de CUDA est **11.6.124**.

L'entraînement du modèle a été effectué en suivant la procédure illustrée par la figure 6.5. Le modèle est d'abord entraîné avec un choix d'hyperparamètres plus ou moins arbitraire pour vérifier que l'entraînement se déroule correctement. Ensuite, une recherche d'hyperparamètres est effectuée pour trouver les meilleurs hyperparamètres. Cette recherche est effectuée uniquement si l'entraînement initial n'a pas donné des résultats satisfaisants.

6.3.1 Choix des hyperparamètres

Pour notre premier test, nous avons entraîné le modèle pour 8 époques (cela a pris 58 min 25 s). Les hyperparamètres ont été choisis comme suit :

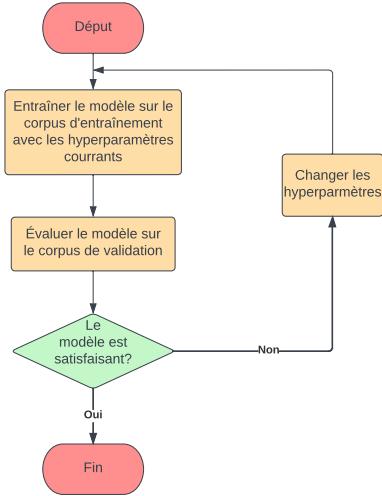


FIGURE 6.3 – Organigramme de la phase d'entraînement

- Dimension de plongement : 64
- Dimension de la couche cachée : 64
- Taux d'apprentissage : 3×10^{-4}
- Dropout : 0.1
- Nombre de couches de l'encodeur/décodeur : 3
- Nombre de têtes d'attention : 4
- Norme maximale des plongements : 1
- Taille de lot : 256
- Norme maximale du gradient : 1
- Coefficient de régularisation L_2 : 10^{-4}
- Nombre de processus pour le chargement des données : 4

6.3.2 Entrainement initial

L'objectif de ce test est de vérifier que le modèle fonctionne correctement. Cela inclut la vérification du bon déroulement du chargeur de données, des passes forward et backward et l'absence de sur-apprentissage. La probabilité de ce dernier point n'est pas négligeable dans notre cas, car le modèle est grand en comparaison avec la complexité de la procédure de génération des données.

Pour chaque époque, nous avons calculé la moyenne de la fonction de perte, de l'exactitude et du score BLEU sur le corpus de validation et sur le corpus d'entraînement. Les résultats de ce test sont présentés dans la Figure 6.4. Ces résultats montrent que l'entraînement ne présente pas d'anomalies. Les 11 premières époques se déroulent sans erreurs et les métriques évoluent comme prévu. Cependant, la Figure 6.4c montre des signes clairs

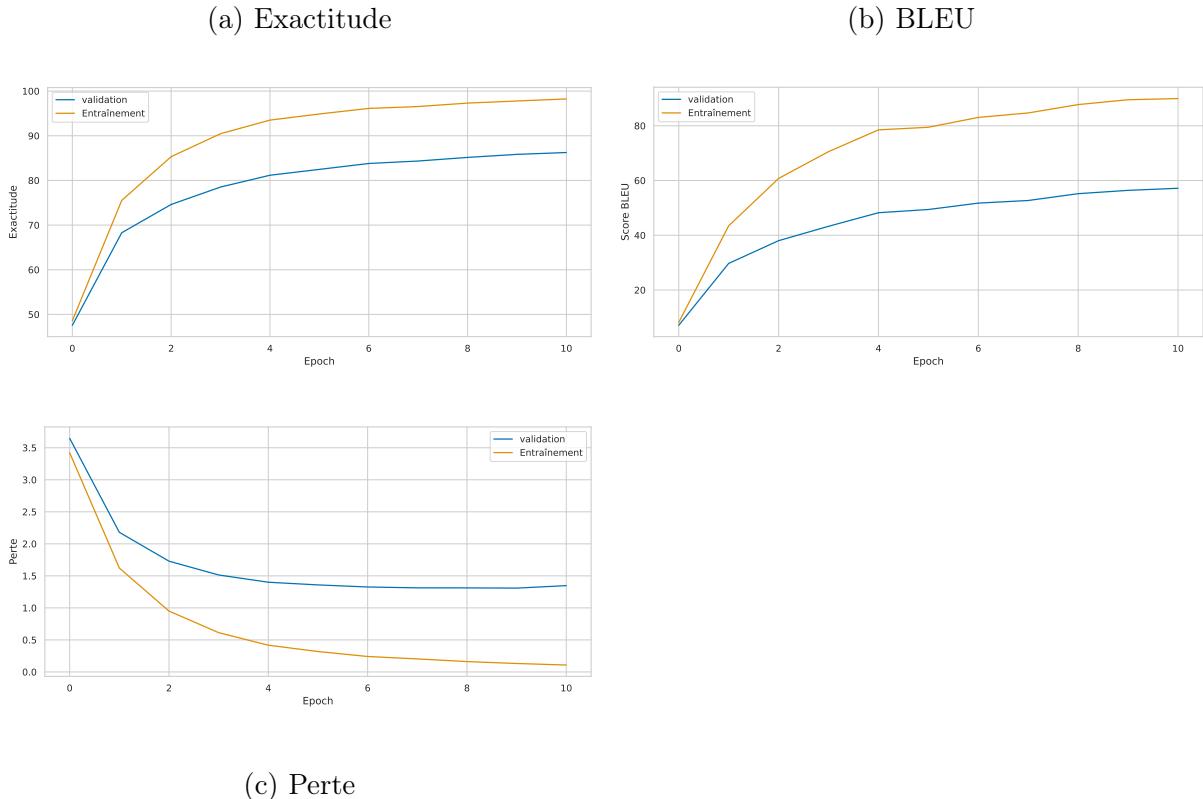


FIGURE 6.4 – Évolution des métriques au cours de l’entraînement.

de sur-apprentissage. L’écart entre les courbes d’entraînement et de validation est très important.

Comme évoqué précédemment, ce résultat n’est pas surprenant. En effet, le modèle peut apprendre les règles de substitution utilisées pour générer les erreurs plutôt que la correspondance entre la phrase d’origine et la phrase erronée. Il est donc nécessaire de trouver une façon de forcer le modèle à apprendre la correspondance entre les phrases.

6.3.3 Entrainement avec les erreurs masquées

Pour remédier au problème de sur-apprentissage rencontré lors de l’entraînement initial, nous avons bruité les données d’entraînement. Pour empêcher le modèle d’apprendre les règles de substitution, nous avons masqué les erreurs en les remplaçant par le token [UNK]. Cette technique est inspirée du MLM de BERT (DEVLIN et al., 2019) et de l’auto-encodage de BART (LEWIS et al., 2019).

Les résultats sont présentés dans la Figure 6.5. Au bout de 8 époques, la perte sur le corpus d’entraînement est de 0.09. L’exactitude est de 98.71% et le score BLEU est de 81.41%. Les résultats sur le corpus de validation sont comparables. Les mêmes métriques valent respectivement 0.16, 97.16% et 77.38%.

Les courbes de la Figure 6.5 montrent une forte correspondance entre les courbes d’entraînement et de validation. Cela suggère que le modèle ne sur-apprenne plus les

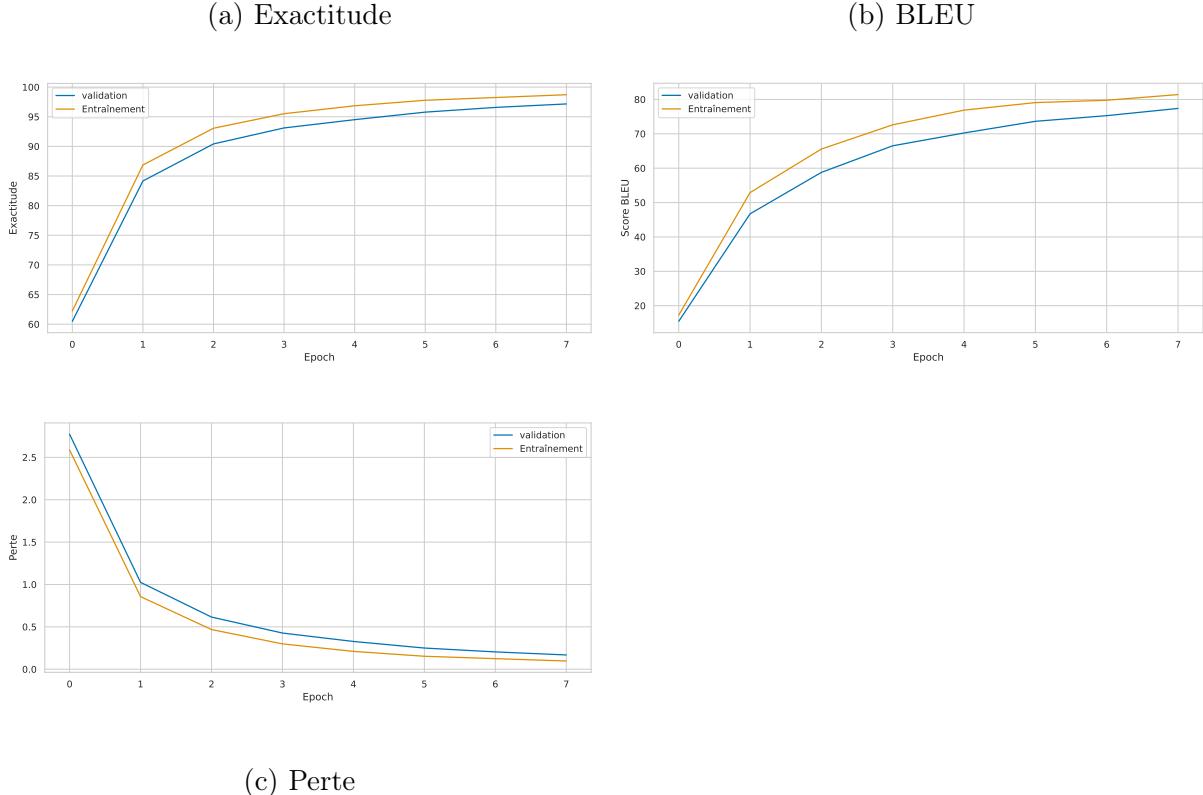


FIGURE 6.5 – Évolution des métriques au cours de l’entraînement.

erreurs. On note également que les pentes des trois courbes ne sont pas négligeables. Il est donc probable que le modèle puisse encore s’améliorer en augmentant le nombre d’époques.

6.4 Réglage des hyper-paramètres

Les résultats présentés dans Section 6.3.3 sont satisfaisants. Il n’est donc pas strictement nécessaire d’effectuer un réglage des hyperparamètres. Cependant, il est possible en le faisant d’obtenir des résultats comparables en moins d’époques ou avec un modèle plus petit. Pour cette raison, nous avons décidé de l’entamer.

6.4.1 Configuration

Nous avons fait le choix de restreindre la portée de ce réglage à deux hyperparamètres : le taux d’apprentissage (η ou `lr`) et le `(dropout)`. Nous avons opté pour une recherche bayésienne avec des lois log-uniformes sur les intervalles $[10^{-5}, 10^{-1}]$ et $[0.1, 0.5]$ respectivement. Les autres hyperparamètres ont été fixés à leurs valeurs données dans Section 6.3.1.

Pour chaque combinaison de `lr` et `dropout`, nous avons entraîné le modèle pendant 2 époques. L’objectif de la recherche est de maximiser le score BLEU sur le corpus de

validation. Nous avons limité le nombre d'essais à 20.

6.4.2 Résultats

Les 20 essais ont été effectués en 2 h 17 min 13 s 11 ms. Les résultats sont présentés dans Figure 6.6. Il s'agit d'une visualisation en coordonnées parallèles des résultats. L'axe

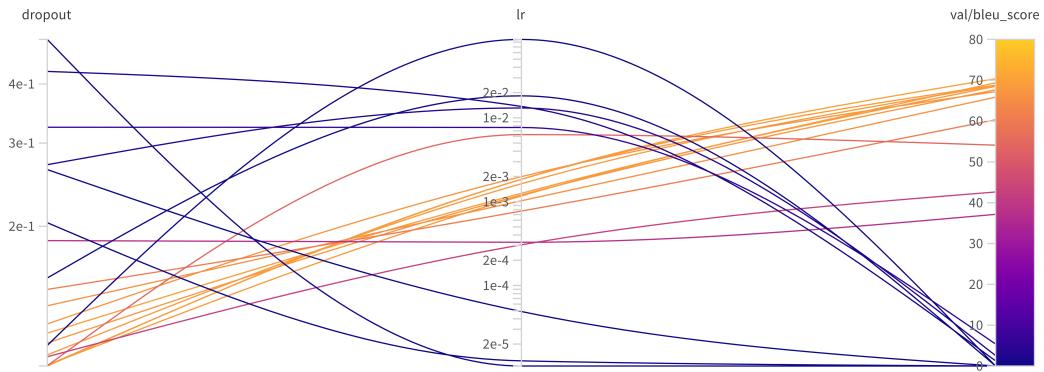


FIGURE 6.6 – Résultats de la recherche bayésienne des hyperparamètres.

de gauche représente la valeur de `dropout` sur une échelle linéaire, celui du milieu la valeur de `lr` sur une échelle logarithmique et celui de droite le score BLEU sur une échelle de 0 à 100. Un essai est représenté par une courbe qui relie les 3 points correspondants sur les 3 axes. La couleur de la courbe indique le score BLEU associé. Elle est interpolée entre le violet (0%) et l'orange (100%). Les mêmes informations sont présentes dans Table 6.2 sous forme numérique. La Figure 6.7 représente l'évolution temporelle du score BLEU de validation.

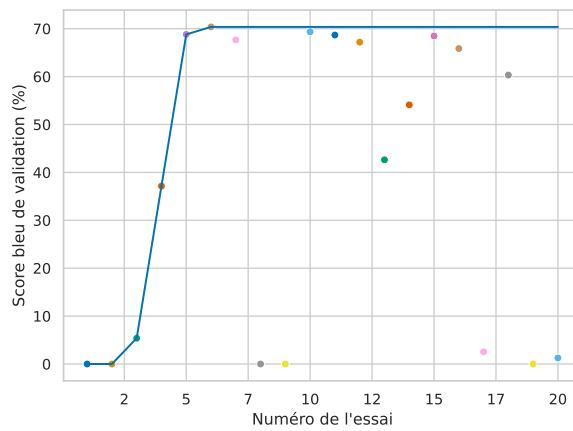


FIGURE 6.7 – Évolution du score BLEU au cours du réglage des hyperparamètres.

On observe sur la Figure 6.6 un regroupement des lignes oranges (les meilleurs essais) dans la région qui correspond à $\text{dropout} \in [0.1, 0.15]$ et $\eta \approx 10^{-3}$. Cela est aussi apparent

dropout	lr	bleu_score(%)
0.263684	0.000049	0.000000
0.155497	0.018280	0.000000
0.324059	0.007670	5.386821
0.186584	0.000328	37.148937
0.101305	0.001844	68.802765
0.101600	0.001849	70.371147
0.113328	0.001201	67.655579
0.111991	0.086200	0.000000
0.496975	0.000011	0.000000
0.118870	0.001259	69.372551
0.106921	0.001636	68.679092
0.124348	0.001989	67.192963
0.105984	0.000305	42.627316
0.101419	0.006308	54.094482
0.101742	0.001170	68.488480
0.135847	0.001025	65.854347
0.270016	0.013106	2.553007
0.147102	0.000772	60.324165
0.203584	0.000013	0.000000
0.425286	0.013681	1.264920

TABLE 6.2 – Résultats de la recherche bayésienne des hyperparamètres.

	Importance	Corrélation
dropout	63.9%	-0.899
lr	36.1%	-0.647

TABLE 6.3 – Importance et corrélation des hyperparamètres.

sur le tableau. Une analyse statistique sur les essais effectués nous permet d'estimer l'importance des hyperparamètres² ainsi que leurs corrélations avec le score BLEU. Ils sont tous deux négativement corrélés avec ce dernier (BLEU augmente quand l'un diminue). Le `dropout` est le plus important entre les deux (voir Table 6.3 et Figure 6.8). La meilleure valeur du score BLEU est obtenue avec $\text{dropout} \approx 0.101599$ et $\eta \approx 0.0018488801$. Pour cette valeur, le score BLEU sur le corpus de validation est de 70.37% après 2 époques.

6.5 Entraînement avec les hyperparamètres optimaux

Après le réglage des hyperparamètres, nous avons entraîné le modèle en spécifiant un nombre maximal d'époques de 20. L'entraînement s'est arrêté après 9 époques à cause du

2. Un foret aléatoire est entraîné pour prédire le score BLEU en fonction des hyperparamètres. L'importance d'un hyperparamètre est mesurée par rapport à ce foret (« Parameter Importance | Weights & Biases Documentation », s. d.).

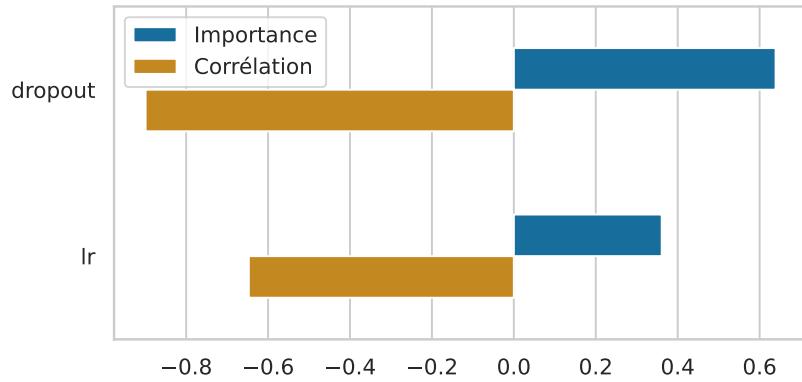


FIGURE 6.8 – Importance des hyperparamètres et corrélation avec le score BLEU.

rappel de fonction `EarlyStopping`, car le score BLEU sur le corpus de validation n'a pas augmenté pendant 3 époques consécutives. L'exécution a duré 1 h 26 min 27 s.

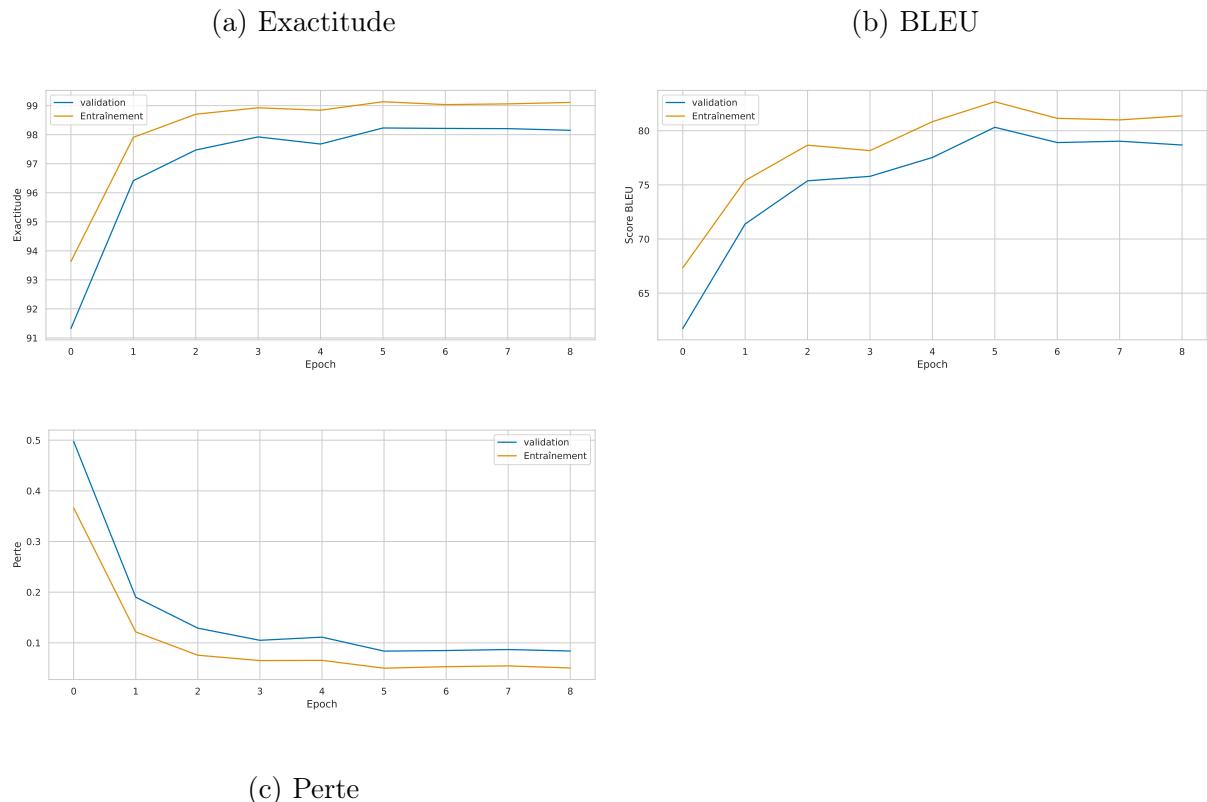


FIGURE 6.9 – Évolution des métriques avec les hyperparamètres optimaux.

Les courbes d'apprentissage sont présentées sur la Figure 6.9. On remarque que les métriques d'entraînement et de validation sont très fortement corrélées. Il est donc improbable que le modèle sur-apprenne sur le corpus d'entraînement. La meilleure valeur du score BLEU a été obtenue après 6 époques et elle vaut 80.32%.

Sur le corpus de test, le score BLEU est de 79.61%, l'exactitude est de 98.15% et la perte de $8.39 \cdot 10^{-2}$. Cela indique que le modèle généralise bien et qu'il n'a pas surappris sur

les corpus d’entraînement et de validation. Comparé à la valeur de base présentée dans la Section 6.2, le score BLEU a augmenté de 16.89%. Une amélioration de 2.22% a été obtenue par rapport au modèle entraîné avec les hyperparamètres par défaut.

6.6 Conclusion

Dans ce chapitre, nous avons présenté les résultats que nous avons obtenus à l’issue de notre travail. Nous avons obtenu une liste de 1104 erreurs pour 217 mots. Des statistiques sur la nature des erreurs et les mots concernés ont été présentées.

Ensuite, nous avons décrit le corpus que nous avons construit à partir de ces erreurs. Il compte 282 k couple de phrases avec une grande différence de perplexité entre les phrases correctes et les phrases erronées. Or, le score BLEU initial est plus élevé que celui atteint par la majorité des modèles de traduction.

L’entraînement d’un modèle de traduction sur ce corpus a montré des signes de surapprentissage. Un deuxième tour d’entraînement masqué a donné des résultats satisfaisants (un score BLEU de 77.38%). Cependant, nous avons fait une recherche d’hyperparamètres qui a réussi à améliorer encore les résultats. Une deuxième phase d’entraînement a été effectuée avec les meilleurs hyperparamètres qui a donné des résultats encore meilleurs (un score BLEU de 79.61%).

Conclusion générale

L'aphasie de Broca est un trouble de communication qui touche une partie grandissante de la population mondiale. Faisant souvent suite à un AVC, l'aphasie de Broca affecte le décodage des mots et la production de la parole. Elle diminue ainsi mesurablement la qualité de vie des personnes qui en sont atteintes (CHAPEY, 2008 ; FEIGIN et al., 2022 ; ROSS & WERTZ, 2010).

Le traitement le plus courant de l'aphasie de Broca est la rééducation de la parole par un orthophoniste. En dépit d'être un traitement efficace, il est gourmand en temps, en argent et en ressources humaines, ce qui en fait un traitement peu accessible pour la majorité des personnes atteintes. Ce manque d'accessibilité, combiné à la gravité de certaines conséquences de l'aphasie de Broca, rend urgent le développement de solutions alternatives (da FONTOURA et al., 2012 ; FLOWERS et al., 2016).

Les méthodes informatiques, notamment les techniques de ML et de NLP, semblent avoir le potentiel de réduire les coûts matériels et humains associés à la rééducation de la parole. Elles peuvent ainsi faciliter l'accès au traitement de l'aphasie de Broca (MISRA et al., 2022 ; QIN et al., 2022 ; SMAÏLI et al., 2022).

Dans ce projet de fin d'étude, nous avons exploré la possibilité d'utiliser la MT et l'ASR, deux techniques de NLP basées sur l'apprentissage S2S pour aider les personnes atteintes de l'aphasie de Broca.

Nous avons d'abord introduit l'aphasie pour familiariser le lecteur avec ses causes, sa portée, ses effets, les traitements disponibles et les défis auxquels ils sont confrontés. Après cela, nous avons présenté la modélisation S2S, le cadre général de la MT et de l'ASR. Nous avons posé le problème et présenté les solutions que nous avons évaluées et comparées. Le résultat de cette comparaison est une supériorité nette du transformeur sur les autres modèles. Nous avons alors exploré dans le troisième chapitre les différentes publications qui ont étudié l'utilisation du transformeur pour la résolution de problèmes de MT et d'ASR.

Ensuite, nous avons présenté la conception d'un système qui combine un modèle de traduction avec un modèle d'ASR pour corriger la parole aphasique. Conformément à cette conception, la réalisation de ce système a été entamée avec Python et PyTorch. En conclusion, les résultats de notre travail ont été présentés. Il s'agit d'un modèle de correction avec une interface web à base de texte pour la partie MT, et d'un corpus annoté manuellement pour la partie ASR. Le modèle de traduction a un score BLEU de 79.61% et le corpus d'ASR contient 48 min 44 s d'enregistrements.

Perspectives et horizons de recherche futurs

En dépit d'être encourageants, les résultats de ce projet de fin d'étude sont loin d'être complets. Plusieurs pistes de recherche peuvent être explorées pour obtenir de meilleurs résultats.

Parmi les axes d'amélioration les plus prometteurs, nous pouvons citer la taille des corpus d'entraînement. La collecte de plus de données peut permettre d'entraîner les modèles sans besoin de données synthétiques. Le remplacement de chatGPT par un modèle moins coûteux comme LLAMA (**Touvron_Lavril_Izacard_Martinet_Lachaux_Lacroix_RoziaLre** et Alpaca (ZHANG et al., 2023) pour la synthèse des erreurs peut permettre d'utiliser un corpus synthétique de tailles plus importantes.

Une autre piste qui mérite d'être explorée est l'utilisation d'un modèle pré-entraîné comme BART ou GPT pour la traduction et Whisper pour l'ASR. Ces modèles peuvent être utilisés directement ou affinés sur un corpus relativement petit pour avoir de meilleurs résultats.

Bibliographie

- (s. d.). <https://www.who.int/publications-detail-redirect/WHO-HIS-HSI-Rev.2012.03>
- (s. d.). <https://www.acqol.com.au/instruments>
- About Python. (2022). <https://pythoninstitute.org/about-python>
- ACHARYA, A. B., & WROTON, M. (2022). Broca Aphasia. In *StatPearls*. StatPearls Publishing. <http://www.ncbi.nlm.nih.gov/books/NBK436010/>
- ALMEIDA, F., & XEXÉO, G. (2019). Word Embeddings : A Survey [arXiv :1901.09069 [cs, stat]], (arXiv :1901.09069). <http://arxiv.org/abs/1901.09069>
- ART, S. M. (2013). *English : Functional areas of the brain*. <https://commons.wikimedia.org/wiki/File:Cerveau.jpg>
- BA, J. L., KIROS, J. R., & HINTON, G. E. (2016). Layer Normalization [arXiv :1607.06450 [cs, stat]], (arXiv :1607.06450). <http://arxiv.org/abs/1607.06450>
- BAHDANAU, D., CHO, K., & BENGIO, Y. (2016). Neural Machine Translation by Jointly Learning to Align and Translate [arXiv :1409.0473 [cs, stat]], (arXiv :1409.0473). <https://doi.org/10.48550/arXiv.1409.0473>
- BARBE, P., & LEDOUX, M. (2012). *Probabilité (L3M1)*. EDP Sciences.
- BASODI, S., JI, C., ZHANG, H., & PAN, Y. (2020). Gradient amplification : An efficient way to train deep neural networks. *Big Data Mining and Analytics*, 3(3), 196-207. <https://doi.org/10.26599/BDMA.2020.9020004>
- BENGIO, Y., SIMARD, P., & FRASCONI, P. (1994). Learning long-term dependencies with gradient descent is difficult. 5(2), 157-166. <https://doi.org/10.1109/72.279181>
- BISHOP, C. M. (2006). *Pattern Recognition and Machine Learning* [Google-Books-ID : qWPwnQEACAAJ]. Springer.
- BROCA, M. P. (1861). REMARQUES SUR LE SIÉGE DE LA FACULTÉ DU LANGAGE ARTICULÉ, SUIVIES D'UNE OBSERVATION D'APHÉMIE (PERTE DE LA PAROLE), 18.
- BRODMANN, K. (2007). *Brodmann's : Localisation in the Cerebral Cortex*. Springer Science & Business Media.
- BROWN, T., MANN, B., RYDER, N., SUBBIAH, M., KAPLAN, J. D., DHARIWAL, P., NEELAKANTAN, A., SHYAM, P., SASTRY, G., ASKELL, A., AGARWAL, S., HERBERT-VOSS, A., KRUEGER, G., HENIGHAN, T., CHILD, R., RAMESH, A., ZIEGLER, D., WU, J., WINTER, C., ... AMODEI, D. (2020). Language Models are Few-Shot Learners. *Advances in Neural Information Processing Systems*, 33, 1877-1901. <https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html>
- BRYANT, C., YUAN, Z., QORIB, M. R., CAO, H., NG, H. T., & BRISCOE, T. (2022). Grammatical Error Correction : A Survey of the State of the Art [arXiv :2211.05166 [cs]], (arXiv :2211.05166). <http://arxiv.org/abs/2211.05166>

- CALIN, O. (2020). *Deep Learning Architectures*. Springer. <https://link.springer.com/book/10.1007/978-3-030-36721-3>
- CHAN, S.-w. (2015). *Routledge Encyclopedia of Translation Technology*. Routledge, Taylor & Francis Group.
- CHAPEY, R. (2008). *Language Intervention Strategies in Aphasia and Related Neurogenic Communication Disorders*. Wolters Kluwer Health/Lippincott Williams & Wilkins.
- CHO, K., van MERRIENBOER, B., BAHDANAU, D., & BENGIO, Y. (2014). On the Properties of Neural Machine Translation : Encoder-Decoder Approaches [arXiv :1409.1259 [cs, stat]], (arXiv :1409.1259). <http://arxiv.org/abs/1409.1259>
- CHUNG, J., GULCEHRE, C., CHO, K., & BENGIO, Y. (2014). Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling [arXiv :1412.3555 [cs]], (arXiv :1412.3555). <http://arxiv.org/abs/1412.3555>
- CHURCH, K. W. (2017). Word2Vec. *Natural Language Engineering*, 23(1), 155-162. <https://doi.org/10.1017/S1351324916000334>
- CLINCHANT, S., JUNG, K. W., & NIKOULINA, V. (2019). On the use of BERT for Neural Machine Translation [arXiv :1909.12744 [cs]], (arXiv :1909.12744). <http://arxiv.org/abs/1909.12744>
- CNSA, C. n. d. s. p. l. (2015). session de sensibilisation. https://www.cnsa.fr/documentation/dossierpressefno_021015_bd.pdf
- COSTA-JUSSÀ, M. R., RAPP, R., LAMBERT, P., EBERLE, K., BANCHS, R. E., & BABYCH, B. (Éd.). (2016). *Hybrid Approaches to Machine Translation*. Springer International Publishing. <https://doi.org/10.1007/978-3-319-21311-8>
- COSTANZA, A., AMERIO, A., AGUGLIA, A., MAGNANI, L., SERAFINI, G., AMORE, M., MERLI, R.,AMBROSETTI, J., BONDOLFI, G., MARZANO, L., & BERARDELLI, I. (2021). "Hard to Say, Hard to Understand, Hard to Live" : Possible Associations between Neurologic Language Impairments and Suicide Risk. *Brain Sciences*, 11(12), 1594. <https://doi.org/10.3390/brainsci11121594>
- CS480/680 Lecture 19 : Attention and Transformer Networks*. (2019). https://www.youtube.com/watch?v=OyFJWRnt_AY
- da FONTOURA, D. R., RODRIGUES, J. d. C., CARNEIRO, L. B. d. S., MONÇÃO, A. M., & de SALLES, J. F. (2012). Rehabilitation of language in expressive aphasias : a literature review. *Dementia & Neuropsychologia*, 6(4), 223-235. <https://doi.org/10.1590/S1980-57642012DN06040006>
- DEVLIN, J., CHANG, M.-W., LEE, K., & TOUTA11A, K. (2019). BERT : Pre-training of Deep Bidirectional Transformers for Language Understanding [arXiv :1810.04805 [cs]], (arXiv :1810.04805). <http://arxiv.org/abs/1810.04805>
- FALCON, W., & THE PYTORCH LIGHTNING TEAM. (2019). *PyTorch Lightning* (Version 1.4). <https://doi.org/10.5281/zenodo.3828935>
- FATHI, S. (2021). *Recurrent Neural Networks*. <https://link.springer.com/book/10.1007/978-3-030-89929-5>
- FEIGIN, V. L., BRAININ, M., NORRVING, B., MARTINS, S., SACCO, R. L., HACKE, W., FISHER, M., PANDIAN, J., & LINDSAY, P. (2022). World Stroke Organization (WSO) : Global Stroke Fact Sheet 2022. *International Journal of Stroke : Official Journal of the International Stroke Society*, 17(1), 18-29. <https://doi.org/10.1177/17474930211065917>
- FLOWERS, H., SKORETZ, S., SILVER, F., ROCHON, E., FANG, J., FLAMAND-ROZE, C., & MARTINO, R. (2016). Poststroke Aphasia Frequency, Recovery, and Outcomes :

- A Systematic Review and Meta-Analysis. *Archives of Physical Medicine and Rehabilitation*, 97, 2188-2201. <https://doi.org/10.1016/j.apmr.2016.03.006>
- FODOR, J. A. (1983). *The Modularity of Mind* [Google-Books-ID : 0vg0AwAAQBAJ]. MIT Press.
- FUKUSHIMA, K. (1980). Neocognitron : A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4), 193-202. <https://doi.org/10.1007/BF00344251>
- GHUMAN, N. S. (2021). *Training and Probing Language Models for Discerning between Speech of People with Aphasia and Healthy Controls* (thèse de doct.). University of Georgia.
- GOLDHAHN, D., ECKART, T., & QUASTHOFF, U. (2012). Building Large Monolingual Dictionaries at the Leipzig Corpora Collection : From 100 to 200 Languages.
- HADJ, A. A. e. (2015). *Analyse syntaxique et traduction : outils et techniques cours et exercices résolus*. Ellipses.
- HALLOWELL, B. (2017). *Aphasia and Other Acquired Neurogenic Language Disorders : A Guide for Clinical Excellence*. Plural Publishing.
- HE, K., ZHANG, X., REN, S., & SUN, J. (2016). Deep Residual Learning for Image Recognition, 770-778. https://openaccess.thecvf.com/content_cvpr_2016/html/He_Deep_Residual_Learning_CVPR_2016_paper.html
- HOCHREITER, 9., & SCHMIDHUBER, J. (1997). Long short-term memory. 9(8), 1735-1780.
- INFORMATIK, F., BENGIO, Y., FRASCONI, P., & SCHMIDHUBER, J. (2003). Gradient Flow in Recurrent Nets : the Difficulty of Learning Long-Term Dependencies. *A Field Guide to Dynamical Recurrent Neural Networks*.
- ISLAM, K., ZAHEER, M. Z., & MAHMOOD, A. (2022). Face Pyramid Vision Transformer. *arXiv preprint arXiv :2210.11974*.
- JACOBS, M., & ELLIS, C. (2021). Estimating the cost and value of functional changes in communication ability following telepractice treatment for aphasia. *PLOS ONE*, 16(9), e0257462. <https://doi.org/10.1371/journal.pone.0257462>
- JDIFOOL, t. p., Mysid. (2006). *Français : Les principaux lobes du cerveau, vue latérale gauche. Inspiré de la figure 728 de Gray's Anatomy*. https://commons.wikimedia.org/wiki/File:Brain%5C_diagram%5C_fr.svg
- KALCHBRENNER, N., & BLUNSMON, P. (2013). Recurrent Continuous Translation Models. *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, 1700-1709. <https://aclanthology.org/D13-1176>
- KALCHBRENNER, N., ESPEHOLT, L., SIMONYAN, K., OORD, A. v. d., GRAVES, A., & KAVUKCUOGLU, K. (2017). Neural Machine Translation in Linear Time [arXiv :1610.10099 [cs]], (arXiv :1610.10099). <http://arxiv.org/abs/1610.10099>
- KAMEOKA, H., TANAKA, K., KWAŚNY, D., KANEKO, T., & NOBUKATSU, H. (2020). ConvS2S-VC : Fully Convolutional Sequence-to-Sequence Voice Conversion. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 28, 1849-1863. <https://doi.org/10.1109/TASLP.2020.3001456>
- KEARNS, M. J., & VAZIRANI, U. (1994). *An Introduction to Computational Learning Theory* [Google-Books-ID : vCA01wY6iywC]. MIT Press.
- KINGMA, D. P., & BA, J. (2017). Adam : A Method for Stochastic Optimization [arXiv :1412.6980 [cs]], (arXiv :1412.6980). <http://arxiv.org/abs/1412.6980>
- LAROCHELLE, H., & HINTON, G. E. (2010). Learning to combine foveal glimpses with a third-order Boltzmann machine. *Advances in Neural Information Processing Sys-*

- tems*, 23. <https://proceedings.neurips.cc/paper/2010/hash/677e09724f0e2df9b6c000b75b5da10d-Abstract.html>
- LAROUSSE. (s. d.). <https://www.larousse.fr/dictionnaires/francais/aphasie/4448>
- LASKA, A. C., HELLBLOM, A., MURRAY, V., KAHAN, T., & VON ARBIN, M. (2001). Aphasia in acute stroke and relation to outcome. *Journal of Internal Medicine*, 249(5), 413-422. <https://doi.org/10.1046/j.1365-2796.2001.00812.x>
- Le tutoriel Python. (2023). <https://docs.python.org/3/tutorial/index.html>
- LECUN, Y., BOSER, B., DENKER, J. S., HENDERSON, D., HOWARD, R. E., HUBBARD, W., & JACKEL, L. D. (1989). Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, 1(4), 541-551. <https://doi.org/10.1162/neco.1989.1.4.541>
- LECUN, Y., BOTTOU, L., BENGIO, Y., & HAFFNER, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 2278-2324. <https://doi.org/10.1109/5.726791>
- LECUN, Y., BENGIO, Y., & HINTON, G. (2015). Deep learning. *Nature*, 521(7553), 436-444. <https://doi.org/10.1038/nature14539>
- LEWIS, M., LIU, Y., GOYAL, N., GHAVVININEJAD, M., MOHAMED, A., LEVY, O., STOYA11, V., & ZETTLEMOYER, L. (2019). BART : Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension [arXiv :1910.13461 [cs, stat]], (arXiv :1910.13461). <http://arxiv.org/abs/1910.13461>
- LI, C., KNOPMAN, D., XU, W., COHEN, T., & PAKHOMOV, S. (2022). GPT-D : Inducing Dementia-related Linguistic Anomalies by Deliberate Degradation of Artificial Neural Language Models [arXiv :2203.13397 [cs]], (arXiv :2203.13397). <http://arxiv.org/abs/2203.13397>
- LI, X., ZHANG, G., HUANG, H. H., WANG, Z., & ZHENG, W. (2016). Performance Analysis of GPU-Based Convolutional Neural Networks. *2016 45th International Conference on Parallel Processing (ICPP)*, 67-76. <https://doi.org/10.1109/ICPP.2016.15>
- LIU, Y., OTT, M., GOYAL, N., DU, J., JOSHI, M., CHEN, D., LEVY, O., LEWIS, M., ZETTLEMOYER, L., & STOYA11, V. (2019). RoBERTa : A Robustly Optimized BERT Pretraining Approach [arXiv :1907.11692 [cs]], (arXiv :1907.11692). <http://arxiv.org/abs/1907.11692>
- LIU, Z., HUANG, J., XU, Y., WU, J., TAO, J., & CHEN, L. (2021). Cost-effectiveness of speech and language therapy plus scalp acupuncture versus speech and language therapy alone for community-based patients with Broca's aphasia after stroke : a post hoc analysis of data from a randomised controlled trial. *BMJ Open*, 11(9), e046609. <https://doi.org/10.1136/bmjopen-2020-046609>
- LOPES, M. (2019). O paciente "Tan". <https://www.astropt.org/2019/05/02/o-paciente-tan/>
- LORCH, M. (2011). Re-examining Paul Broca's initial presentation of M. Leborgne : Understanding the impetus for brain and language research. *Cortex*, 47(10), 1228-1235. <https://doi.org/10.1016/j.cortex.2011.06.022>
- LU, A., HUANG, H., HU, Y., ZBIJEWSKI, W., UNBERATH, M., SIEWERDSEN, J. H., WEISS, C. R., & SISNIEGA, A. (2023). Deformable motion compensation for intraprocedural vascular cone-beam CT with sequential projection domain targeting and vessel-enhancing autofocus. *Medical Imaging 2023 : Image-Guided Procedures, Robotic Interventions, and Modeling*, 12466, 169-174.

- LUONG, M.-T., PHAM, H., & MANNING, C. D. (2015). Effective Approaches to Attention-based Neural Machine Translation [arXiv :1508.04025 [cs]], (arXiv :1508.04025). <http://arxiv.org/abs/1508.04025>
- MACWHINNEY, B. (2007). The talkbank project. *Creating and Digitizing Language Corpora : Volume 1 : Synchronic Databases*, 163-180.
- MACWHINNEY, B., FROMM, D., FORBES, M., & HOLLAND, A. (2011). AphasiaBank : Methods for studying discourse. *Aphasiology*, 25(11), 1286-1307. <https://doi.org/10.1080/02687038.2011.589893>
- MARTIN, L., MULLER, B., ORTIZ SUÁREZ, P. J., DUPONT, Y., ROMARY, L., de la CLERGERIE, É., SEDDAH, D., & SAGOT, B. (2020). CamemBERT : a Tasty French Language Model. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 7203-7219. <https://doi.org/10.18653/v1/2020.acl-main.645>
- MARTINS, A. (2018). Lecture 9 : Machine Translation and Sequence-to-Sequence Models.
- MEISTER, C., VIEIRA, T., & COTTERELL, R. (2021). If beam search is the answer, what was the question ? [arXiv :2010.02650 [cs]], (arXiv :2010.02650). <http://arxiv.org/abs/2010.02650>
- MISRA, R., MISHRA, S. S., & GANDHI, T. K. (2022). Assistive Completion of Agrammatic Aphasic Sentences : A Transfer Learning Approach using Neurolinguistics-based Synthetic Dataset [arXiv :2211.05557 [cs, q-bio]], (arXiv :2211.05557). <http://arxiv.org/abs/2211.05557>
- MOHAMMED, N., NARAYAN, V., PATRA, D. P., & NANDA, A. (2018). Louis Victor Leborgne (“Tan”). *World Neurosurgery*, 114, 121-125. <https://doi.org/10.1016/j.wneu.2018.02.021>
- MORRISON, M. (2016). I would tell you if I could : Language loss, depression, and the challenge of treating patients with aphasia. 8(1).
- MUKHERJEE, A. (2021). A Study of the Mathematics of Deep Learning [arXiv :2104.14033 [cs, math, stat]], (arXiv :2104.14033). <http://arxiv.org/abs/2104.14033>
- National Aphasia Association*. (s. d.). <https://www.aphasia.org/>
- Noyaux gris centraux. (s. d.). <https://www.msdmanuals.com/fr/professional/multimedia/figure/noyaux-gris-centraux>
- OLAH, C. (2014). Understanding Convolutions. <https://colah.github.io/posts/2014-07-Understanding-Convolutions/>
- OPPENHEIM, A. V., & SCHAFER, R. W. (2013). *Discrete-time Signal Processing*. Pearson.
- PALLAVI, J., PERUMAL, R. C., & KRUPA, M. (2018). Quality of Communication Life in Individuals with Broca’s Aphasia and Normal Individuals : A Comparative Study. *Annals of Indian Academy of Neurology*, 21(4), 285-289. https://doi.org/10.4103/aian.AIAN_489_17
- PAPINENI, K., ROUKOS, S., WARD, T., & ZHU, W.-J. (2002). Bleu : a Method for Automatic Evaluation of Machine Translation. *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, 311-318. <https://doi.org/10.3115/1073083.1073135>
- Parameter Importance | Weights & Biases Documentation. (s. d.). <https://docs.wandb.ai/guides/app/features/panels/parameter-importance>
- PASCANU, R., MIKOLOV, T., & BENGIO, Y. (s. d.). On the difficulty of training recurrent neural networks.
- PASZKE, A., GROSS, S., MASSA, F., LERER, A., BRADBURY, J., CHANAN, G., KILLEEN, T., LIN, Z., GIMELSHEIN, N., ANTIGA, L., DESMAISON, A., KOPF, A., YANG, E.,

- DEVITO, Z., RAISON, M., TE1I, A., CHILAMKURTHY, S., STEINER, B., FANG, L., ... GARNETT, R. (2019). PyTorch : An Imperative Style, High-Performance Deep Learning Library. <http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>
- QIN, Y., LEE, T., KONG, A. P. H., & LIN, F. (2022). Aphasia Detection for Cantonese-Speaking and Mandarin-Speaking Patients Using Pre-Trained Language Models. *2022 13th International Symposium on Chinese Spoken Language Processing (ISCSLP)*, 359-363. <https://doi.org/10.1109/ISCSLP57327.2022.10037929>
- RADFORD, A., KIM, J. W., XU, T., BROCKMAN, G., MCLEAVEY, C., & SUTSKEVER, I. (2022). Robust Speech Recognition via Large-Scale Weak Supervision [arXiv :2212.04356 [cs, eess]], (arXiv :2212.04356). <http://arxiv.org/abs/2212.04356>
- RADFORD, A., NARASIMHAN, K., SALIMANS, T., & SUTSKEVER, I. (2018). Improving Language Understanding by Generative Pre-Training.
- RADFORD, A., WU, J., CHILD, R., LUAN, D., AMODEI, D., & SUTSKEVER, I. (2019). Language Models are Unsupervised Multitask Learners.
- RAI, A., & BORAH, S. (2021). Study of Various Methods for Tokenization. In J. K. MANDAL, S. MUKHOPADHYAY & A. ROY (Éd.), *Applications of Internet of Things* (p. 193-200). Springer. https://doi.org/10.1007/978-981-15-6198-6_18
- ROSS, K., & WERTZ, R. (2010). Quality of life with and without aphasia. *Aphasiology*. <https://doi.org/10.1080/02687030244000716>
- SCHNEIDER, S., BAEVSKI, A., COLLOBERT, R., & AULI, M. (2019). wav2vec : Unsupervised Pre-training for Speech Recognition [arXiv :1904.05862 [cs]], (arXiv :1904.05862). <http://arxiv.org/abs/1904.05862>
- SCIENCES, N. A. o., MEDICINE, I. o., & ACKERMAN, S. (1992). *Discovering the Brain* [Google-Books-ID : 3ypUq9nuncQC]. National Academies Press.
- SEBASTIAN, R., & MIRJALILI, V. (2017). *Python Machine Learning : Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow* (2^e éd., T. 1). Packt Publishing.
- SENNRICH, R., HADDOW, B., & BIRCH, A. (2016). Neural Machine Translation of Rare Words with Subword Units [arXiv :1508.07909 [cs]], (arXiv :1508.07909). <http://arxiv.org/abs/1508.07909>
- SHIM, K., & SUNG, W. (2022). A Comparison of Transformer, Convolutional, and Recurrent Neural Networks on Phoneme Recognition [arXiv :2210.00367 [cs, eess]], (arXiv :2210.00367). <http://arxiv.org/abs/2210.00367>
- SMAÏLI, K., LANGLOIS, D., & PRIBIL, P. (2022). Language rehabilitation of people with BROCA aphasia using deep neural machine translation. *Fifth International Conference Computational Linguistics in Bulgaria*, 162.
- SREEDHARAN, S. (2018). *REAL-TIME FMRI BASED NEUROFEEDBACK FOR REHABILITATION OF POST-STROKE PATIENTS WITH APHASIA* (thèse de doct.). <https://doi.org/10.13140/RG.2.2.10868.37760/1>
- Stack Overflow Developer Survey. (2022). https://survey.stackoverflow.co/2022/?utm_source=social-share&utm_medium=social&utm_campaign=dev-survey-2022
- STAHLBERG, F. (2020). Neural Machine Translation : A Review. *Journal of Artificial Intelligence Research*, 69, 343-418. <https://doi.org/10.1613/jair.1.12007>
- SUS, (2021). Wav2Vec 2.0 : A Framework for Self-Supervised Learning of Speech Representations. <https://towardsdatascience.com/wav2vec-2-0-a-framework-for-self-supervised-learning-of-speech-representations-7d3728688cae>

- SZABO, V., PLESIAK, M., YANG, Y., & HEUMANN, C. (s. d.). *Modern Approaches in Natural Language Processing*. https://slds-lmu.github.io/seminar_nlp_ss20/index.html
- van RIJSBERGEN, C. J. (2002, janvier 3). *Information Retrieval*. Butterworth.
- VASILEV, I., SLATER, D., SPACAGNA, G., ROELANTS, P., & ZOCCA, V. (2019). *Python Deep Learning : Exploring deep learning techniques and neural network architectures with PyTorch, Keras, and TensorFlow, 2nd Edition* [Google-Books-ID : ESKEDwAAQBAJ]. Packt Publishing Ltd.
- VASWANI, A., SHAZEER, N., PARMAR, N., USZKOREIT, J., JONES, L., GOMEZ, A. N., KAISER, & POLOSUKHIN, I. (2017). Attention is All you Need. *Advances in Neural Information Processing Systems, 30*. <https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fdbd053c1c4a845aa-Abstract.html>
- VOLNY, P., 11AK, D., & ZEZULA, P. (2012). Employing Subsequence Matching in Audio Data Processing.
- YANG, S., WANG, Y., & CHU, X. (2020). A survey of deep learning techniques for neural machine translation. *arXiv preprint arXiv:2002.07526*.
- ZHANG, R., HAN, J., ZHOU, A., HU, X., YAN, S., LU, P., LI, H., GAO, P., & QIAO, Y. (2023). LLaMA-Adapter : Efficient Fine-tuning of Language Models with Zero-init Attention [arXiv:2303.16199 [cs]], (arXiv:2303.16199). <http://arxiv.org/abs/2303.16199>
- ZHU, J., XIA, Y., WU, L., HE, D., QIN, T., ZHOU, W., LI, H., & LIU, T.-Y. (2020). Incorporating BERT into Neural Machine Translation [arXiv:2002.06823 [cs]], (arXiv:2002.06823). <http://arxiv.org/abs/2002.06823>

Annexe A

Dépendances et bibliothèques

```
lightning==2.0.2
torch==2.0.0
pytorch_memlab==0.2.4
PyYAML==6.0
tokenizers==0.13.3
torchdata==0.6.0
torchmetrics==0.11.4
torchtext==0.15.1
torchview==0.2.6
tqdm==4.64.1
beautifulsoup4==4.11.1
openai==0.27.2
pandas==1.5.3
PyHyphen==4.0.3
python-dotenv==1.0.0
Requests==2.30.0
scikit_learn==1.2.0
tokenizers==0.13.3
tqdm==4.64.1
evaluate==0.4.0
```