République Algérienne Démocratique et Populaire
الجمهورية الجزائرية الديـموقراطية الشعبية
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
وزارة التعليم العالي والبحث العلمي

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

2022 / 2023

Table des matières

Table des figures

Algorithmes et extraits de code

Sigles et abréviations

ASR	reconnaissance automatique de la parole
BLEU	bilingual evaluation understudy
BPE	byte pair encoding
DL	apprentissage profond
ML	apprentissage automatique
MT	traduction automatique
NLP	traitement automatique du langage
NMT	traduction automatique neuronale
S2S	séquence-à-séquence

Chapitre 1

Conception

Dans les chapitres précédents, nous avons effectué une étude bibliographique sur l'aphasie de Broca et les méthodes de traitement automatique du langage (NLP, de l'anglais : natural language processing) qui peuvent être utilisées pour la traiter (particulièrement les modèles séquence-à-séquences (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.

1.1 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 1.1. Ce système est composé de deux parties principales : (a) le sous-système d'reconnaissance automatique de la parole (ASR, de l'anglais : automatic speech recognition) qui permet de transcrire la parole aphasique en texte (partie gauche) et (b) le sous-système de traduction automatique neuronale (NMT, de l'anglais : neural machine translation) 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 transcrites 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 1.1). 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.

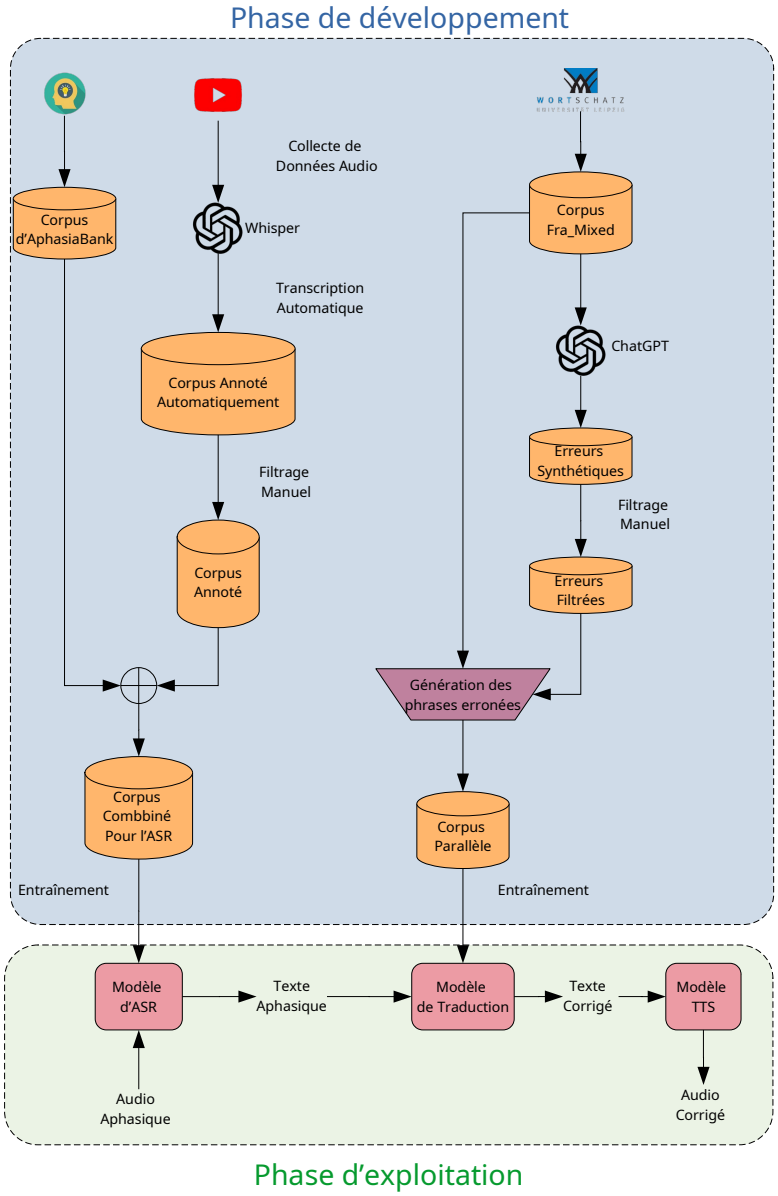

FIGURE 1.1 – Architecture générale de la solution.

1.2 Reconnaissance automatique de la parole

Le modèle d'ASR permet de transformer la parole aphasique 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 1.2. Dans cette section, nous allons détailler ces étapes. Nous notons que la dernière condition de la Figure 1.1 n'a pas été satisfaite. En effet, nous n'avons pas pu entraîner un modèle ASR à cause du manque de données.

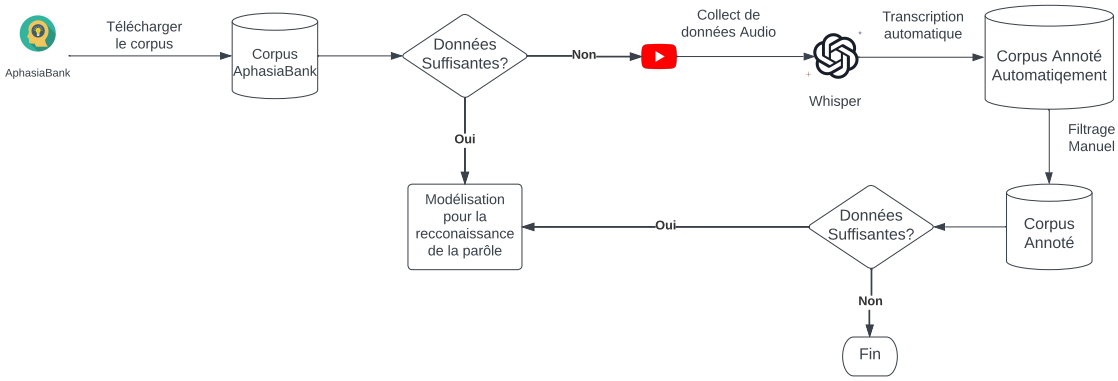

FIGURE 1.2 – Déroulement de la partie ASR.

1.2.1 Préparation des données

La première étape de tout projet de apprentissage automatique (ML, de l'anglais : machine learning) 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.

Choix du corpus et collecte de données existantes

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 de très loin insuffisant 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 1.1. On y observe

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

TABLE 1.1 – 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 transcrites, 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.

1.2.2 Création et entraînement du modèle

Les données collectées n'étant pas suffisantes, il n'est pas possible d'entraîner un modèle de apprentissage profond (DL, de l'anglais : deep learning). 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.

1.3 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 compréhensible. La procédure de sa création est décrite dans la Figure 1.3. Dans cette section, nous allons détailler les étapes de sa construction.

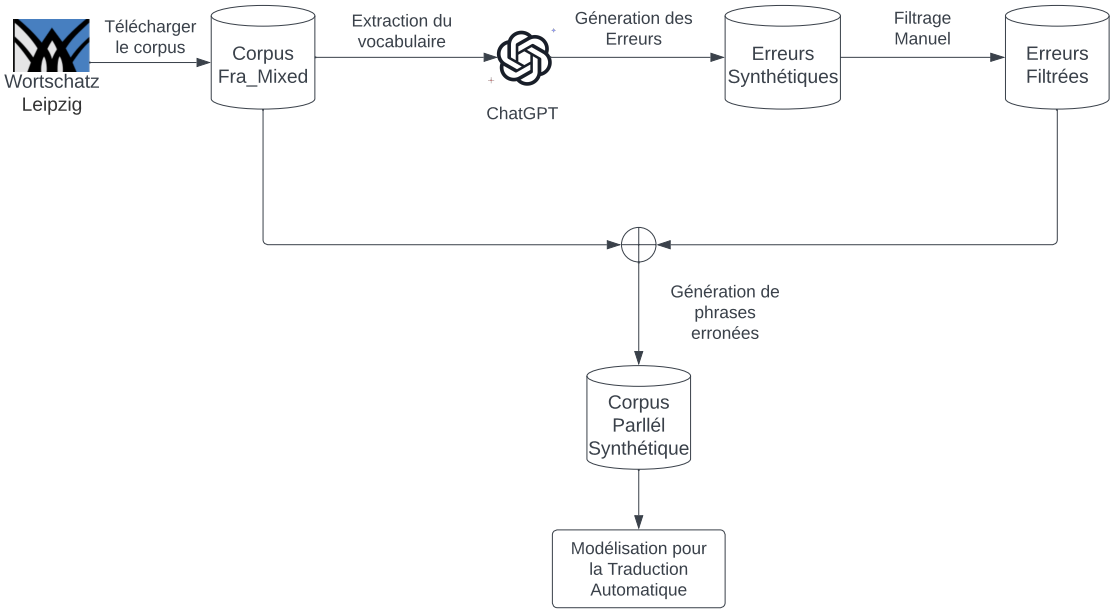

FIGURE 1.3 – Déroulement de la partie NMT.

1.3.1 Création d'un corpus parallèle

L'entraînement d'un modèle de traduction nécessite la présence d'un *corpus parallèle*, c'et-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 traiter par des experts pour corriger les erreurs dedans. Or, nous avons déjà établie 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 1.4) :

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."

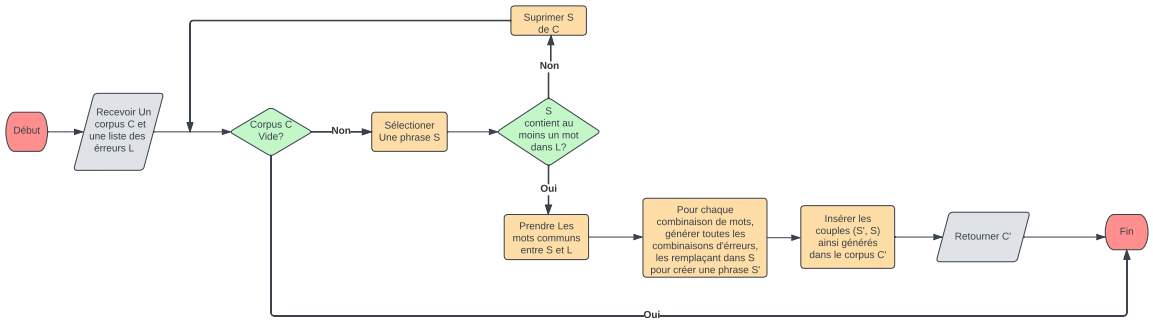

FIGURE 1.4 – Organigramme de la création du corpus parallèle.

avec

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

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.

1.3.2 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 **??**, nous avons introduit le tokeniseur byte pair encoding (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 :

$$\text{score}(x, y) = \frac{\mathbb{P}(x, y)}{\mathbb{P}(x)\,\mathbb{P}(y)} \tag{1.1}$$

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 **??**, 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 $\Sigma \to \mathbb{R}^d$ 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). 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 1.5).

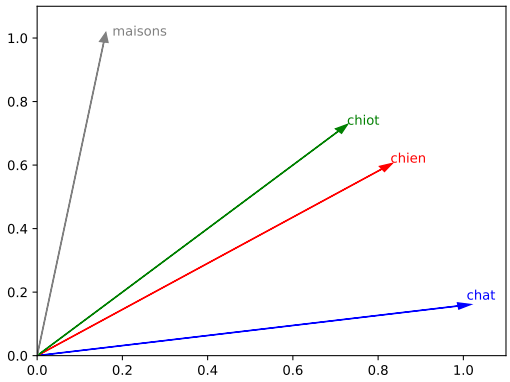

FIGURE 1.5 – Les plongements lexicaux de quelques mots avec $d = 2$.

Plusieurs algorithmes existent pour calculer les plongements lexicaux (voir Section **??**). 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 :

$$W = \overbrace{\left.\begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1N} \\ w_{21} & w_{22} & \cdots & w_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ w_{d1} & w_{d2} & \cdots & w_{dN} \end{bmatrix}\right\}}^{d=64} |\Sigma| = 5000 \tag{1.2}$$

qui a donc $|\Sigma| \times d = 5000 \times 64 = 320000$ 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 **??**, 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'activation [1]. 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).

1.3.3 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 qui est fixé à 3×10^{-4}.

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 tailles de lot allant de 16 à 256.

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

1. $\text{ReLU} : \mathbb{R} \to \mathbb{R}, x \mapsto \max(0, x)$.

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.

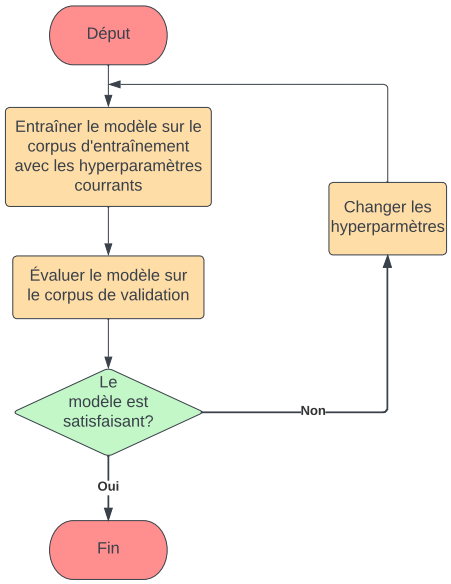

FIGURE 1.6 – Organigramme de la phase d'entraînement.

Le problème de réglage des hyperparamètres est aussi un problème d'optimisation. Or, 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 1.6. 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.

1.3.4 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 (où une liste de nombres) qui représente la qualité du modèle. La traduction automatique (MT, de l'anglais : machine translation)

étant un exemple de problème de classification en grande dimension [2] (YANG et al., 2020), les métriques utilisées sont celles de la classification.

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 $(\Omega, \mathcal{A})$, leur entropie croisée $H(p, q)$ est définie par :

$$H(p, q) = H(p) + D_{KL}(p \parallel q) \tag{1.3}$$

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 1.3 devient :

$$H(p, q) = \mathop{\mathbb{E}}_{X \sim p}[-\log q(x)] = -\sum_{x \in \Omega} p(x) \log q(x) \tag{1.4}$$

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* [3], 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.

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 1.7 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.

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

3. Pour un espace probabilisable $(\Omega, \mathcal{A})$ et $a \in \Omega$, la loi de Dirac $\delta_a : \mathcal{A} \to \{0, 1\}$ est définie par : $\delta_a(X) = \begin{cases} 1 & \text{si } a \in X \\ 0 & \text{sinon} \end{cases}$

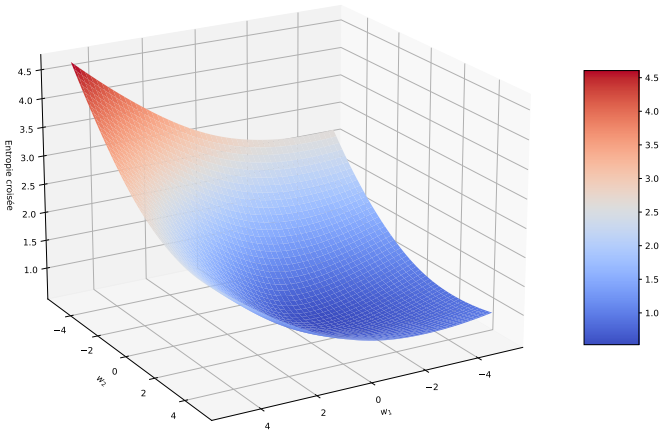

FIGURE 1.7 – Entropie croisée d'un classifieur linéaire binaire.

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.

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 :

$$\text{exactitude} = \frac{\operatorname{tr} M}{\operatorname{sum}(M)} = \frac{\sum_{i=1}^{n} M_{ii}}{\sum_{i=1}^{n}\sum_{j=1}^{n} M_{ij}} \tag{1.5}$$

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).

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 :

$$\text{rappel}_i = \frac{M_{ii}}{\sum_{j=1}^{n} M_{ij}} \tag{1.6}$$

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 a tendance 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 :

$$\text{précision}_j = \frac{M_{jj}}{\sum_{i=1}^{n} M_{ij}} \tag{1.7}$$

elle nous permet de savoir, sachant la prédiction du modèle, la probabilité qu'elle soit correcte. Une grande précision suggère que le modèle a tendance 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.

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 $\alpha = \frac{1}{1+\beta^2}$:

$$F_\beta = \frac{1}{\frac{\alpha}{P} + \frac{1-\alpha}{R}} \tag{1.8}$$

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 bilingual evaluation understudy (BLEU) (voir Section **??**) 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 **??**).

1.4 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 pris 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.

Bibliographie

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning* [Google-Books-ID : qWPwnQEACAAJ]. Springer.

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

Goldhahn, D., Eckart, T., & Quasthoff, U. (2012). Building Large Monolingual Dictionaries at the Leipzig Corpora Collection : From 100 to 200 Languages.

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

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

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

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

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.

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/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html

Yang, S., Wang, Y., & Chu, X. (2020). A survey of deep learning techniques for neural machine translation. *arXiv preprint arXiv :2002.07526*.

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
```