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

BLEU	bilingual evaluation understudy
GPT	generative pre-trained transformer

Chapitre 1

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.

1.1 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.

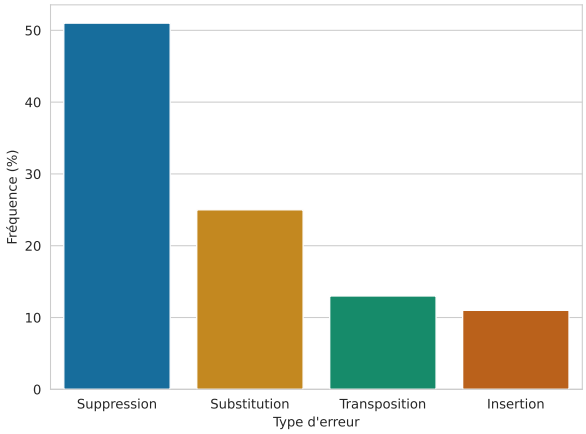


FIGURE 1.1 – Fréquences des catégories d'erreurs

Les fréquences de ces erreurs (pour les 32 premiers mots qui ont 327 modifications) sont présentées dans la Figure 1.1. 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 1.1).

```python
def corrupt_word(
    word,
    p_remove=0.51,
    p_substitute=0.25,
    p_transpose=0.13,
    p_insert=0.11,
    p_skip=0.5,
    all_syllables=None,
):
    from random import seed, randint, random
    from hyphen import Hyphenator

    hyphenator = Hyphenator("fr_FR")
    syls = hyphenator.syllables(word)

    # skip words that are too short
    if len(syls) < 3:
        return word

    # skip all words with probability p_skip
    if random() < p_skip:
        return word

    # remove a syllable with probability p_remove
    if random() < p_remove:
        idx = randint(0, len(syls) - 1)
        del syls[idx]

    # substitute a syllable with probability p_substitute
    if random() < p_substitute:
        idx1 = randint(0, len(syls) - 1)
        syls[idx] = choice(all_syllables)

    # transpose two syllables with probability p_transpose
    if random() < p_transpose:
        idx1 = randint(0, len(syls) - 1)
        idx2 = randint(0, len(syls) - 1)
        syls[idx1], syls[idx2] = syls[idx2], syls[idx1]

    # insert a syllable with probability p_insert
    if random() < p_insert:
        idx = randint(0, len(syls) - 1)
        syls.insert(idx, choice(all_syllables))
    return "".join(syls)
```

Extrait de code 1.1 – 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.

1.2 Corpus créé

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

1.2.1 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 1.1).

$$\text{perplexité}(\mathcal{C}, \mathcal{M}) = \prod_{s \in \mathcal{C}} \mathcal{M}(s)^{-\frac{1}{|\mathcal{C}|}} \tag{1.1}$$

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 **??**). 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 1.1 – 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

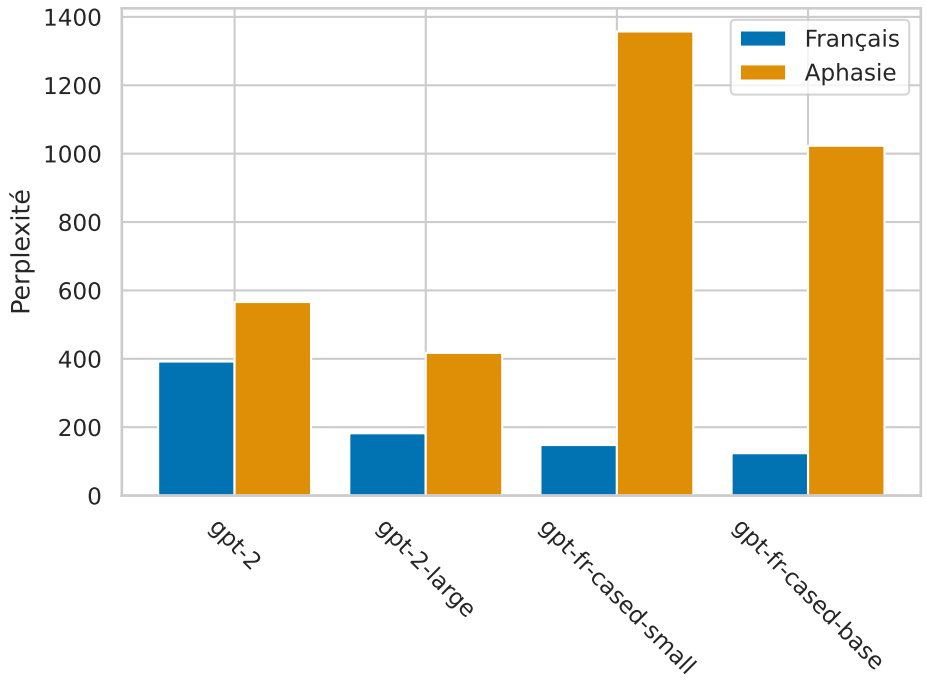


FIGURE 1.2 – Perplexité des phrases du corpus par rapport à différents modèles de langue.

Le code utilisé pour calculer la perplexité est donné par l'Extrait de code 1.2. Ce code a été exécuté sur Google Colaboratory avec une instance GPU munie de 16 Go de mémoire vive.

Les résultats sont donnés par Figure 1.2 et Table 1.1. 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.

```python
import pandas as pd
import evaluate

# Load perplexity metric
perplexity = evaluate.load("perplexity", module_type="measurement")

# Load data
df = pd.read_csv("val.csv")
fr = df.french.tolist()
aph = df.aphasia.tolist()

# Compute perplexities
fr = perplexity.compute(
  model_id='asi/gpt-fr-cased-base',
  add_start_token=False,
  data=fr
)
aph = perplexity.compute(
  model_id='asi/gpt-fr-cased-base',
  add_start_token=False,
  data=aph
```

```
)

print(f'{fr["mean_perplexity"]=:.2f}, {aph["mean_perplexity"]=:.2f}')
```

Extrait de code 1.2 – Calcul de la perplexité avec le modèle `gpt-fr-cased-base`.

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.

1.2.2 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 **??**). 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.

L'Extrait de code 1.3 donne le code utilisé pour calculer le score BLEU. 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). L'exécution de ce code donne un score BLEU de 62.72% sur le corpus de validation.

```
import pandas as pd
from torchmetrics.functional import bleu_score

df = pd.read_csv("val.csv")

aphasia = df.aphasia.to_list()
french = df.french.tolist()

french = [[s] for s in french]
score = bleu_score(aphasia, french) * 100
print(f"{score=:.2f}%")
```

Extrait de code 1.3 – Calcul du score `BLEU` sur le corpus de validation.

1.3 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`.

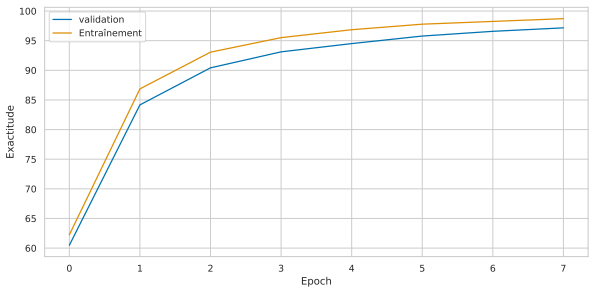
1.3.1 Choix des hyperparamètres

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 :

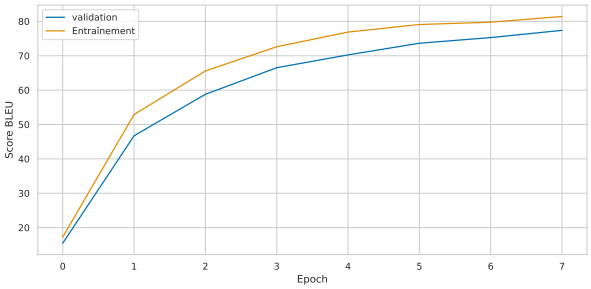
— Dimension de plongement : 64
— Dimension de la couche cachée : 64
— Taux d'apprentissage : 3×10^{-4}
— Dropout : 0.1
— Nombre de couches de l'encodeur/décodeur : 3
— Nombre de têtes d'attention : 4
— Norme maximale des plongements : 1
— Taille de lot : 256
— Norme maximale du gradient : 1
— Coefficient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

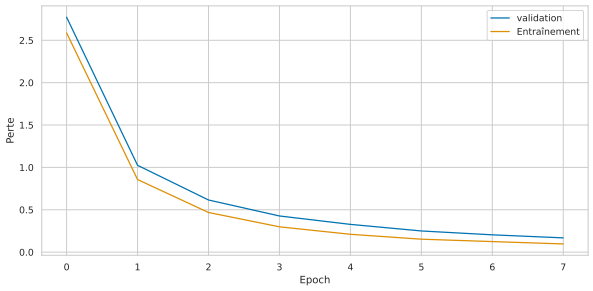
1.3.2 Résultats

(a) Exactitude



(b) BLEU





(c) Perte

FIGURE 1.3 – Évolution des métriques au cours de l'entraînement.

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 sont présentés dans la Figure 1.3. 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%. Cela suggère que le modèle ne sur-apprenne pas et qu'il est capable de généraliser. 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.

1.4 Réglage des hyper-paramètres

Les résultats présentés dans Section 1.3.2 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.

1.4.1 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 1.3.1.

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.

1.4.2 Résultats

Les 20 essais ont été effectués en 2 h 17 min 13 s 11 ms. Les résultats sont présentés dans Figure 1.4. La figure en question est une visualisation en coordonnées parallèles des

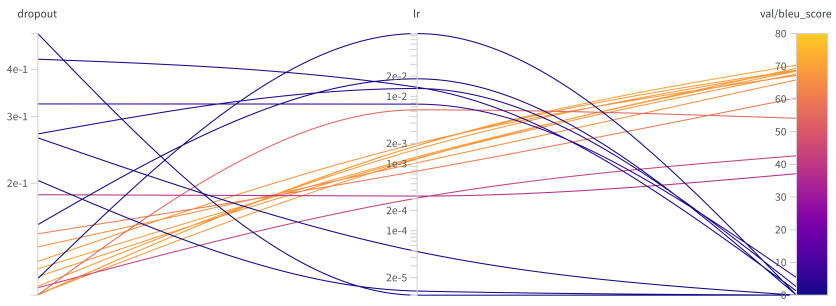


FIGURE 1.4 – Résultats de la recherche bayésienne des hyperparamètres.

résultats. L'axe 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 1.2 sous forme numérique.

`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 1.2 – Résultats de la recherche bayésienne des hyperparamètres.

On observe sur la figure un regroupement des lignes oranges (les meilleurs essais) dans la région qui correspond à `dropout` $\in [0.1, 0.15]$ et $\eta \approx 10^{-3}$. Cela est aussi apparent sur le tableau. Une analyse factorielle sur les essais effectués nous permet d'estimer l'importance

	Importance	Corrélation
`dropout`	63.9%	−0.899
`lr`	36.1%	−0.647

TABLE 1.3 – Importance et corrélation des hyperparamètres.

des hyperparamètres 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 1.3 et Figure 1.5). La meilleure valeur du score BLEU est obtenue avec `dropout` ≈ 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.

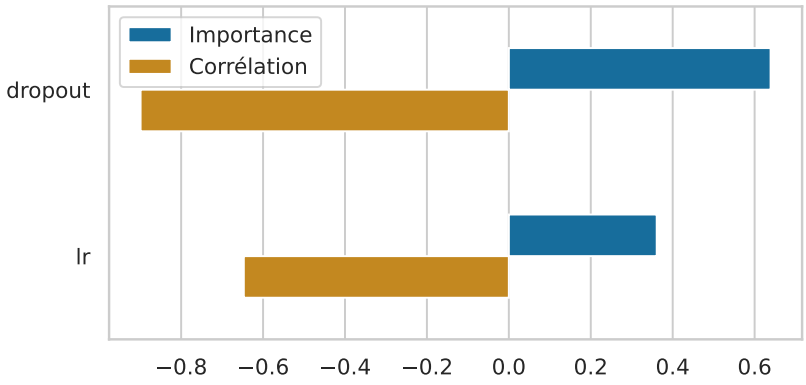


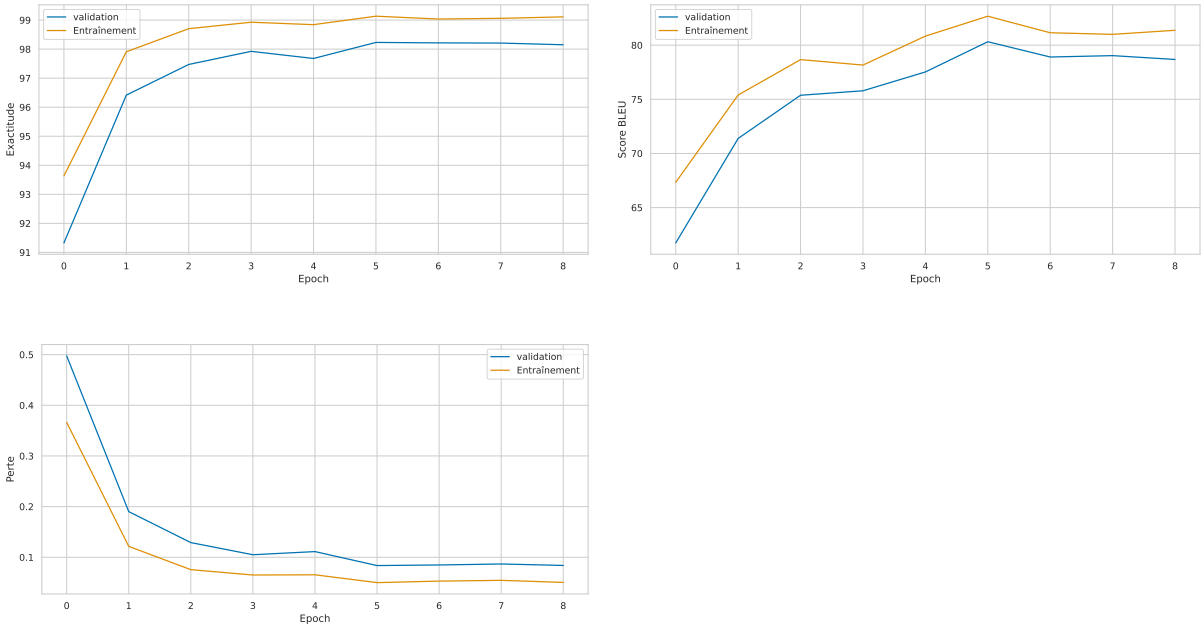
FIGURE 1.5 – Importance des hyperparamètres et corrélation avec le score BLEU.

1.5 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 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.

(a) Exactitude

(b) BLEU



(c) Perte

FIGURE 1.6 – Évolution des métriques avec les hyperparamètres optimaux.

Les courbes d'apprentissage sont présentées sur la Figure 1.6. On remarque que les métriques d'entraînement et de validation sont très fortement corrélées. Il est donc im-

probable 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 98.64%, l'exactitude de 79.61% et de 98.15% et la perte de 8.39×10^{-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 1.2, 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.

1.6 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 donné des résultats satisfaisants. 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.

Enfin, nous avons développé une interface web pour que les utilisateurs puissent utiliser notre modèle. Il s'agit d'une simple application de traduction de texte.

Bibliographie

GHUMMAN, 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.

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
```