République Algérienne Démocratique et Populaire
الجمهورية الجزائرية الديموقراطية 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 Master**

**Pour l'obtention du diplôme de Master 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

**Page de garde** **i**

**Table des matières** **i**

**Table des figures** **ii**

**Liste des algorithmes** **iii**

**Sigles et abréviations** **iv**

**1 Conception** **1**

1.1 Architecture générale de la solution . . . . . 1

1.2 Reconnaissance automatique de la parole . . . . . 4

1.2.1 Préparation des données . . . . . 4

1.2.2 Création et entraînement du modèle . . . . . 5

1.3 Traduction automatique neuronale . . . . . 5

1.3.1 Création d'un corpus parallèle . . . . . 6

1.3.2 Création du modèle . . . . . 7

1.3.3 Entraînement . . . . . 9

**Bibliographie** **12**

# Table des figures

1.1 Architecture générale de la solution. . . . . 2

1.2 Système de réhabilitation de la parole aphasique. . . . . 3

1.3 Organigramme de la création du corpus parallèle. . . . . 7

1.4 Organigramme de la phase d'entraînement. . . . . 11

# Liste des algorithmes

# Sigles et abréviations

| | |
|---|---|
| ASR | reconnaissance automatique de la parole |
| BPE | byte pair encoding |
| DL | apprentissage profond |
| ML | apprentissage 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.

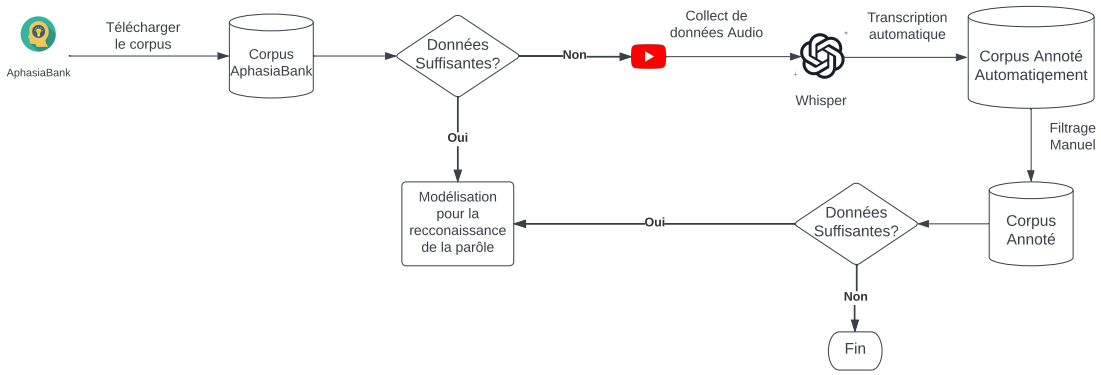
Dans ce chapitre, nous allons présenter les détails de la conception de notre système. Nous commençons par décrire la démarche suivie pour le concevoir. Puis, nous présentons l'architecture générale du système, que nous détaillons par la suite.

## 1.1 Architecture générale de la solution

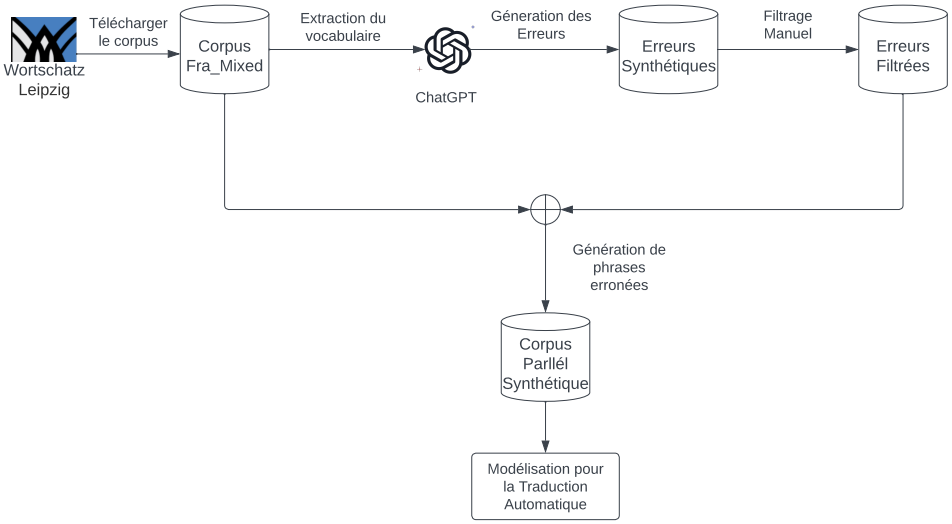
Pour résoudre le problème de la réhabilitation de la parole aphasique, nous proposons un système dont l'architecture générale est illustrée dans la Figure 1.1. Ce système est composé de deux parties principales : (a) le sous-système d'reconnaissance automatique de la parole (ASR, de l'anglais : automatic speech recognition) qui permet de transcrire la parole aphasique en texte et dont l'architecture est illustrée dans la Figure 1.1a 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 et dont l'architecture est illustrée dans la Figure 1.1b.

Pour la partie ASR, nous avons choisi de l'organiser en deux activités principales :

(i) Préparation d'un corpus de donnée (parole/écrit), ce qui passe par :
- La collecte de données existantes : consultation de corpus existants et de bases de données de parole aphasique, téléchargement et évaluation de ces données. Si les données collectées sont suffisantes (en qualité et en volume), passer à la modélisation, sinon, passer à l'étape suivante.
- La collecte de données supplémentaires : repérage et téléchargement de vidéos de parole aphasique sur internet.
- Transcription des données collectées : dans un premier temps, les vidéos sont transcrites automatiquement à l'aide d'un modèle ASR existant.




(a) Déroulement de la partie ASR.




(b) Déroulement de la partie NMT.

FIGURE 1.1 – Architecture générale de la solution.

— Filtrage manuel : les transcriptions automatiques sont filtrées manuellement pour corriger les erreurs de transcription. Le résultat de cette étape est un corpus de vidéos transcrites semi-automatiquement.

— Évaluation du corpus : si le corpus est suffisant (en qualité et en volume), passer à la modélisation, sinon, passer à la partie NMT.

(ii) Création et entraînement d'un modèle sur ce corpus.

Une organisation similaire est adoptée pour la partie NMT :

(i) Création d'un corpus parallèle (parole aphasique/parole saine) : contrairement à la partie ASR, aucun corpus parallèle n'est disponible pour l'aphasie de Broca. Dans l'absence de possibilité d'en créer un, nous avons choisi de créer un corpus parallèle synthétique. Cela peut être réalisé en suivant la procédure suivante :

— Prendre un corpus du Français.

— Extraire le vocabulaire de ce corpus.

— Sélection des mots : choisir dans ce vocabulaire les mots qui sont *"fréquents"* *"difficiles"* à prononcer.

— Création des erreurs : passer ces mots au modèle chatGPT pour générer des variantes erronées.

— Filtrage manuel : traiter manuellement les erreurs générées pour ne garder que celles qui sont similaires aux erreurs produites dans l'aphasie de Broca.

— Création du corpus parallèle : à partir du corpus de l'original et des erreurs filtrées, créer un corpus parallèle.

(ii) Création et entraînement d'un modèle sur ce corpus : une fois le corpus parallèle créé, il est possible de créer un modèle de traduction et de l'entraîner sur ce corpus. La démarche traditionnelle de apprentissage profond (DL, de l'anglais : deep learning) est suivie pour cette étape :

— Division du corpus : le corpus est divisé en trois parties : 1. corpus d'entraînement, utilisé pour entraîner le modèle, 2. corpus de validation, utilisé pour évaluer le modèle pendant l'entraînement et 3. corpus de test, utilisé pour évaluer le modèle après l'entraînement.

— Création du modèle : dans cette étape, l'architecture du modèle est choisie en fonction de la tâche en question.

— Entraînement du modèle : le modèle est entraîné sur le corpus d'entraînement pour ajuster ses paramètres.

— Évaluation du modèle : le corpus de test est utilisé pour mesurer la performance du modèle. Plusieurs métriques peuvent être utilisées pour cette évaluation. Si les résultats sont satisfaisants, passer à l'étape suivante, sinon, passer à l'étape précédente.

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.

Une fois les deux modèles entraînés, ils peuvent être enchainés pour former un système "speech-to-speech" `SA.mp3` représente le son aphasique, la composante ASR du système

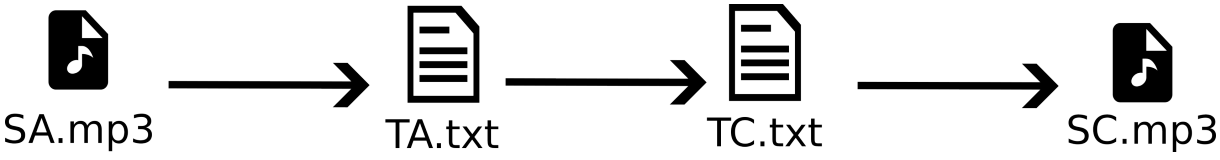



FIGURE 1.2 – Système de réhabilitation de la parole aphasique.

permet de le transcrire en texte, produisant un texte aphasique `TA.txt`. Ce texte est ensuite traduit en texte correct `TC.txt` par la composante NMT du système. Finalement, le texte correct est transformé en son correct `SC.mp3` par un outil de synthèse vocale.

## 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 établies dans la section précédente. Dans cette section, nous allons détailler ces étapes.

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

| | 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.

**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 donc 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 transcrits, 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 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.

## 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. Dans cette section, nous allons détailler les étapes de sa construction.

### 1.3.1 Création d'un corpus parallèle

L'entraînement d'un modèle de traduction nécessite la présence d'un *corpus parallèle* c.-à-d. 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 en suite 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 différent 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 1.3) :

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.

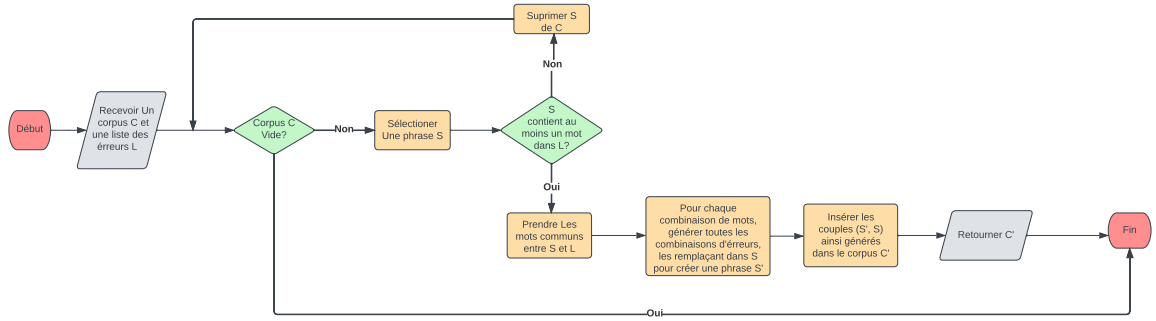



FIGURE 1.3 – Organigramme de la création du corpus parallèle.

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 utilisées 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 et 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 $\Sigma$ 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.

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 :

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

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.

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

### Algorithmes d'optimisation

Plusieurs algorithmes d'optimisation sont utilisé 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 $\eta$ qui est fixé à $3 \times 10^{-4}$.

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.

### 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, 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 1.4. 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 noté 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 et donc la seule manière d'obtenir une estimation non biaisée de la performance du modèle.

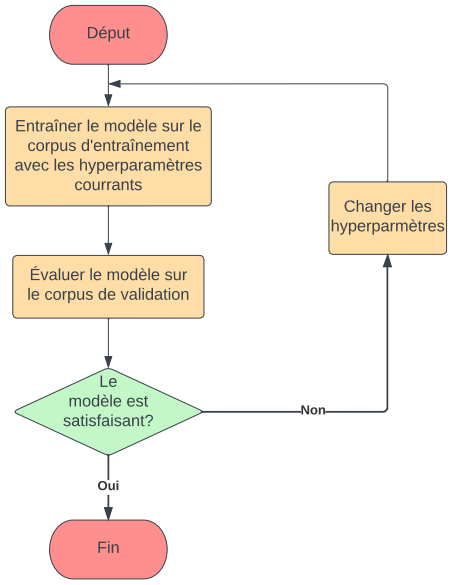



FIGURE 1.4 – Organigramme de la phase d'entraînement.

# Bibliographie


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.

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