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

Liste des algorithmes et d'extraits de code

Sigles et abréviations

API	interface de programmation d'application
ASR	reconnaissance automatique de la parole
BLEU	bilingual evaluation understudy
CNN	réseau de neurones à convolutions
DL	apprentissage profond
GPT	generative pre-trained transformer
ML	apprentissage automatique
MLOps	Machine Learning Operations
MT	traduction automatique
NMT	traduction automatique neuronale
RNN	réseau de neurones récurrent

Chapitre 1

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.

1.1 Outils et technologies

La création d'un système aussi sophistiqué que le nôtre est une tâche assez complexe. Elle comporte plusieurs étapes dont certaines nécessite 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.

1.1.1 Python

Python est un langage de programmation open-source interprété, orienté objet et multiparadigme. Introduit 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 répoertoire officiel de Python contient plus de 450 000 packages. Tous les programmes écrits dans le cadre de ce projet sont écrits en Python.

1.1.2 Jupyter

Jupyter est un environnement d'exécution interactif pour Python (et plusieurs autres langages). Il permet de créer des documents appelés "notebooks" qui combine 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.

1.1.3 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 trouver 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 dédié au apprentissage automatique (ML, de l'anglais : machine learning) 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.

1.1.4 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`.

1.1.5 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).

1.1.6 PyTorch

PyTorch est une bibliothèque open-source de apprentissage profond (DL, de l'anglais : deep learning) pour Python. Elle permet de créer et d'entraîner rapidement et facilement des réseaux de neurones. Elle est développée par Facebook AI, mais elle fait 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 (réseau de neurones à convolutions (CNN, convolutional neural network), réseau de neurones récurrent (RNN, de l'anglais : recurrent neural network), 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`.

1.1.7 Lightning

Lightning.AI est une plateforme pour créer, entraîner et déployer des modèles de DL. La bibliothèque `lightning` distribuée contient 3 packages (voir Figure 1.1) :

— `lightning.pytorch`

— `lightning.fabric`

— `lightning.applications`

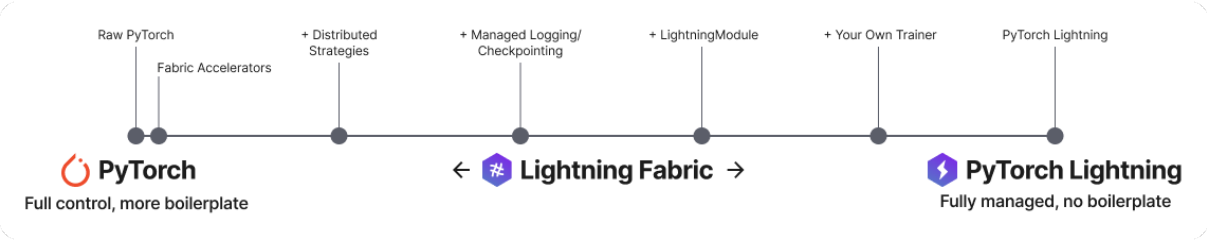

FIGURE 1.1 – Composantes de la bibliothèque `lightning` (FALCON & THE PYTORCH LIGHTNING TEAM, 2019).

`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 bilingual evaluation understudy (BLEU). Nous avons fait usage de ce package pour calculer ces métriques.

1.1.8 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`.

1.1.9 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.

1.1.10 Open AI

Open AI est une entreprise de recherche en intelligence artificielle. C'est cette entreprise qui a créé les modèles generative pre-trained transformer (GPT) et Whisper discutés dans le chapitre **??**.

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.

1.1.11 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.

1.2 Création 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 reconnaissance automatique de la parole (ASR, de l'anglais : automatic speech recognition). Ensuite nous passons à celui de la partie traduction automatique (MT, de l'anglais : machine translation).

1.2.1 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 transcrites.

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=d4Cybwx3sHk
    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.

1.2.2 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 1.1). 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`.

```
import os
import time

import openai
from yaml import Dumper, dump

openai.api_key = os.getenv("openai")

system_prompt = (
    "You are a linguisitics researcher. "
    + "you are trying to understand the speaking patterns "
    + "of people with aphasia. "
    + "You will be given a french word. "
    + "You must try to modify it in the same way "
    + "a french speaker with aphasia would."
    + "Your response should be a single word, "
    + "with no punctuation, nor line breaks."
)

num_errors_per_word = 10
errors = {}
for word in words:
    prompt = (
        f'Deforme le mot "{word}" a la facon dont le ferait '
        + "un aphasique de Broca. Repond avec un seul mot."
    )

    success = False
    ntries = 0
    while not success:
        try:
```

```
            responses = openai.ChatCompletion.create(
                model="gpt-3.5-turbo",
                messages=[
                    {"role": "system", "content": system_prompt},
                    {"role": "user", "content": prompt},
                ],
                timeout=100,
                n=num_errors_per_word,
            )
        except openai.error.RateLimitError:
            ntries += 1
            print(f"Retry {ntries}")
            time.sleep(1)
            continue
        success = True
    errors[word] = list(
        response_object["message"]["content"]
        for response_object in responses["choices"]
    )

with open("errors.yaml", "x", encoding="utf-8") as f:
    dump(errors, f, Dumper, allow_unicode=True, sort_keys=False)
```

Extrait de code 1.1 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 1.2. 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.

```
from itertools import product


def get_corrupted_sentences(corpus, errors):
    result = {}
    for sentence in corpus:
        variants = []
        corruptable_words = [
            word
            for word in errors  # The keys of the errors dictionary
            if word in sentence.split(" ")  # Only those in the sentence
        ]
        corruptions = list(
            product(  # Cartesian product of all possible corruptions
                *[
                    [(word, word)]  # Identity corruption
                    + [
                        (word, error) for error in errors[word]
                    ]  # Modification corruptions
                    for word in corruptable_words
                ]
            )
        )
        for corruption in corruptions:
```

```
            splt = sentence.split(" ")
            for word, error in corruption:
                splt[splt.index(word)] = error
            if splt != sentence.split():
                variants.append(" ".join(splt))
        result[sentence] = set(variants)
    return result
```

Extrait de code 1.2 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.

1.3 Création du modèle de traduction automatique

La section précédente fournie 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 traduction automatique neuronale (NMT, de l'anglais : neural machine translation).

1.3.1 Création du modèle

Comme discuté dans Section 1.1, 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 1.3 montre la méthode d'initialisation de cette classe.

```
def __init__(
    self,
    d_model: int,
    nhead: int,
    source_vocab_size: int,
    target_vocab_size: int,
    source_pad_idx: int,
    num_encoder_layers: int,
    num_decoder_layers: int,
    max_len: int,
    dim_feedforward: int,
    dropout: float,
    lr: float,
):
    super().__init__()

    self.input_embedding = nn.Embedding(source_vocab_size, d_model)
    self.input_position_embedding = nn.Embedding(max_len, d_model)

    self.output_embedding = nn.Embedding(target_vocab_size, d_model)
    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 1.3 Méthode d'initialsation 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 et `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 1.2 montre le graphe de calcul du modèle résultant.

	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 1.1 – Résumé du modèle.

En utilisant les valeurs proposées dans le Chapitre **??** pour les hyperparamètres :

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

nous obtenons un modèle dont la Table 1.1 résume les paramètres par couche.

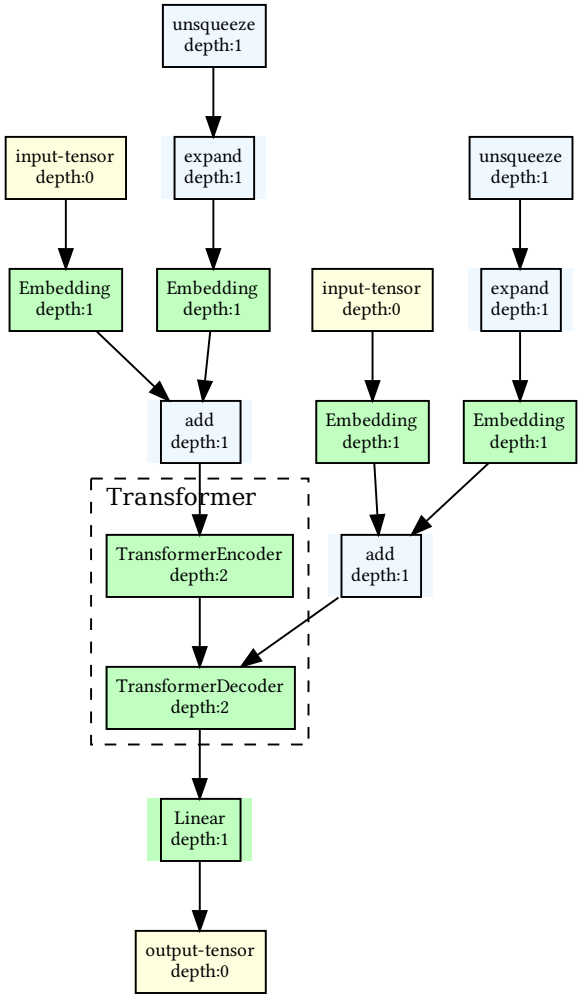

FIGURE 1.2 – Graphe de calcul du modèle définit dans la classe `Transformer` (profondeur = 2).

1.3.2 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 1.4

— `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)`.

```
trainer = Trainer(
    max_epochs=epochs,
    deterministic=True,
    logger=WandbLogger(project="unaphasiator")
    callbacks=[early_stopping, checkpoint],
    gradient_clip_val=1.0,
)
```

Extrait de code 1.4 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 implémente 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.

1.3.3 Réglage des hyperparamètres

Bibliographie

About Python. (2022). https://pythoninstitute.org/about-python

Falcon, W., & The PyTorch Lightning team. (2019). *PyTorch Lightning* (Version 1.4). https://doi.org/10.5281/zenodo.3828935

Le tutoriel Python. (2023). https://docs.python.org/3/tutorial/index.html

Stack Overflow Developer Survey. (2022). https://survey.stackoverflow.co/2022/?utm_source=social-share&utm_medium=social&utm_campaign=dev-survey-2022