Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université des Sciences et de la Technologie Houari Boumediene
Faculté d'électronique et d'informatique
Département d'informatique

Mémoire de Master

Domaine : Informatique

Spécialité : Systèmes Informatiques Intelligents

Thème

Reconnaissance Automatique de la Parole pour les Applications de Traitement Automatique du langage Naturel (TALN)

Présenté par :
- Yasmine HADJERES
- Sofiane MEDJKOUNE

Proposé et dirigé par :
- Pr. Ahmed GUESSOUM
- Mme Asmaa AOUICHAT

Devant le jury composé de :

- Pr. F. KHELLAF	Présidente
- Mme M. DJEFFAL	Membre

Projet N° : 068/2019

Sommaire

Table des figures

Liste des tableaux

LISTE DES ALGORITHMES

Déclaration de non-plagiat

Nous, soussigné(e)s, déclarons que ce mémoire a été rédigé par nous-mêmes, que l'œuvre contenue dans ce document est la nôtre, sauf comme indiqué explicitement dans le texte, et que ce travail n'a pas été soumis pour un autre diplôme ou une autre qualification professionnelle. Nous déclarons aussi avoir respecté les conventions de recherche utilisées pour citer et faire des références précises dans ce mémoire aux travaux, idées et phrases/mots d'autres personnes. Nous déclarons aussi que nous comprenons ce que plagiat veut dire et que c'est un acte de malhonnêteté intellectuelle.

Nom et Prénom: HADJERES Yasmine

Date: 27 Juin 2019

Signature HADJERES Yasmine.

Nom et Prénom: MEDJKOUNE Sofiane

Date: 27 Juin 2019

Signature MEDJKOUNE Sofiane.

REMERCIEMENTS

Nous commençons par adresser nos remerciements à Dieu le miséricordieux qui nous a donné la force, la patience, le courage et tous les moyens pour mener à terme notre projet de fin d'études.

À notre encadreur : Professeur Ahmed Guessoum

Tous nos remerciements vont vers vous, professeur Ahmed Guessoum. D'abord pour la qualité des enseignements techniques et humains que vous avez su nous transmettre ces deux dernières années. Nous vous remercions pour la bienveillance, la patience, les remarques précieuses ainsi que la multitude de conseils que vous nous avez donnés durant toute la période de ce projet de fin d'études. Pour finir, merci pour la chance que vous nous avez donnée de travailler à vos côtés.

À notre co-encadreur : Mademoiselle Asma Aouichat

Nous avons eu l'honneur de travailler à vos côtés durant tout un semestre. Votre gentillesse, votre compréhension et votre disponibilité ont toujours suscité notre admiration. Veuillez bien recevoir nos remerciements les plus distincts.

Aux membres du jury

Un grand merci au professeur F. KHELLAF que nous avons l'honneur d'avoir comme présidente du jury ainsi qu'a Mme M. DJEFFAL, membre du jury. Merci à vous de prendre le temps d'étudier notre travail et de le critiquer.

Enfin, nous tenons à remercier toutes les personnes qui nous ont aidé de près ou de loin à la réalisation de ce modeste travail.

DÉDICACES

Je dédie ce projet de fin d'études à,

Mes chers parents, qui m'ont donné la force et le courage de les rendre fiers. Que Dieu vous protège et vous donne tout le bonheur du monde.

Ma petite soeur et à mes grands parents.

Tous mes amis et plus particulièrement Nazime, Yasser, Anas, Anaïs, Anis, Lilia, Ramzi, Wissam, Massil, Medina, Adel, Ghiles, Naila.

La famille Open Minds Club, vous n'avez pas idée de ce que vous m'avez apporté.

Vous êtes les personnes à qui je tiens le plus, j'espère vous garder près de moi. Merci.

Sofiane

DÉDICACES

Je dédie ce travail à :

Ma famille en commençant par mes chers parents, mon mari, mon frère, mes grands parents et ma belle famille, que Dieu vous garde tous pour moi.

À tous mes amis en commençant par : Hadia, Chanez, Lilia, Naila, Fida, Salim, Sofiane (C), Nadjib.

Yasmine

INTRODUCTION

L'intelligence artificielle (IA) vise à permettre aux machines d'émuler l'intelligence humaine. Cette branche de l'informatique inclut l'apprentissage, le raisonnement et l'auto-correction. L'intelligence artificielle, au fil des décennies, a révolutionné différents domaines en passant par la vision artificielle, les systèmes experts et le traitement automatique du langage naturel (TALN) afin de rendre les interactions entre l'homme et la machine plus naturelles. Depuis le début des années 1960s, plusieurs applications de TALN commencèrent à voir le jour à savoir, la traduction automatique, les systèmes de questions-réponses, mais aussi, les systèmes de reconnaissance de la parole.

La reconnaissance de la parole a suscité un grand engouement que ce soit dans le domaine de la recherche ou dans le domaine multimédia. À ce jour, toutes les grandes compagnies comme Google, Apple, Facebook, Amazon ou Microsoft travaillent activement sur l'amélioration des technologies du traitement automatique de la parole afin d'offrir une meilleure interaction avec leurs produits. Cependant, une mauvaise reconnaissance peut mener à une mauvaise communication de l'information, une perte de temps et une frustration de l'utilisateur. Le monde arabe, et les arabophones plus particulièrement, ne disposent pas d'outils de reconnaissance de la parole aussi performants que ceux qui existent pour d'autres langues comme l'anglais. Ceci pose donc le défi d'améliorer ces outils pour la langue arabe.

Il est clair que de nos jours, les systèmes de reconnaissance de la parole pour les langues étrangères, et notamment l'anglais, ont atteint des niveaux de fiabilité très intéressants. Ceci n'est malheureusement pas encore le cas pour la langue arabe qui présente des résultats nettement moins satisfaisants. Cette modeste performance est due au manque de corpus assez riches, mais aussi due à la complexité morphosyntaxique de la langue arabe par rapport aux autres langues. Le constat n'est cependant pas que négatif car, aujourd'hui, plusieurs équipes de recherches et entreprises travaillent sur l'amélioration de ces systèmes. Il y a eu également l'introduction de nouvelles techniques en matière d'apprentissage profond qui permettent de tirer plus facilement profit des corpus de dialogue en arabe et, ainsi, d'outrepasser la complexité de la langue.

Notre travail s'inscrit dans le cadre d'un projet de recherche. Dans l'optique de contribuer à l'amélioration des technologies du traitement automatique de la parole, nous avons développé un système de reconnaissance de la parole pour la langue arabe. Pour ce faire, nous avons commencé par étudier les concepts théoriques de ces systèmes ainsi que les différents travaux réalisés dans ce sens en examinant la littérature et l'état de l'art. Nous avons travaillé après cela sur la conception des différents modules du système pour passer ensuite à l'implémentation des différentes approches et techniques et faire une étude comparative des résultats obtenus. Nous avons enfin considéré les systèmes de questions-réponses comme domaine d'application.

Comme contribution additionnelle à ce projet, nous avons mis en place un environnement de développement de systèmes de reconnaissance de la parole basé sur plusieurs architectures. Cet environnement permettra d'intégrer le module de reconnaissance de la parole à toute application de TALN. En plus de cela, il donnera la possibilité d'améliorer le système ou encore d'en créer un nouveau à l'aide de la panoplie d'outils que nous offrons.

Le premier chapitre de ce mémoire aura pour but de présenter les systèmes de reconnaissance automatique de la parole ainsi que les systèmes de questions-réponses. Le deuxième chapitre portera sur l'étude des approches et techniques utilisées pour le développement de ces systèmes. Cela nous permettra de mieux appréhender l'étude de l'état de l'art et de nous en inspirer. Le troisième chapitre quant à lui, sera le chapitre où nous concevons le système de reconnaissance de la parole, présentons les différents résultats obtenus ainsi qu'un cas d'application avec les systèmes de questions-réponses dans le quatrième chapitre. Nous concluons ce document avec une conclusion générale ainsi que les perspectives d'amélioration.

CHAPITRE 1

SYSTÈMES DE RECONNAISSANCE DE LA PAROLE ET APPLICATION AUX SYSTÈMES DE QUESTIONS-RÉPONSES

1.1 Introduction

Dans le but de rendre notre interaction avec les outils technologiques plus intuitive, nous nous sommes servi de la voix pour remplacer l'utilisation du clavier. Pour ce faire, il a fallu développer des systèmes de reconnaissance de la parole. Ces systèmes devaient être capables d'interpréter le langage naturel humain et le transformer en requêtes afin qu'il soit utilisé dans d'autres domaines tel que les systèmes de questions-réponses.

Dans ce chapitre, nous présentons les principes et composants de base des systèmes de reconnaissance de la parole, systèmes de synthèse vocale et des systèmes de questions-réponses.

1.2 Reconnaissance de la parole

1.2.1 Définition

La reconnaissance de la parole, ou speech recognition en anglais, est la capacité d'une machine à comprendre des mots parlés. Un microphone enregistre la voix d'une personne et le matériel convertit le signal d'ondes sonores analogiques en un signal audio numérique. Les données audio sont ensuite traitées par un système, qui retranscrit ces dernières en mots [Christensson, 2014].

D'après [S Jurafsky and Martin, 2000], tout bon système de reconnaissance de la parole doit pouvoir reconnaître un vocabulaire étendu (entre 20000 et 60000 mots distincts en moyenne), ne doit pas dépendre de l'orateur et doit traiter un discours continu qui est en accord avec le langage naturel humain.

Ces systèmes reposent sur deux approches :

- Architecture basée sur la reconnaissance de phonèmes : qui est considérée comme l'approche classique pour cette tâche de reconnaissance.
- Architecture bout à bout (End-To-End en anglais) : qui se base sur les techniques les plus avancées de l'apprentissage artificielle pour des systèmes plus performants mais également plus flexibles.

La reconnaissance de la parole est classée comme un domaine multidisciplinaire qui se base sur les statistiques, l'apprentissage automatique, la phonétique, la linguistique, le traitement de signal et une profonde compréhension des techniques d'intelligence artificielle.

1.2.2 Historique

La parole est le principal moyen de communication entre les humains. À cet effet, le domaine de la reconnaissance de la parole a attiré beaucoup d'attention au cours des six dernières décennies. Les premières expérimentations virent le jour au cours des années 1950s avec le système "Audrey" en 1952 [Davis et al., 1952] qui reconnaissait des chiffres émis par un unique orateur. En 1962 IBM présente "Shoebox" [IBM, 1962], un système permettant de reconnaître 16 mots de la langue anglaise. Malgré ces premières avancées, c'est dans les années 1970s que ce domaine décolle réellement avec le système militaire "Harpy" [T. Lowerre, 1976] qui pouvait reconnaître 1011 mots, ce qui représentait le vocabulaire d'un enfant de 3 ans. Les années 1980s furent marquées par l'utilisation de nouvelles méthodes statistiques pour la prédiction qui, en théorie, permettaient de prédire un nombre illimité de mots. Mais, si ce n'était pas le cas en pratique à cette époque là, ce le fut deux décennies plus tard [H. Juang and Rabiner, 2005]. Les systèmes de reconnaissance de la parole n'ont cessé de se perfectionner au cours des années 1990s et 2000s comme le détaille [Pinola, 2011] pour atteindre le statut d'état de l'art dans les années 2010s ce qui correspond à l'utilisation répandue de l'apprentissage automatique ouvrant ainsi la porte à de nouvelles approches que nous découvrons dans la suite de ce document.

Il est important d'ajouter que ce ne sont pas que les performances de ces systèmes qui ont évolué au fil des décennies mais également les paramètres que nous prenons en compte lors de la reconnaissance de la parole. Alors que ces systèmes ne pouvaient donner de résultats concluants que si le discours était formel et restreint, les requêtes élémentaires et bien définies, l'environnement propre et sans bruit, la prononciation académique et monolingue, nos systèmes sont aujourd'hui capables de faire de la reconnaissance sous tout type de conditions comme le mentionnent [Yu and Deng, 2014] dans la figure 1.1 ci-dessous.

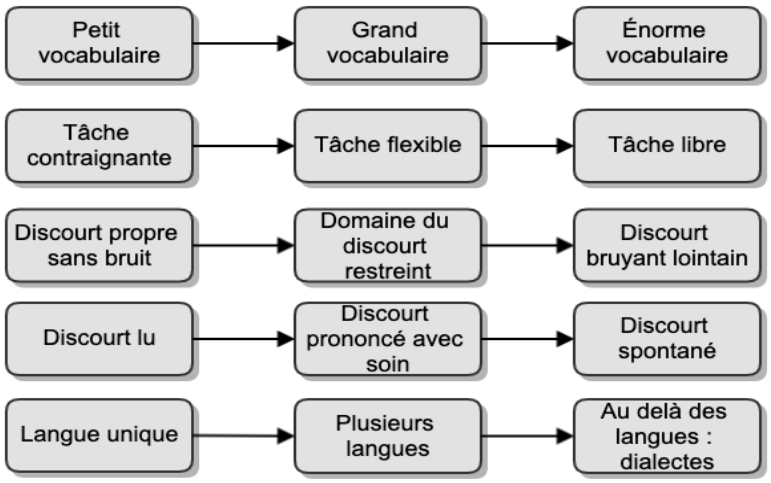

FIGURE 1.1 – Évolution des contraintes lors de la reconnaissance de la parole

1.2.3 Utilisation des systèmes de reconnaissance de la parole

Le fait que la reconnaissance de la parole ait suscité autant d'intérêt ces dernières décennies n'est pas dû au hasard. En effet, ces systèmes sont d'une grande utilité tant pour améliorer la communication entre les humains que la communication entre l'humain et la machine [Yu and Deng, 2014].

Les systèmes de reconnaissance de la parole peuvent être utilisés, entre autres applications, afin de :

- développer des assistants intelligents basés sur la reconnaissance de la parole qui, de nos jours, sont présents dans beaucoup d'appareils électroniques permettant ainsi à l'utilisateur d'exprimer des requêtes en langage naturel sans avoir à passer par un clavier réduisant ainsi considérablement le temps et les efforts fournis par ce dernier,
- faciliter la traduction lors d'une communication orale où les communiquants ne partagent pas la même langue,
- gagner en mobilité dans des environnements où une interaction écrite avec nos machines n'est pas possible tel que des travaux manuels ou tout simplement dans sa voiture,
- nouveau moyen d'interaction pour les personnes mal voyantes leur permettant ainsi d'interagir avec autrui.

1.2.4 Architecture d'un système de reconnaissance de la parole basée reconnaissance de phonèmes

Les systèmes basés reconnaissance de phonèmes partagent une architecture commune et ce, indépendamment de la langue traitée. Cette architecture comporte quatre modules principaux qui sont représentés à travers le diagramme suivant [Yu and Deng, 2014] :

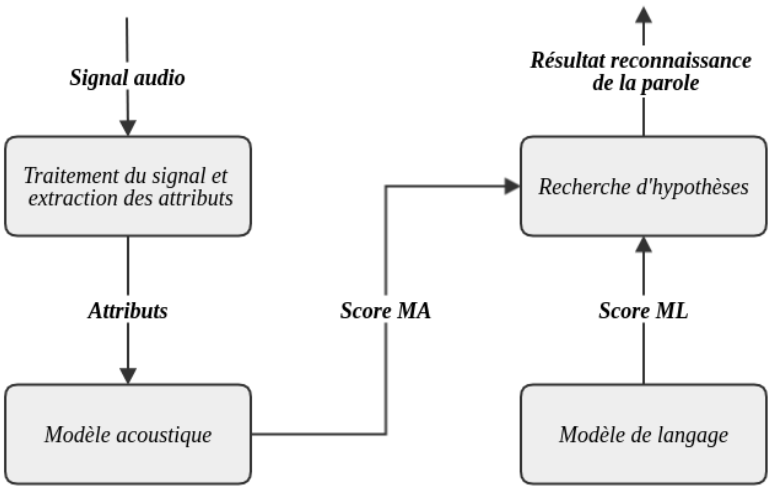

FIGURE 1.2 – Architecture d'un système de reconnaissance de la parole basée reconnaissance de phonèmes

Nous présentons dans ce qui suit chaque module de l'architecture :

1.2.4.1 Traitement du signal et extraction des attributs

Traduit de "signal processing and feature extraction", ce module de l'architecture prend un signal audio en entrée, passe par une étape de réduction du bruit suivie de la transformation du signal analogique en fréquences et donne en sortie des vecteurs contenant les caractéristiques de ces fréquences sous forme d'attributs. L'une des méthodes les plus répandues pour l'extraction de ces attributs est l'utilisation du MFCC [1] [Rashidul Hasan et al., 2004] qui prend un enregistrement audio en entrée, y applique un certain nombre de filtres et renvoie en sortie une matrice numérique de caractéristiques qui est un spectrogramme représentant la variation du signal. Nous présentons le MFCC en détail dans le troisième chapitre de ce mémoire.

1.2.4.2 Modèle acoustique

Le modèle acoustique représente la pierre angulaire de cette architecture, il a pour objectif de transformer les informations contenues dans le vecteur de caractéristiques produit par le module précédent (traitement du signal) en unités linguistiques qui seront traitées au niveau du module suivant. [Salvi, 1999] présente les approches utilisées pour répondre à ce besoin :

- l'approche appelée algorithmes dynamiques de déformation temporelle [J. Keogh and Pazzani, 2002] qui a pour but de comparer directement les séquences des données temporelles, tel que des enregistrements audio, en utilisant différentes mesures de distance tel que la distance euclidienne par exemple. Ces approches sont anciennes et très rarement utilisée de nos jours.

1. MFCC : Mel Frequency Ceptral Coefficients.

- l'approche statistique : nous parlons ici de HMM [2] [Rabiner, 1989] et de GMM [3] [Reynolds and Rose, 1995] mais aussi de modèles d'apprentissage profond ou encore des modèles hybrides HMM-DNN [4] [Deng et al., 2013]. Ces techniques seront brièvement présentées dans le chapitre 2.

1.2.4.3 Modèle de langage

Les modèles de langage sont utilisés dans les systèmes de reconnaissance de la parole pour améliorer leur performance. En pratique, ce modèle est utilisé pour rechercher une interprétation de l'entrée du modèle acoustique et prédire une suite de mots correspondant à cette entrée. D'après [Jurafsky and Martin, 2008], ce modèle peut être basé sur :

- une grammaire : Utilisée généralement lorsque la gamme de phrases à reconnaître est restreinte en terme de taille et peut être capturée par une grammaire déterministe. Ce type de modèles est très peu utilisé de nos jours ; ou
- un modèle probabiliste, le plus souvent utilisant la notion de N-Grammes [M. Katz, 1987] qui sera détaillée par la suite.

Le plus grand avantage du modèle de langage est de réduire la perplexité de la reconnaissance en imposant au système de respecter une cohérence dans la suite des mots reconnue par le modèle acoustique et ainsi, améliorer les performances de la tâche de reconnaissance [Qatab and N. Ainon, 2010].

1.2.4.4 Recherche d'hypothèses

Pour le modèle acoustique et le modèle de langage, un score est généré pour chacun afin d'évaluer le résultat et ce sont ces scores qui sont utilisés dans le module de recherche d'hypothèses. Ce module compare le score associé au modèle acoustique à celui généré par le modèle de langage afin de décider du résultat final de la reconnaissance. Si le score du modèle de langage est supérieur à celui du modèle acoustique, le système effectue une rectification dans l'ordre d'apparition des mots reconnus [Yu and Deng, 2014]. Dans le cas de la reconnaissance de la parole, le score engendré est la performance du modèle qui est représentée par des métriques que nous détaillons dans le chapitre suivant.

La figure 1.3 est un exemple de reconnaissance de la parole basée reconnaissance de phonèmes.

2. Modèles de Markov Cachés, traduit de l'anglais : "Hidden Markov Models"
3. Modèles de mélange Gaussien, traduit de l'anglais : "Gaussian Mixture Models"
4. Réseaux de Neurones Profonds, traduit de l'anglais "Deep Neural Networks"

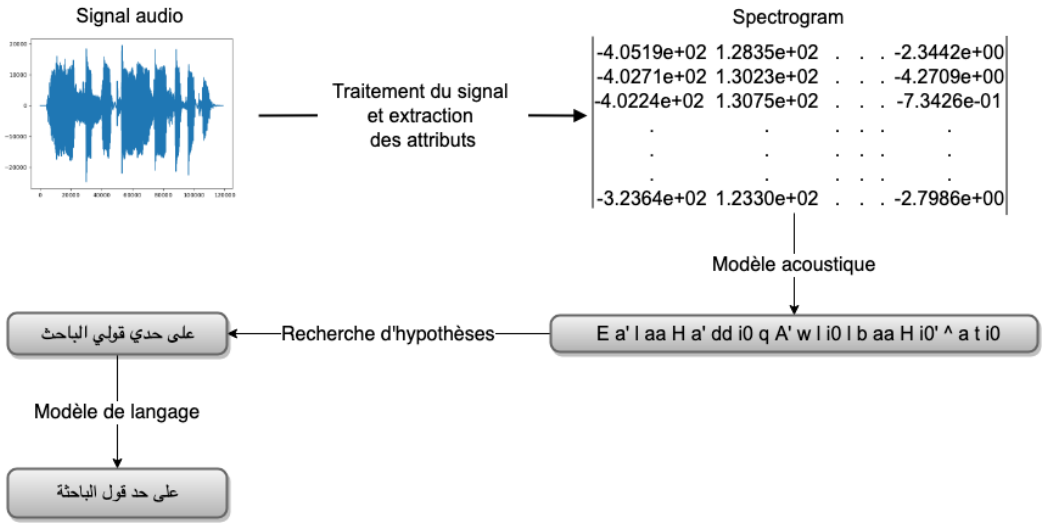

FIGURE 1.3 – Exemple de reconnaissance de la parole basée reconnaissance de phonèmes

1.2.5 Architecture End-To-End d'un système de reconnaissance de la parole

L'architecture bout à bout, ou End-To-End en anglais, porte ce nom car contrairement à l'architecture précédente elle consiste en un seul module qui prend en entrée un enregistrement audio et produit la transcription qui y correspond. Cette approche pour appréhender les systèmes de reconnaissance de parole a été introduite par [Graves and Jaitly, 2014] pour palier aux difficultés rencontrées dans l'approche basée reconnaissance de phonèmes :

- complexité de développement
- pauvreté des corpus disponibles, et
- barrière de la langue.

Nous commençons par présenter la figure suivante qui illustre les composantes principales de cette architecture :

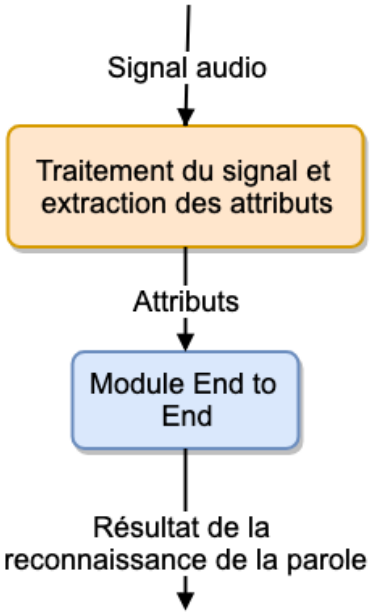

FIGURE 1.4 – Architecture End-To-End d'un système de reconnaissance de la parole

Comme nous le voyons, cette architecture se compose de deux modules principaux.

1.2.5.1 Traitement du signal et extraction des attributs

Ce module a la même fonction que le module portant le même nom dans l'approche basée reconnaissance de phonèmes qui est de générer, à partir d'une enregistrement audio, un spectrogramme sous forme de matrice numérique représentant les variations du signal de l'enregistrement.

1.2.5.2 Module End-To-End

Ce module, que nous considérons pour l'instant comme une boîte noire, est le module qui apprend à générer une transcription à partir d'un spectrogramme jouant ainsi le rôle du modèle acoustique et du modèle de langage. Ce module a donc pour but d'abstraire la difficulté de la tâche de reconnaissance et simplifier le pipeline du développement et ce, en se basant sur des techniques plus récentes d'apprentissage automatique et d'apprentissage profond tel que les réseaux de neurones que nous présentons à travers les deux chapitres suivants.

La figure suivante est un exemple qui a pour but de présenter les étapes de la reconnaissance d'un enregistrement audio pour l'approche End-To-End.

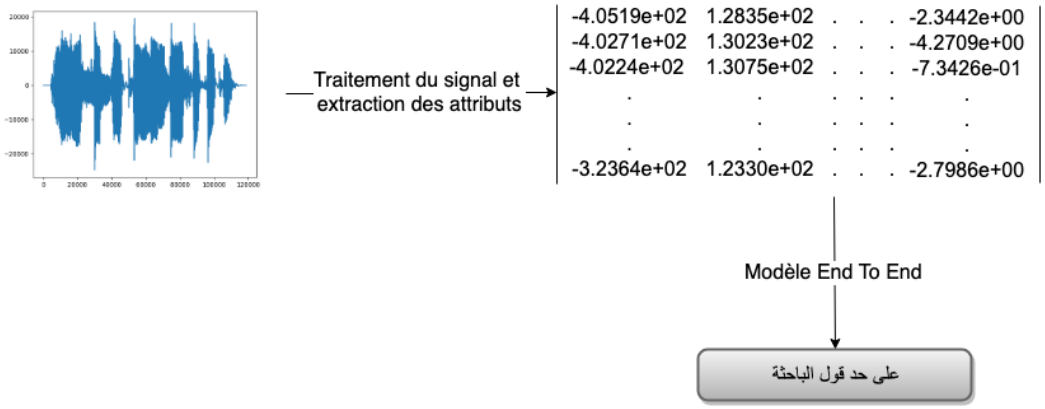

FIGURE 1.5 – Exemple de reconnaissance avec l'architecture End-To-End

1.3 Systèmes de synthèse vocale

1.3.1 Présentation

Les systèmes de synthèse vocale ou, Text-To-Speech en anglais, sont utilisés pour convertir des mots écrits (dans un document par exemple) en discours audible et doivent êtres capables de lire n'importe quel texte. Ces systèmes sont largement utilisés de nos jours et il y a de plus en plus d'applications qui fournissant des services de synthèse vocale. L'engouement pour ces systèmes est dû aux possibilités que ceux-ci offrent [Akbaraly, 2019], parmis ces possibilités nous citons :

- l'aide aux mal-voyants,
- l'aide à l'assimilation des connaissances,
- l'aide à la traduction et à la prononciation, et
- les assistants intelligents.

Comme pour la tâche de reconnaissance de la parole, il existe une approche qui se base sur la synthèse des phonèmes et une approche bout à bout. Nous présentons dans ce qui suit ces deux approches en mettant l'accent sur la complexité de conception d'un système basée synthèse de phonèmes par rapport à un système bout à bout.

1.3.2 Architecture d'un système de synthèse vocale basée synthèse de phonèmes

Les systèmes basés synthèse de phonèmes se basent sur deux parties principales : conversion du texte en phonèmes puis, conversion des caractéristiques linguistiques des phonèmes en discours [Amrouche et al., 2017]. Nous présentons dans ce qui suit quelques généralités sur cette approche. Un système de synthèse vocale, indépendamment du texte et de la langue , se compose des modules de base suivants [O. Arik et al., 2017] :

- **Conversion de graphème vers phonèmes** : Permet de convertir un texte écrit en phonèmes et ce en utilisant un alphabet phonétique qui est un ensemble de symboles ou de codes utilisés pour indiquer le son d'une lettre (comme Arpabet [Arpabet, 2017] par exemple).
- **Modèle de segmentation** : il localise les temps de début et fin de chaque phonème et permet d'effectuer un alignement entre les phonèmes ainsi que les lettres présentes présentes dans le catalogue de voix disponible.
- **Modèle de fréquence fondamentale** : il prédit si un phonème est exprimé et si tel est le cas, il retourne la fréquence du signal correspondante à ce phonème selon la durée de ce dernier.
- **Modèle de synthèse audio** : il combine les sorties des modèles précédents et renvoie en sortie une synthèse vocale du texte désiré.

1.3.3 Architecture End-To-End d'un système de synthèse vocale

L'architecture bout à bout [Wang et al., 2017] se compose de deux modules principaux comme le montre la figure 1.6 :

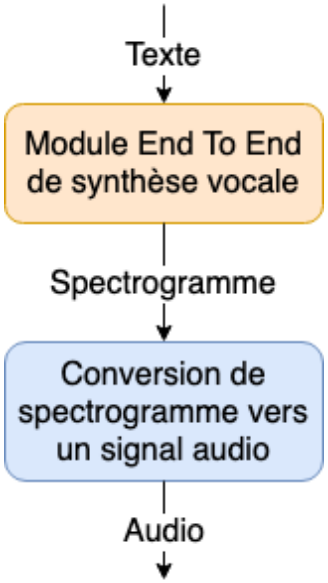

FIGURE 1.6 – Architecture End-To-End d'un système de synthèse vocale

Cette architecture est donc composée des :

- **Module End-To-End de synthèse vocale** : qui prend un texte en entrée et génère à partir de ce texte un spectrogramme. Nous pouvons considérer ce processus comme le processus inverse que celui présenté dans le module séquence à séquence pour les systèmes de reconnaissance de la parole.

- **Conversion de spectrogramme vers signal audio** : ce module permet de convertir un spectrogramme en un signal audio générant ainsi un enregistrement.

La figure 1.7 est un exemple de traduction d'un texte vers un enregistrement sonore à l'aide de l'architecture End-To-End.

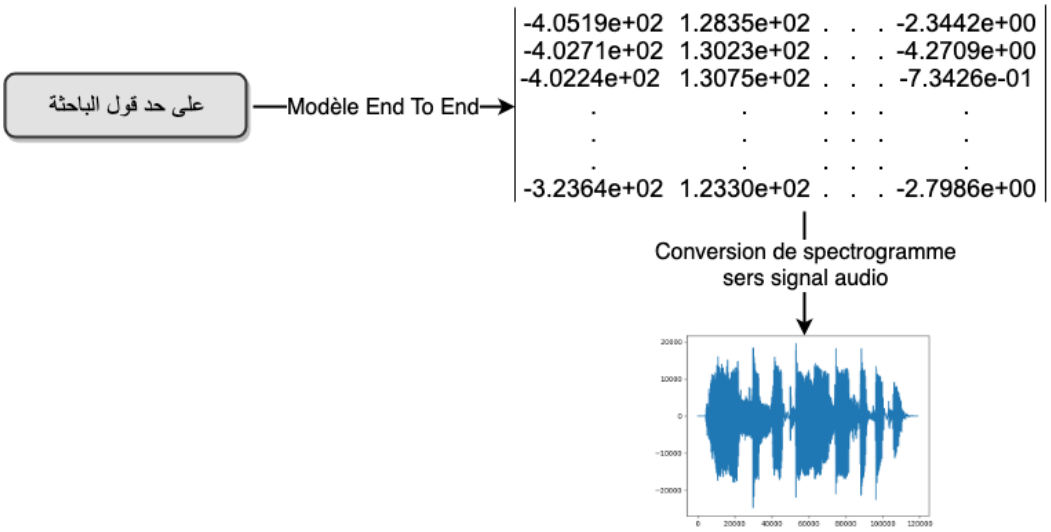

FIGURE 1.7 – Exemple de synthèse vocale avec l'architecture End-To-End

1.3.4 Quelques applications de synthèse vocale

Il existe à ce jour plusieurs applications de synthèse vocale [Fearn, 2018] ; nous présentons dans ce qui suit les meilleures applications en terme de performance et de simplicité d'utilisation.

- **Amazon Polly** : Alexa n'est pas le seul outil d'intelligence artificielle créé par le géant de la technologie Amazon ; il propose également un système de synthèse vocale intelligent appelé Polly [Amazon, 2016] et utilise des techniques avancées d'apprentissage profond. Ce logiciel transforme le texte en discours réaliste et les développeurs peuvent utiliser le logiciel pour créer des produits et des applications embarquant des systèmes de traitement de la parole. Amazon Polly comporte une API permettant d'intégrer facilement les fonctionnalités de synthèse vocale dans des livres électroniques, des articles et d'autres supports. Pour convertir du texte en parole, il suffit de l'envoyer par l'intermédiaire de l'API, qui enverra un flux audio directement à votre application. Amazon Polly est disponible pour plusieurs langues dont l'anglais, le français, le portugais, le japonais et le mandarin.

- **Voice Reader Home** : basée en Allemagne, Linguatec est une autre société qui crée des applications de synthèse vocale depuis plusieurs années. À présent dans sa 15ème édition, Voice Reader [Linguatec, 2016] peut convertir rapidement du texte en fichiers audio et peut convertir rapidement des textes tels

que des documents Word, des courriels, des EPUB et des PDF pour les écouter sur un PC ou un appareil mobile par la suite. Cette application a l'avantage d'offrir 67 voix différentes et prendre en charge jusqu'à 45 langues telles que l'anglais, l'arabe, le français, l'espagnol, l'italien, le danois et le turc.

- **Capti Voice** : Les applications de synthèse vocale sont également populaires dans le monde de l'éducation, où elles sont notamment utilisées pour améliorer la compréhension. Positionnée comme une solution d'aide à la lecture hors ligne et en ligne, Capti Voice dans sa version 2.0 [Capti, 2016] est utilisée par un grand nombre d'écoles, de collèges, d'entreprises et de professionnels du monde entier. Soutenant plus de 20 langues, dont la langue arabe, l'application peut être utilisée pour améliorer le vocabulaire ainsi que dans le cadre de stratégies de lecture active. Elle peut décrire un éventail de contenus, notamment des livres électroniques, des articles et des pages web. Capti Voice a cependant le désavantage de proposer des licences chères ce qui rend son utilisation très contraignante dans les milieux démunis.

- **Natural Reader** : c'est une application de synthèse vocale basée sur le cloud et destinée d'avantage à un usage personnel. Natural Reader [NaturalReader, 2017] permet de convertir des textes écrits tels que des documents Word et PDF, des livres électroniques et des pages Web en des discours continus. Etant basée sur la technologie du cloud, cette application peut accéder au service de synthèse vocale quel que soit le support du moment que l'application est connectée au même service cloud. Natural Reader propose le service de synthèse vocale pour plusieurs langues dont : l'anglais, l'arabe, l'allemand, le français et le mandarin.

1.3.5 Discussion

À travers cette présentation, nous notons que le développement d'un système basé synthèse de phonèmes peut s'avérer plus complexe car il nécessite en premier lieu la collecte d'un corpus de données contenant une description des temps de début et de fin de chaque phonème et la collecte d'un tel corpus peut s'avérer délicate comparée à la collecte d'un simple corpus d'enregistrements sonores et transcriptions pour l'approche End-To-End. En plus de cela, l'approche End-To-End présente un développement plus intuitif et plus prometteur grâce à l'avancée des technologies qui y sont utilisées.

Nous nous contentons de ces généralités en ce qui concerne les architectures des systèmes de synthèse vocale car le corps de notre travail consiste à développer un système de reconnaissance de la parole et son application sur une problématique donnée. Nous introduisons dans ce qui suit les systèmes de questions-réponses, leurs différents types ainsi qu'une présentation de quelques exemples de ces systèmes.

1.4 Les systèmes de questions-réponses

1.4.1 Présentation

Les systèmes de questions-réponses sont des systèmes intelligents. Selon [Stupina et al., 2016], un système de questions-réponses est un système d'information capable de comprendre des questions et y répondre de manière précise dans un langage naturel humain comme par exemple donner une réponse à la question "Quels sont les meilleurs laboratoires de recherche en intelligence artificielle ?"

Les systèmes de questions-réponses sont généralement classifiés selon le domaine auquel ils appartiennent ainsi que le type de questions.

1.4.2 Classification des systèmes de questions-réponses par domaine

Nous commençons par introduire les deux domaines auxquels peuvent appartenir les systèmes de questions-réponses. D'après [Reddy and Madhavi, 2017] ceux-ci peuvent être ouverts ou fermés.

1.4.2.1 Domaine ouvert

Les systèmes de questions-réponses de domaine ouvert sont des systèmes qui ne sont pas limités à un domaine spécifique. En d'autres termes, pour cette classe de systèmes, l'utilisateur a la possibilité d'introduire la question qu'il désire dans le système étant donné que ce type de systèmes recherche généralement les réponses dans une grande collection de documents comme les recherches sur le web à titre d'exemple.

1.4.2.2 Domaine fermé

Les systèmes de questions-réponses appartenant au domaine fermé consistent en un référentiel limité de questions spécifiques à un domaine (le domaine médical à titre d'exemple) et peuvent répondre à un nombre limité de questions. La qualité des réponses dans le domaine fermé est élevée, les réponses sont obtenues à partir de données structurées (telles que des bases de données), des données semi-structurées (par exemple, des textes annotés au format XML) ou des données non structurées (textes libres).

1.4.3 Classification des systèmes de questions-réponses par types de questions

Le type de la question posée par un utilisateur est très important car il permet de déduire, à priori, le type de réponse attendue et ainsi de générer cette dernière d'une

manière plus efficace. Les différents types de questions posées par les utilisateurs sont classés d'après [Mishra and Jain, 2015] en :

- questions de type factoïde,
- questions de type liste,
- questions de confirmation, et
- questions de causalité.

Prenons pour exemple la question "Qui était le premier président de l'Algérie ?", le domaine de la question posée par l'utilisateur est l'histoire de l'Algérie, la réponse attendue est une entité nommée et plus précisément le nom d'une personne qui est "Ahmed Benbella". Cette question est de type factoïde.

Dans le tableau 1.1 nous présentons les principaux types de questions.

Type	**Réponse attendue**	**Exemple**
Factoïde	Entité nommée (nom d'une personne nom d'une organisation, lieu ou date)	**Question** : Quelle est la capitale de l'Algérie ? **Réponse** : Alger.
Liste	Une liste d'entités nommées	**Question** : quels sont les pays les plus propres du monde ? **Réponse** : Suède, Nouvelle Zélande, Australie, Canada.
Définition	Information concernant une entité nommée	**Question** : Qui est Thomas Pesquet ? **Réponse** : un spationaute français de l'Agence spatiale européenne.
Confirmation	Les réponses peuvent être de différents types : Oui/Non Arguments, etc.	**Question** : Est-ce-que l'USTHB se situe en Algérie **Réponse** : Oui.
Causale	Des explications sur une entité.	**Question** : Pourquoi les abeilles produisent du miel ? **Réponse** : Les abeilles fabriquent du miel pour nourrir les larves et surtout pour avoir des réserves de nourriture en hiver.

TABLE 1.1: Classification des questions par type de réponse

1.4.4 Architecture des systèmes de questions-réponses

L'architecture type d'un système de questions-réponses (voir la figure 1.8) se compose de trois modules principaux [Abouenour, 2014] : un module d'analyse des questions, un module de traitement des documents et un module d'extraction et validation de la réponse.

1.4.4.1 Module d'analyse des questions

Dans ce module, la question posée par l'utilisateur est analysée afin d'identifier son type, d'en extraire les caractéristiques et la structure de la réponse attendue et de déterminer les contraintes qui pèsent sur la réponse attendue.

1.4.4.2 Module de traitement des documents

Ce module est un composant essentiel du système de questions-réponses. Son rôle est d'extraire les documents et les passages pertinents pour la question à partir d'une collection de données (Web ou autre) pour effectuer ensuite un processus de classement qui permettra d'améliorer la pertinence des passages candidats qui correspondent le mieux à la question de l'utilisateur.

1.4.4.3 Module d'extraction et validation de la réponse

Dans les systèmes de questions-réponses, ce module peut être conçu pour extraire la réponse à partir d'un ou plusieurs passages. Ce module ne renvoie la réponse que si les passages candidats fournis par le module de traitement des documents sont pertinents et contiennent réellement la réponse. Certains systèmes intègrent également la validation des réponses dans ce module.

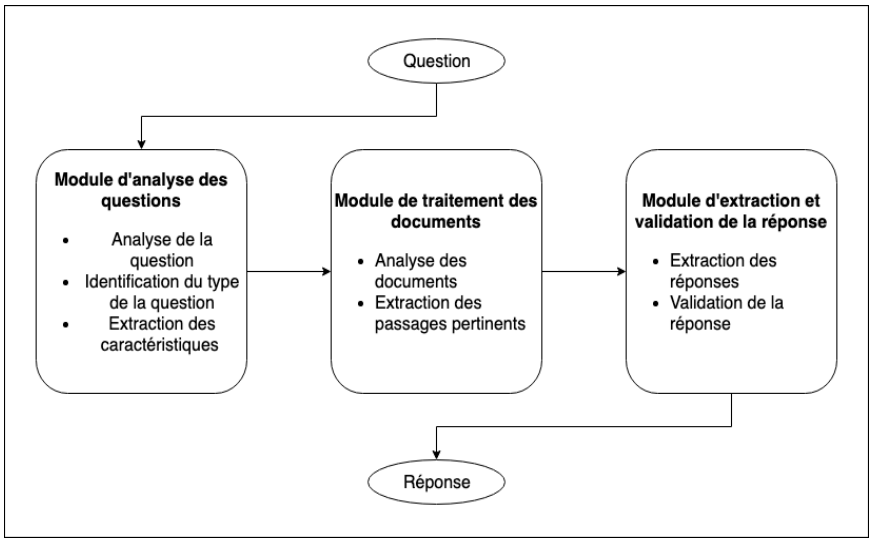

FIGURE 1.8 – Architecture générique d'un système de questions-réponses

1.4.5 Quelques systèmes de questions-réponses

Les systèmes de questions-réponses pour la langue arabe ne sont pas aussi nombreux que pour l'anglais par exemple, ce qui peut s'expliquer par le nombre limité de corpus de qualité existants.

Dans ce qui suit, nous présentons quelques exemples de systèmes de questions-réponses pour la langue anglaise et la langue arabe.

1.4.5.1 Start

START est un des systèmes de questions-réponses les plus connus pour la langue anglaise. Développé par Boris Katz et ses collaborateurs du groupe InfoLab du laboratoire d'informatique et d'intelligence artificielle du MIT, START est en ligne depuis 1993 [Katz et al., 2006]. Actuellement, ce système peut répondre à des millions de questions en anglais en langage naturel. Les réponses sont en format texte ou multimédia qui sont tirés d'un ensemble de ressources d'informations hébergées localement ou accessibles à distance via Internet.

1.4.5.2 DefArabicQA

DefArabicQA [Trigui et al., 2010] est un système de questions-réponses de définition, en d'autres termes ce système traite les questions du type ما هو ؟ ou ما هي ؟ . Il identifie les définitions candidates en utilisant un ensemble de modèles lexicaux [5], filtre ces définitions candidates en utilisant des règles heuristiques qui se déduisent de l'observation d'un ensemble de définitions candidates annotées [6] et les hiérarchise en utilisant une approche statistique. Le type de réponse attendu est déduit par la suite du pronom interrogatif de la question.

1.4.5.3 Al-Bayan

Al-Bayan [Mohamed et al., 2015] est un système de questions-réponses spécifique au Coran. Il prend une question en arabe en entrée et récupère des versets accompagnés de leur *tafsir* (interprétation des versets) sémantiquement pertinents en tant que passages candidats.

La spécificité de ce système réside dans le fait qu'il automatise le processus de sélection des réponses. Les réponses candidates sont ensuite classées en :

- Réponses qui répondent directement à la question.
- Réponses qui peuvent être utiles.
- Réponses qui ne sont pas pertinentes.

5. Un modèle lexical est une séquence de chaînes de caractères (par exemple, des mots, des lettres et des signes de ponctuation) qui fournissent un contexte pour identifier les réponses exactes [Trigui et al., 2010].

6. Un ensemble de définitions candidates divisées en définitions de candidats incorrectes et correctes

1.5 Conclusion

Au vu de l'importance des systèmes de reconnaissance de la parole, l'étude de ces derniers et leur implémentation pour la langue arabe dans des applications telles que les systèmes de questions-réponses est devenu un besoin pour le monde arabe en particulier et pour les utilisateurs de l'arabe de manière générale. Au cours de ce chapitre, nous avons abordé les aspects de bases des systèmes de reconnaissance de la parole, systèmes de synthèse vocale ainsi que des systèmes de questions-réponses comme cas d'application pour la reconnaissance de la parole.

Dans le chapitre suivant, nous introduisons les différentes approches utilisées pour le développement d'un système de reconnaissance de la parole pour ensuite présenter les travaux faisant état de l'art tant pour la langue arabe que pour les autres langues afin de nous guider à concevoir notre propre système.

CHAPITRE 2

TRAVAUX SUR LES SYSTÈMES DE RECONNAISSANCE DE LA PAROLE

2.1 Introduction

L'utilisation des systèmes de reconnaissance de la parole permet d'accéder à l'information de manière rapide et intuitive. Aujourd'hui, beaucoup de nos appareils électroniques ont une exploitation plus naturelle car ils embarquent des fonctionnalités de reconnaissance de la parole. Cependant, nous remarquons que ces systèmes traitent rarement la langue arabe de manière efficace.

Dans ce chapitre nous présentons l'état de l'art pour les systèmes de reconnaissance de la parole et notamment pour la langue arabe. Mais avant cela, nous définissons les différentes techniques et approches utilisées pour la conception des systèmes End-To-End et des systèmes basés reconnaissance de phonèmes pour la reconnaissance de la parole.

2.2 Techniques utilisées pour l'approche End-To-End

[Mitchell, 1997] définit l'apprentissage automatique comme étant un programme informatique qui apprend d'une expérience E à l'égard de certaines classes de tâches T et de la mesure de performance P. Si sa performance aux tâches en T, la performance mesurée par P, s'améliore avec l'expérience E.
En d'autres termes, l'apprentissage automatique est une technique d'intelligence artificielle basée sur différentes notions mathématiques et plus particulièrement statistiques. Elle permet aux ordinateurs d'apprendre à partir des données et résoudre des problèmes ou effectuer des tâches complexes sans être explicitement programmés à cet effet [Goodfellow et al., 2016].

L'apprentissage automatique se présente sous plusieurs types et ce en fonction du problème et du jeu de données à disposition. Ces types sont l'apprentissage supervisé,

l'apprentissage non supervisé et l'apprentissage par renforcement [Shalev-Shwartz and Ben-David, 2013]. Pour la conception de notre système, nous nous intéressons au paradigme d'apprentissage supervisé.

L'apprentissage est dit supervisé lorsque le système apprend à classer des données à partir d'un algorithme d'entraînement [1], traduit de "training" en anglais, ainsi que des exemples de données connus [Henrik et al., 2016]. Ce type d'apprentissage se divise en deux parties :

- la première correspond à générer un modèle ayant la capacité de prédire la classe d'un ensemble de données qu'il rencontre pour la première fois. La génération de ce modèle se fait grâce à un algorithme d'apprentissage ainsi qu'à des données étiquetées (labelisées), et
- la deuxième consiste à prédire l'étiquette d'une nouvelle donnée dont nous ignorons le label à partir du modèle préalablement généré.

Comme mentionné dans le chapitre précédent, l'architecture End-To-End se base sur différents types de réseaux de neurones pour permettre de prédire la transcription associée à une entrée audio. Ce choix d'algorithme d'apprentissage, en plus des avantages classiques des réseaux de neurones, se justifie par :

- la capacité d'apprendre et de modéliser des relations complexes et non-linéaires ce qui est le cas lorsqu'il s'agit d'associer du texte à de la parole, et
- contrairement à beaucoup d'autres techniques, certains réseaux de neurones tels que les réseaux récurrents n'imposent pas de restriction en terme de taille des données aux variables d'entrée. Dans notre cas, les données sont des séries chronologiques, traduit de Time Series en anglais, et se présentent sous forme de séquences de taille variable. Étant donné leur capacité à apprendre des relations cachées entre les données, les réseaux de neurones sont la solution de prédilection à notre problématique.

Nous présentons dans ce qui suit les différents types de réseaux de neurones ainsi que ce qu'ils peuvent apporter à notre tâche.

2.2.1 Réseaux de neurones artificiels

Les réseaux de neurones artificiels, ou Artificial Neural Networks (ANN) en anglais, sont des réseaux fortement connectés de processeurs élémentaires fonctionnant en parallèle. Chaque processeur élémentaire calcule une sortie unique sur la base des informations qu'il reçoit [Touzet, 2019]. Ces réseaux artificiels sont inspirés de l'interconnexion des neurones chez les êtres vivants et sont composés d'une succession de couches où chacune prend ses entrées depuis les sorties de la couche précédente. De base, tout réseau de neurones est composé d'une couche d'entrée, une ou plusieurs couches cachées et une couche de sortie comme le montre la figure suivante [Touzet, 2019] :

1. Nous utilisons pour la suite de ce document l'expression : Algorithme d'apprentissage

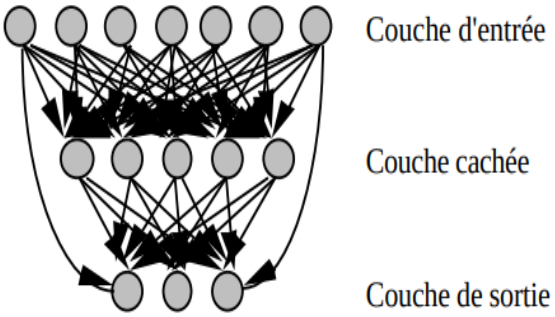

FIGURE 2.1 – Couches d'un réseau de neurones

Il existe plusieurs catégories de réseaux de neurones artificiels, [Touzet, 2019] les classifie en :

- **Réseaux multicouche** : Les connexions se font entre les neurones des couches en aval, et chaque neurone d'une couche est connecté à tous les neurones de la couche suivante et celle-ci seulement.
- **Réseaux à connexions locales** : Chaque neurone entretient des relations avec un nombre réduit et localisé de neurones de la couche avale.
- **Réseaux à connexion complète** : Structure d'interconnexion la plus générale où chaque neurone est connecté à tous les neurones du réseau.

2.2.2 Réseaux de neurones profonds

Les réseaux de neurones profonds, traduits de l'anglais Deep Neural Networks (DNN), sont des réseaux qui se composent de plusieurs couches cachées où chaque type de couche présente des avantages pour des cas d'utilisation spécifiques. Cependant, quelque soit l'architecture interne de ces réseaux, ces derniers partagent la même intuition des réseaux de neurones qui consiste à introduire des données et entraîner le modèle à interpréter ces dernières et à prédire un label qui, dans notre cas, est la transcription d'une entrée audio.

Dans le cas de l'architecture End-To-End du système de reconnaissance de la parole, nous nous intéressons aux réseaux de neurones récurrents et réseaux à convolutions.

2.2.2.1 Réseaux de neurones récurrents

Les réseaux de neurones récurrents, ou Recurrent Neural Networds (RNN) en anglais, font partie d'un groupe plus large d'algorithmes appelés modèles de séquence. Les modèles de séquence ont fait des pas de géant dans les domaines qui traitent des données séquentielles comme la reconnaissance de la parole, la génération de musique, l'analyse des séquences ADN ou encore la traduction automatique [Peixeiro, 2019]. Les réseaux de neurones récurrents sont idéals pour appréhender ce genre de

problèmes car contrairement aux réseaux de neurones classiques (ou encore les réseaux à convolutions que nous présentons dans ce qui suit), ces réseaux sont conçus pour prendre une série d'entrées sans déterminer leur taille au préalable.
Le point fort des RNNs reste néanmoins leur capacité à mémoriser les suites de séquences de données ce qui s'avère essentiel pour la reconnaissance de la parole [Graves et al., 2013]. En effet ces réseaux se souviennent des séquences de données passées et la décision à prendre (classification par exemple) est influencée par ce qui a été appris du passé et ce grâce à une composante de ce réseau qui est le vecteur d'états cachés ou Hidden States Vector en anglais.

La figure suivante [Banerjee, 2018] détaille le flux de l'information dans ce réseau.

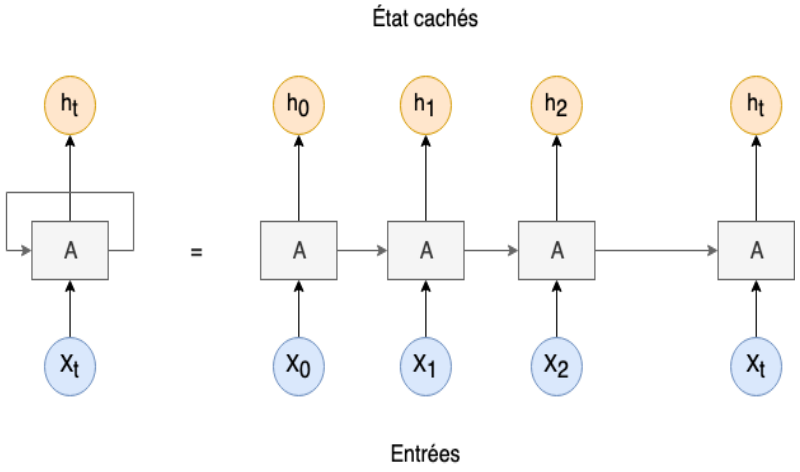

FIGURE 2.2 – Réseau de neurones récurrent

Nous remarquons donc qu'à chaque timestep [2], le RNN génère un output ainsi qu'un vecteur d'états cachés qui fera office d'input pour le prochain timestep.
Le vecteur d'états cachés est une représentation abstraire du séquencement des données et c'est cette information qui permet au réseau de prendre en compte les informations précédentes à l'instant **t**. Il est cependant important de noter que les RNNs souffrent, lorsque la taille des séquences est suffisamment grande, du problème du *Vanishing Gradient* qui arrive lorsque les poids du modèle ne sont plus mis à jour et que le gradient avoisine le zero. Il existe néanmoins d'autres réseaux tel que les LSTMs [3] basés sur le principe des réseaux de neurones récurrents et qui évitent le *Vanishing Gradient* [Nguyen, 2018b].

2.2.2.2 Réseaux de neurones à convolutions

Les réseaux de neurones à convolutions, ou Convolutional Neural Networks (CNN) en anglais, ont été introduits par [Lecun et al., 1998] et sont analogues aux réseaux de neurones classiques car ils sont composés de neurones qui s'auto-optimisent par l'apprentissage [O'Shea and Nash, 2015]. L'une des plus grandes limitations des réseaux de neurones traditionnels est qu'ils ne donnent pas de bon résultats avec

2. Timestep : Pas de temps ou découpe temporelle d'une donnée
3. Long Short-Term Memory : type de réseau récurrent que nous détaillons dans le chapitre suivant.

des données multi-dimensionnelles au vu de la complexité de calcul et c'est là où réside la grande différence entres les réseaux à convolutions et les ANNs. Un CNN est capable de capturer avec succès les dépendances spatiales et temporelles des données grâce à l'application de filtres appropriés. L'architecture s'ajuste mieux aux données spatiales ou encore séquentielles en raison de la réduction du nombre de paramètres impliqués comme c'est le cas des images, des vidéos ou encore du son comme dans notre cas. Dans un CNN, deux opérations sont appliquées :

- **Convolution :** Consiste à appliquer un masque de convolution de taille **(M × M)** sur l'image en spécifiant un nombre de filtres afin d'extraire les informations pertinentes sous forme d'un ensemble de matrices (ou de filtres) de taille **(N-M+1 × N-M+1)**.
- **Pooling :** Il existe deux variantes pour cette opération : Pooling Maximum et Pooling Moyen. Cette opération consiste à appliquer un masque de taille **(M × M)** à la matrice et ne garder que le maximum ou la moyenne de ce masque selon que nous appliquons un Pooling Maximum ou moyen et générer un ensemble de matrices de taille **(N/M × N/M)**.

La figure suivante illustre le fonctionnement d'un réseaux de neurones à convolution à travers un exemple numérique pour une image en 2D.

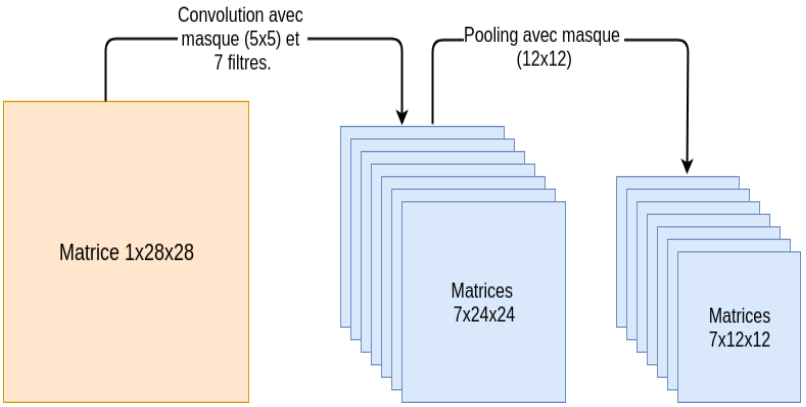

FIGURE 2.3 – Réseau de neurones à convolution

Dans le cadre de ce projet, nous traitons des enregistrements audio qui sont des données de type séries temporelles, ces données sont sous forme de vecteurs d'attributs nous utilisons donc des réseaux de neurones à convolution en 1D [**?**]. La logique de fonctionnement de ces réseaux est la même à la différence que les masques de convolution et de pooling sont des vecteurs en une dimension au lieu de matrices en deux dimensions pour le cas des images.

2.3 Approches utilisées pour les systèmes de reconnaissance basés reconnaissance de phonèmes

2.3.1 Modèle acoustique

2.3.1.1 Modèles de Markov Cachés

Traduit de Hidden Markov Models (HMM), les Modèles de Markov Cachés est une approche probabiliste permettant de représenter des distributions de probabilités à partir d'une chaîne d'observations. Par définition, un modèle de Markov caché est un processus doublement stochastique avec un processus sous-jacent non observable (processus caché) mais pouvant uniquement être observé au moyen d'un autre ensemble de processus stochastiques produit à partir de la séquence d'événements observée [Rabiner and Juang, 1986]. Ces modèles sont basés sur deux propriétés. Premièrement, l'observation à l'instant t a été générée par un processus dont l'état S est caché à l'observateur. Deuxièmement, nous supposons que l'état du processus caché satisfait la propriété de Markov qui dit qu'étant donné la valeur de l'état S_{t-1}, l'état actuel S_t est indépendant de tous les états antérieurs à t - *1*. En d'autres termes, à un moment donné, l'état résume tout ce que nous devons savoir sur l'historique du processus afin de prédire son avenir [Ghahramani, 2001].

Afin d'illustrer ce que nous venons d'expliquer nous prenons l'exemple suivant [Tulyakov, 2014] :

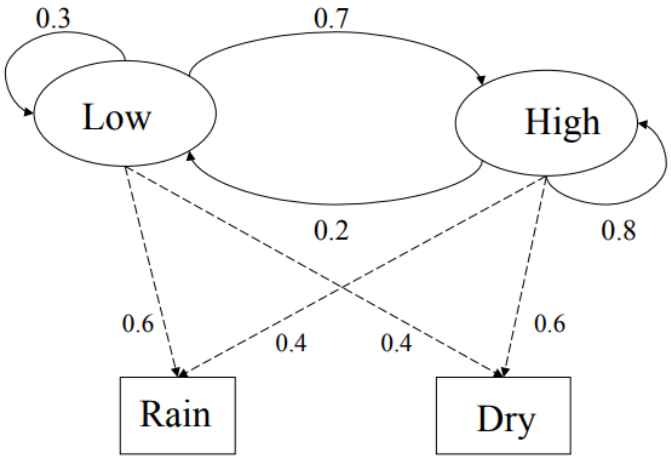

FIGURE 2.4 – Exemple Modèle de Markov Caché

- High et Low sont les états et Rain et Dry sont les observations.
- Les probabilités de transition sont : P(Low|Low)=0.3 , P(High|'Low)=0.7 ...
- Les probabilités d'obvservation sont : P('Rain'|'Low')=0.6, P('Dry'|'Low')=0.4 ...
- Les probabilités initiales sont : P('Low')=0.4, P('High')=0.6.

Une chaîne de Markov Caché nous permet de calculer la probabilité d'une séquence d'observations. Dans le cas de la reconnaissance de la parole nous utilisons cette propriété pour calculer la probabilité de d'avoir une suite de phonèmes sachant un ensemble d'observations (le spectrogramme) et un ensemble d'états (les phonèmes). Afin de tirer profit des HMMs pour la conception du modèle acoustique, nous avons besoin du vecteur de distributions de probabilités. Pour cela nous pouvons utiliser des Modèles de Mélange Gaussien.

2.3.1.2 Modèles de Mélange Gaussien

Traduit de "Gaussian Mixture Models (GMM)", les Modèles de Mélange Gaussien consistent en une fonction de densité de probabilité paramétrique représentée par la somme pondérée de valeurs Gaussiennes.

Les GMMs sont couramment utilisés comme modèle paramétrique de la distribution de probabilité de mesures ou de caractéristiques continues dans un système biométrique, telles que les caractéristiques spectrales liées au tract vocal dans un système de reconnaissance de la parole [Reynolds, 2009].

Les GMMs permettent de calculer la vraisemblance de variables aléatoires continues offrent ainsi la possibilité de modéliser les distributions de probabilités sur des vecteurs d'entités en entrée associés à chaque état d'un HMM. Malgré tous leurs avantages, les GMMs présentent un grave inconvénient : ils sont statistiquement inefficaces pour modéliser des données situées sur ou à proximité d'une variété non linéaire dans l'espace de données [Hinton et al., 2012]. Dans le cas des systèmes de reconnaissance de la parole, le discours est produit en modulant un nombre relativement réduit de paramètres, ce qui implique que sa véritable structure sous-jacente est beaucoup plus petite que ce qui apparaît immédiatement dans un vecteur contenant des centaines de caractéristiques.

2.3.2 Modèle de langage

Dans un système de reconnaissance de la parole, un modèle de langage peut être créée à partir du principe de N-gramme. Un N-gramme est une suite de N caractères, mots ou même phonèmes qu'on peut retrouver dans une chaîne de caractères ou plus généralement dans un texte [S Jurafsky and Martin, 2000]. À titre d'exemple, la phrase : "ce projet est passionnant" contient :

- Quatre 1-grammes (ou Uni-grammes) qui sont des chaînes de caractère de longueur de un mot à savoir : "ce", "projet", "est" et "passionnant".
- Trois 2-grammes (ou Bi-grammes) qui sont : "ce projet", "projet est" et "est passionnant".
- Deux 3-grammes (ou Tri-grammes) qui sont : "ce projet est" et "projet est passionnant".
- Un 4-grammes qui est la phrase entière.

Le choix du N-gramme à utiliser dépend de la structure de la langue que nous voulons traiter (la langue arabe dans notre cas). Utiliser un N-gramme avec un N trop petit risquerait de ne pas permettre au modèle de langage de détecter les incohérence dans les suites de phonèmes reconnues par le modèle acoustique. Par ailleurs, si la suite de mots est trop longue nous pouvons nous retrouver dans un cas de sur-apprentissage [4] et donc de mémoriser les phonèmes associés à une entrée audio rencontrée pendant la phase d'entraînement mais donner de mauvais résultats pour de nouvelles entrées audio non rencontrées.

2.4 Mesures de performances

Il existe plusieurs mesures possibles pour évaluer la performance d'un système de reconnaissance de la parole. Nous présentons les quatre métriques que nous utiliserons pour la suite de ce mémoire afin d'évaluer différents systèmes et architectures.

2.4.1 Taux d'erreur des mots

Le taux d'erreur de mots, traduit de Word Error Rate (WER) en anglais, est utilisé pour la reconnaissance continue des mots et où la taille d'une séquence n'est pas prédéfinie. Cette mesure est calculée à partir de la formule suivante [Mccowan et al., 2004] :

$$WER = \frac{S + D + I}{N} \tag{2.1}$$

où :
S : nombre de mots qui ont été substitués lors de la reconnaissance.
D : nombre de mots qui n'ont pas été écoutés ou ont plutôt été supprimés.
I : nombre de mots qui ont été rajoutés sans qu'ils soient présents dans la parole.
N : nombre total de mots de l'entrée.

2.4.2 Taux d'erreur de caractères

Le taux d'erreur de caractères, traduit de Character Error Rate CER en anglais, est une mesure similaire à la métrique WER. Celle ci est généralement utilisée pour mesurer la performance de l'architecture End-To-End où le résultat de la reconnaissance peut être une suite de caractères. Cette mesure est calculée à partir de la formule suivante :

$$CER = \frac{S + D + I}{N} \tag{2.2}$$

où :
S : nombre de caractères qui ont été substitués lors de la reconnaissance.

4. Désigne le fait qu'un modèle s'adapte bien aux données d'apprentissage et présente de très bons résultats mais, présente des résultats bien inférieurs une fois face à des données qu'il n'a jamais croisé [Guyon and Yao, 1999].

D : nombre de caractères qui n'ont pas été écoutés ou ont plutôt été supprimés.
I : nombre de mots qui ont été rajoutés sans qu'ils soient présents dans la parole.
N : nombre total de caractères de l'entrée.

2.4.3 Exactitude des mots

L'exactitude des mots ou, Word Accuracy en anglais, est une mesure que nous utilisons lors de l'entraînement de notre modèle afin de visualiser la performance de ce dernier et ainsi modifier les paramètres utilisés en fonction du besoin. Cette mesure est calculée à partir de la formule suivante :

$$WAcc = 1 - WER = \frac{N - S - D - I}{N} = \frac{H - I}{N} \tag{2.3}$$

où :
S : nombre de mots qui ont été substitués lors de la reconnaissance.
D : nombre de mots qui n'ont pas été écoutés ou ont plutôt été supprimés.
I : nombre de mots qui ont été rajoutés sans qu'ils soient présents dans la parole.
N : nombre total de mots de l'entrée.
H : nombre de mots correctement reconnus.

2.4.4 Exactitude des caractères

L'exactitude des caractères ou, Character Accuracy en anglais, est une mesure que nous utilisons lors de l'entraînement de notre modèle afin de visualiser la performance de ce dernier et ainsi modifier les paramètres utilisés en fonction du besoin. Cette mesure est calculée à partir de la formule suivante :

$$CAcc = 1 - CER = \frac{N - S - D - I}{N} = \frac{H - I}{N} \tag{2.4}$$

où :
S : nombre de caractères qui ont été substitués lors de la reconnaissance.
D : nombre de caractères qui n'ont pas été écoutés ou ont plutôt été supprimés.
I : nombre de caractères qui ont été rajoutés sans qu'ils soient présents dans la parole.
N : nombre total de caractères de l'entrée.
H : nombre de caractères correctement reconnus.

2.5 Systèmes de reconnaissance de la parole les plus performants

Nous effectuons une comparaison entre les deux types de système de reconnaissance de la parole en commençant par présenter les systèmes basés reconnaissance de phonèmes.

2.5.1 Systèmes basés reconnaissance de phonèmes

Nous présentons dans cette partie les systèmes de reconnaissance de la parole qui sont basés reconnaissance de phonèmes les plus performants [Matarneh et al., 2017].

2.5.1.1 Dragon Mobile SDK

Dragon Mobile SDK est un système closed source [5] Développé par Nuance Communication [Nuance, 2019], leader mondial du développement de systèmes de reconnaissance de la parole. Ce système client serveur est apprécié pour sa documentation riche ainsi que le nombre important de frameworks [6] qu'il propose. Ce système utilise la plate-forme Speech Kit, également développée par Nuance communication, permettant une simple intégration des systèmes de reconnaissance et synthèse vocale. Cette plateforme fournit ces services à travers un accès asynchrone à des serveurs minimisant ainsi les coûts d'exécution en terme de temps et de ressources matérielles. Dragon Mobile SDK [7] [Nuance, 2019] est l'un des systèmes les plus performants pour la langue anglaise avec un taux d'erreur avoisinant les 1% [Matarneh et al., 2017] mais a le gros désavantage de ne permettre qu'un nombre limité de requêtes par jour dans sa version gratuite.

2.5.1.2 Google Speech Recognition

Produit du géant Google, Google Speech Recognition est un système closed source présent sur tout type de support sous système Android ou tout simplement intégrant l'API [Google, 2019a]. Cette dernière a un taux d'erreur de mots (WER) d'environ 2% [Google, 2019b], un temps de réponse moyen très court et a l'avantage de ne pas limiter le nombre de requêtes par jour, ce qui en fait l'un des systèmes de reconnaissance de la parole les plus utilisés [Matarneh et al., 2017].

2.5.1.3 Microsoft Speech API

Microsoft Speech API est le système de reconnaissance de la parole closed source développé par Microsoft [Azure, 2019] et offrant une interface appelée Microsoft Speech Application Programming Interface (SAPI) qui, combiné au Microsoft Speech SDK, offre des services de commande vocale. Cette API comprend un ensemble de méthodes et de données efficaces et bien intégrées au .NET Framework, fournissant ainsi une plate-forme de développement accessible et facile d'intégration d'où l'avantage de ce système. Bien que la Microsoft Speech API soit semblable à la Google Speech Recognition API en terme d'accessibilité et de temps de réponse, celle ci

5. Un logiciel Closed Source est un logiciel dont le code source est gardé en toute sécurité et crypté. Cela signifie que l'utilisateur ne peut pas copier, modifier ou supprimer des parties du code sans conséquence. Cela peut aller de l'annulation de la garantie à des répercussions juridiques

6. désigne un ensemble cohérent de composants logiciels structurels, qui sert à créer les fondations ainsi que les grandes lignes de tout ou d'une partie d'un logiciel (architecture)

7. Software Development Kit (SDK) : est un ensemble d'outils logiciels destinés aux développeurs et facilitant le développement d'un logiciel sur une plateforme donnée

présente néanmoins un taux d'erreur de 5.1% [Microsoft, 2017] supérieur à celui fournit par le produit de Google.

2.5.1.4 Kaldi Speech Recognition Toolkit

Kaldi Speech Recognition Toolkit est un système open source [8] qui propose un grand nombre d'approches modernes couramment utilisées pour la reconnaissance de la parole permettant ainsi d'utiliser divers algorithmes pour réduire la taille des caractéristiques du signal acoustique et ainsi améliorer les performances du système [Kaldi, 2011]. Kaldi est écrit en C ++ et possède une structure très modulaire, ce qui permet d'ajouter facilement de nouvelles fonctions et de corriger rapidement les erreurs. Ce système prend en charge différentes plateformes, mais ne fournit qu'une console qui complique son intégration à d'autres applications. Par défaut, Kaldi prend uniquement en charge la langue anglaise et fournit une documentation détaillée orientée aux lecteurs expérimentés dans le domaine de la reconnaissance de la parole.

Parmi les systèmes de reconnaissance de la parole open source, Kaldi Speech Recognition Toolkit est celui qui présente le taux d'erreur le moin élevé qui est d'environ 6.5% [Belenko and Balakshin, 2017] et la vitesse de reconnaissance les plus élevées, ainsi que des algorithmes et structures de données de pointe.

2.5.1.5 CMU Sphinx

L'un des systèmes open source les plus répandus est sans doute le CMU Sphinx ou simplement Sphinx, développé par le groupe Xuedong Huang de l'Université Carnegie Mellon [Sphinx, 2017]. Ce système inclut un ensemble de modèles acoustiques et modèles de langage pour certaines langues dont : Anglais, Français, Allemand, Mandarin et Espagnol. Du fait que ce soit un système Open Source, ce dernier fournit des fonctions pour créer et entraîner nos propres modèles acoustiques et modèles de langage pour les langues non disponibles ou même améliorer celles disponibles selon le domaine d'application. Sphinx présente une taux d'erreur de 21.4% [Belenko and Balakshin, 2017] qui n'est pas satisfaisant et particulier par rapport aux systèmes présentés précédemment cependant, Sphinx mais est le plus rapide en terme de temps de reconnaissance. Ce qui fait la force de CMU Sphinx est le fait qu'il contienne des modules permettant d'améliorer les modèles présents par défaut dans l'API mais surtout permet à l'utilisateur de créer ses propres modèles et ainsi améliorer le taux d'erreur des modèles par défaut de Sphinx.

2.5.1.6 Discussion

Nous pouvons tirer la conclusion que parmi les systèmes Open Source, Kaldi est le plus performant mais Sphinx est le système le plus simple à implémenter.

8. Un logiciel Open Source est un logiciel dont le code source est disponible. Un utilisateur peut lire, utiliser ou même modifier ce code suivant certaines licences. Plus d'informations dans l'annexe de ce mémoire.

Cependant, pour obtenir des résultats satisfaisant avec Sphinx,, un corpus de qualité est nécessaire sinon le taux d'erreur de la reconnaissance sera élevé. Pour ce qui est des systèmes Closed Source, le plus performant est Dragon Mobile SDK parce qu'il convient mieux aux tâches de reconnaissance, offre une bonne documentation et l'API est simple à intégrer. Nous notons par contre les restrictions en terme d'utilisation. Par conséquent, il devient difficile de mettre en œuvre un produit personnalisé basé sur Dragon Mobile SDK et c'est pour cela que Google Speech Recognition API est plus pratique pour sa simplicité d'intégration, son temps de réponse en raison de la grande puissance de calcul, et bien sûr, le nombre illimité de requêtes par jour.

2.5.2 Systèmes basés reconnaissance de phonèmes pour la langue arabe

La langue arabe présente beaucoup de défis et difficultés par rapport à d'autres langues. Parmi ces difficultés nous citons la complexité morphologique et la diacritisation qui est un élément important dans la structure morpho-syntaxique de la langue [Hamdi, 2012].

Les travaux réalisés ne sont pas toujours basés sur le même corpus. Pour la langue arabe, les travaux sont la plupart du temps menés à partir de différents corpus ce qui rend la performance de la reconnaissance difficile à améliorer et peu flexible contrairement à la langue anglaise par exemple où cette tâche se fait à partir de grands corpus communs. Travailler sur des corpus communs permet de gagner du temps et d'améliorer progressivement la qualité des systèmes car les performances peuvent être comparées et de ce fait, être améliorées [Al-Anzi and AbuZeina, 2018].

En plus de la qualité du corpus, il est important de mentionner que la variation de la prononciation dans un discours joue un rôle important dans la capacité à reconnaître des mots. La référence [Strik and Cucchiarini, 1999] résume les difficultés associées à la façon dont les mots sont prononcés tels que l'assimilation, la co-articulation, la réduction, la suppression et l'insertion.

Dans ce qui suit, nous présentons les corpus utilisés, les architectures ainsi que les performances enregistrées par les différents systèmes.

Papier	**Corpus utilisé**	**Modèle Acoustique**	**Modèle de Langage**	**WER**
[Kirchhoff and Stoleke, 2016]	CallHome Corpus	CMU Sphinx HMM based Acoustic Model	N-grammes allant Jusqu'à N=6	55.02
[Alghamdi and Al-Muhtaseb, 2017]	Arabic Broadcast News Corpus	CMU Sphinx HMM based AcousticModel	Bi-grammes et Tri-grammes	13.66
[Abushariah, 2012]	8.043 déclarations recueillies de huit locuteurs environ 8 heures de discours	HMMs basés GMMs	Bi-grammes et grammaire à contexte libre	10.07
[Menacer and Smaili, 2017]	Numlar , NetDC GigaWord Arabic corpus	GMM-HMM DNN-HMM	Bi-grammes et 4-grammes	13.45

TABLE 2.1: Performances des systèmes basés reconnaissance de phonèmes pour la langue arabe

À travers ce tableau récapitulatif, nous voyons clairement que plus le corpus utilisé est riche, plus la performance du système s'améliore.

En plus du corpus, nous remarquons que dans les premières références (avant 2012), les modèles acoustiques se basaient systématiquement des HMMs associés à des GMMs pour la reconnaissance des phonèmes. Cette pratique se voit de plus en plus remplacée par des architectures hybrides HMM-DNN qui présentent de très hautes performances lorsque le système utilise un corpus de qualité pour entraîner les modèles acoustique et de langage et c'est ce qui fait défaut à la langue arabe pour l'instant.

2.5.3 Systèmes de reconnaissance de la parole End-To-End

Nous présentons dans cette partie les travaux faisant état de l'art en ce qui concerne les systèmes End-To-End de reconnaissance de la parole. Étant une approche exploitée que depuis très récemment, il existe moins de travaux proposant de très bon résultats et c'est d'autant plus vrai pour ce qui est de la langue arabe. La tableau suivant est un récapitulatif des travaux, incluant les travaux pour la langue arabe, faisant état de l'art des systèmes End-To-End de reconnaissance de la parole.

Papier	**Corpus utilisé**	**Architecture du modèle**	**WER**
[Awni Hanuun, 2014]	Switchboard Hub5'00, Fisher Corpus WSJ Corpus pour l'anglais	Réseaux de neurones récurrents	11.85
[Dzmitry Bahdanau and Bengio, 2016]	WSJ Corpus pour l'anglais	Réseaux de neurones récurrents avec ajout d'un modèle de langage tri-gram	9.3
[]Dario Amodei, 2016]	12000 Heures pour l'anglais+ 9400 Heures pour le mandarin	Trois couches CNN + 7 couches RNNs + Couche multiconnectée	7.93
[Abdelrahman Ahmed and Toral, 2018]	Al Jazeerah QCRI Corpus de 1200 heures pour l'arabe	Réseaux de neurones récurrent	12.03

TABLE 2.2: Performance des systèmes End-To-End pour la langue Arabe

Nous remarquons à travers ce tableau, que les performances des sytèmes End-To-End sont quelque peu inférieures à celles des systèmes tel que Dragon Mobile SDK ou la Google Speech Recognition API. Cependant, il est clair que ces systèmes sont prometteurs compte tenu de leur performance actuelle, de l'intérêt qu'ils suscitent de la part de la communauté scientifique et des avancées des techniques d'apprentissage profond.

2.6 Conclusion

Il est à présent clair que développer un système de reconnaissance de la parole n'est pas une tâche aisée et requiert des ressources importantes en terme de données ainsi qu'en terme de maîtrise technique. Au cours de ce chapitre, nous avons présenté différents systèmes de reconnaissance de la parole et nous nous sommes inspirés des différentes approches et techniques possibles pour appréhender cette problématique.

Dans le chapitre suivant, nous entamons la conception de notre système à commencer par la préparation de notre corpus, le choix de l'architecture du système de reconnaissance de la parole et son développement, pré-traitement des questions reconnus et tuning [9] de notre système de questions-réponses afin qu'il puisse répondre au mieux à tout type de question.

9. Tuning : Ajustement dans un contexte technique

CHAPITRE 3

CONCEPTION DE ASER-SYSTEM

3.1 Introduction

Comme nous l'avons présenté dans le premier chapitre à travers la section 1.2.4 et la section 1.2.5, il existe deux approches pour développer un système de reconnaissance de la parole : l'approche basée reconnaissance de phonèmes et l'approche End-To-End ; chacune présente un intérêt pour notre problématique, qui est la conception et le développement d'un système de reconnaissance de la parole pour la langue arabe. Le choix de l'approche se fait à travers l'étude des avantages et inconvénients, dans le contexte général puis dans le contexte de ce projet particulièrement, suivie d'une phase d'expérimentations pour appuyer notre raisonnement.

Nous présentons en premier lieu dans ce chapitre un comparatif de ces deux approches et nous justifions notre choix. Nous poursuivons par l'étape de collecte des données que nous utiliserons pour l'apprentissage de *ASeR-System (**A**rabic **S**pe**e**ch **R**ecognition System)*, notre système de reconnaissance de la parole. Nous détaillons par la suite les différentes architectures que nous avons utilisées pour nos expérimentations ainsi que la méthodologie suivie pour définir les différentes architectures de ce système.

3.2 Choix de l'approche pour la conception de ASeR-System

Afin de choisir l'approche que nous allons utiliser pour développer notre système de reconnaissance de la parole, nous étudions les avantages et inconvénients de l'approche End-To-End ainsi que l'approche basée reconnaissance de phonèmes en nous basant sur des critères que nous présentons ci-dessous. Nous commençons par comparer les corpus que nous utiliserons pour chacune des deux approches.

3.2.1 Corpus utilisé

Développer un système de reconnaissance de la parole End-To-End requiert une grande quantité de données par rapport à l'approche basée reconnaissance de phonèmes pour que le modèle puisse comprendre la structure morphologique de la langue. Cependant, la collecte de données pour l'approche End-To-End est plus intuitive car pour former un corpus. En effet, il suffit de récupérer des enregistrements audio et leurs transcriptions ce qui facilite grandement la collecte de données et l'enrichissement des corpus.

Le corpus utilisé pour l'approche basée reconnaissance de phonèmes quant à lui doit contenir les descriptions phonétiques des transcriptions. Il doit également contenir les informations temporelles qui sont les temps de début et de fin de chaque phonème ce qui peut s'avérer plus difficile à préparer.

3.2.2 Comparaison des performances

Nous avions noté dans la présentation de l'état de l'art que les systèmes basés reconnaissance de phonèmes les plus performants tel que Dragon Mobile SDK ou encore Google Speech API présentent des taux d'erreur moin élevés que les systèmes End-To-End. Ceci est dû à la qualité des corpus phonétiques utilisés pour ces systèmes qui ne sont, la plupart du temps, disponibles qu'au sein des laboratoires qui les développent. Nous avons également noté qu'entre 2014, où les systèmes End-To-End furent introduits, et aujourd'hui, les performances de ces derniers se sont largement améliorés se rapprochant ainsi de la performance des systèmes basés reconnaissance de phonèmes. Ceci laisse à penser qu'à l'avenir, avec des corpus plus riches en terme de quantité de données, les systèmes End-To-End seraient aussi performants que l'humain en terme de reconnaissance de la parole. Cette intuition se base sur les performances de l'état de l'art que nous avons rapportées dans les sections 2.5.2 et 2.5.3 de ce document.

3.2.3 Complexité de développement

Le développement d'un système basé reconnaissance de phonèmes requiert :

- une équipe de linguistes pour la collecte d'un corpus qui contient la description phonétique de chaque enregistrement audio. Cette tâche peut s'avérer complexe et fastidieuse,
- le développement d'un modèle acoustique en utilisant les HMMs combinés à des GMMs ou encore à des réseaux de neurones profonds ainsi qu'une connaissance approfondie de la structure morphologique de langue traitée ce qui nécessite un expertise supplémentaire, et
- le développement d'un modèle de langage pour corriger et/ou valider l'interprétation du modèle acoustique.

Comparé à l'approche basée reconnaissance de phonèmes, le développement d'un système End-To-End se limite à la collecte d'un corpus d'enregistrements et de

transcriptions qui peut se faire à l'aide d'algorithmes et sans intervention humaine et au développement d'un modèle Séquence à Séquence [1] en utilisant des techniques d'apprentissage profond.

3.2.4 Capacité d'amélioration des performances

Le bon développement de tout système nécessite que ce dernier soit simple à maintenir et à enrichir. Ce n'est pas forcément le cas d'un système basé reconnaissance de phonèmes où la collecte de données pour le modèle acoustique est une tâche fastidieuse qui nécessite l'intervention de linguistes.

À contrario, un système End-To-End est plus facile à entretenir et surtout à améliorer. En premier lieu, la collecte de données présente moins de difficultés et ne nécessite pas d'intervention humaine. En second lieu, les avancées des différentes techniques d'apprentissage profond et la communauté de développeurs derrière ces techniques font des systèmes End-To-End une approche plus répandue et plus sûre aujourd'hui.

3.2.5 Discussion

Nous déduisons à travers ce comparatif, que chacune des deux approches présente des avantages que l'autre n'a pas. L'approche basée reconnaissance de phonèmes présente de meilleurs résultats avec un corpus moins volumineux grâce aux descriptions phonétiques mais la génération de tels corpus et leur enrichissement est une tâche difficile. Les systèmes End-To-End quant à eux, ne nécessitent pas de descriptions phonétiques pour les corpus, ce qui facilite grandement leur enrichissement et donc l'amélioration des performances de ces systèmes. En plus des données, le développement d'un système End-To-End repose sur des techniques plus performantes et mieux documentées.

Notre choix s'est donc porté sur le développement d'un système de reconnaissance de la parole End-To-End. Dans la suite de ce chapitre, nous présentons les techniques que nous avons utilisées dans ce projet ainsi que les différentes architectures possibles pour ce développement. Nous commençons la conception de notre système par la collecte des données nécessaires à l'apprentissage de nos modèles.

3.3 Environnement de développement pour la reconnaissance de la parole

Dans le cadre de ce projet, nous développons notre propre système End-To-End de reconnaissance de la parole. En plus de ce système, nous nous attelons à la création d'un environnement de développement permettant à toute personne de non

1. Séquence à Séquence : Traduit de l'anglais "Sequence To Sequence", est un paradigme pour le développement de modèles capable de prédire des données séquentielles.

seulement utiliser notre système, mais aussi et surtout de contribuer à son amélioration. Cette initiative est motivée par le fait que la recherche dans le domaine de la reconnaissance de la parole et la compréhension du langage naturel est des plus actives. Ainsi, une telle contribution serait une valeur ajoutée pour la recherche et le développement au niveau des différentes universités, laboratoires ou encore entreprises au niveau national.
Cet environnement doit faire office d'un outil semi-automatisé pour l'utilisation ou l'amélioration d'un système de reconnaissance de la parole et doit proposer les fonctionnalités suivantes :

- Intégration rapide et intuitive du module de reconnaissance de la parole dans toute application.
- Possibilité de choix entre plusieurs modèles pré-entraînés pour la tâche de reconnaissance.
- Module semi-automatisé de nettoyage et présentation des enregistrements audio et des transcriptions.
- Module automatisé de pré-traitement des données.
- Module d'apprentissage profond avec différents modèles pré-définies et possibilité d'ajout de modèles personnalisés.
- Gestion dynamique de la mémoire pour que l'environnement soit performant sur tout type de machine.

La figure 3.1 est un schéma représentant les différentes possibilités qu'offre notre environnement de développement pour la reconnaissance de la parole.

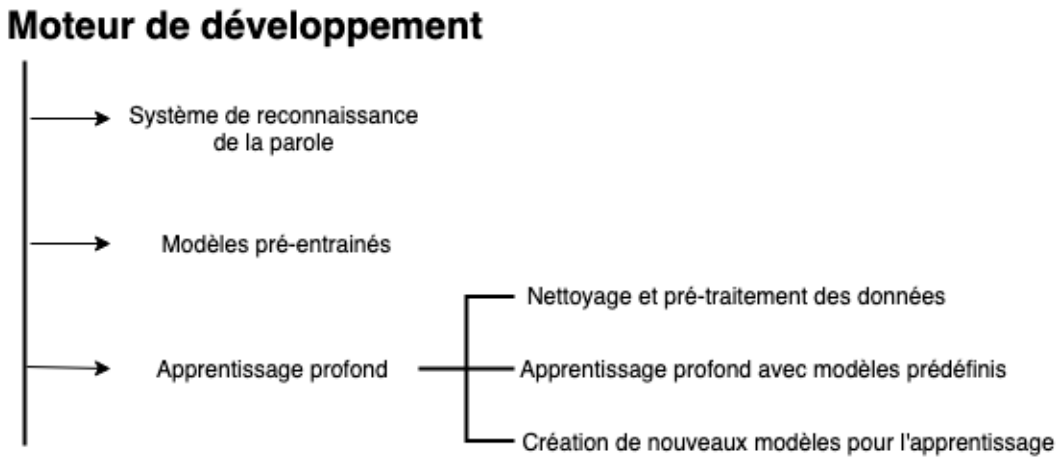

FIGURE 3.1 – Principales fonctions de l'environnement de développement pour la reconnaissance de la parole

Nous détaillons dans le chapitre suivant les différents modules de l'environnement de développement ainsi que leur fonctionnement.

3.4 Collecte et pré-traitement des données

Afin d'entraîner un modèle pour la tâche de reconnaissance de la parole, de grandes ressources en terme de données sont requises. Nous nous sommes inspirés des travaux de [Ahmed et al., 2018] afin d'accéder à des données utilisables pour l'apprentissage de notre modèle.
Avant de parler de l'extraction et du pré-traitement de ces données, il est important de définir les critères [Graves and Jaitly, 2014] que doivent remplir ces dernières pour l'apprentissage. Nos données doivent obligatoirement :

- contenir des enregistrements audio et les transcriptions qui y correspondent,
- être de l'arabe moderne standard et non pas un dialecte,
- contenir un minimum d'une vingtaine d'orateurs, et
- être d'une longueur d'au moins 500 heures de dialogue.

Pour la conception de notre solution, nous testons nos modèles sur différentes tailles de corpus comme nous l'expliquons dans la suite de ce chapitre. Ces corpus sont respectivement constitués de : six heures de dialogues, 60 heures de dialogue, 260 heures de dialogue et 1200 heures de dialogue. Nous nommons corpus d'essai les six heures de dialogue car la source de données pour ce corpus est différente de celle du reste des données ; nous nommons ce second corpus corpus élargi.

3.4.1 Collecte du corpus d'essai

Les six heures de dialogue du corpus d'essai furent une partie du corpus collecté par l'équipe QCRI [2] et est disponible en libre accès sur la plateforme Github [QCRI, 2017]. De cette référence, nous tirons un fichier texte contenant une liste de liens où chacun correspond à un enregistrement audio. Il a donc fallu développer un algorithme d'extraction automatique pour télécharger ces enregistrements. Nous avons également accès à un fichier contenant la transcription associée à chaque enregistrement.
La figure 3.2 résume ce processus d'extraction :

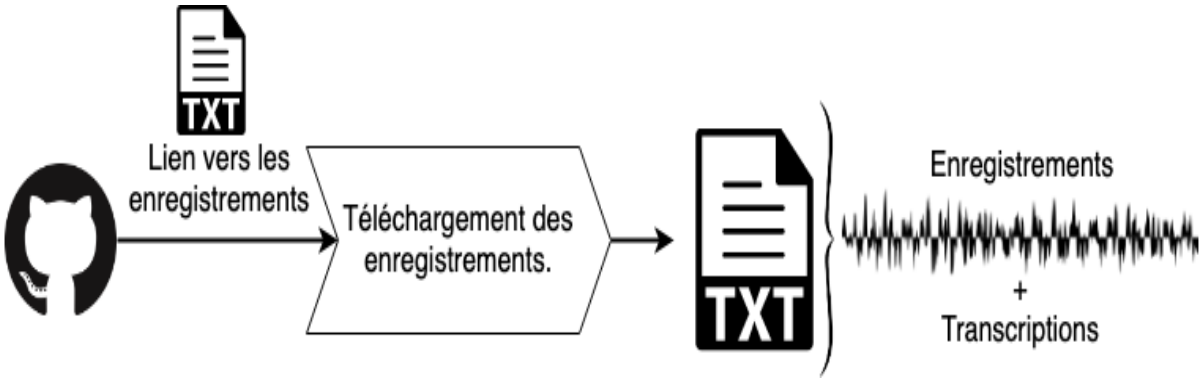

FIGURE 3.2 – Processus d'extraction des six heures de dialogue du corpus d'essai

2. https ://www.qcri.org

Le résultat est un corpus composé de 2219 enregistrements d'Arabe Moderne Standard (Modern Standard Arabic, MSA) enregistrés par une dizaine d'orateurs et où chaque enregistrement est d'une durée qui varie entre six secondes et une minute environ.

3.4.2 Collecte des données du corpus élargi

Pour la suite de la conception du système, nous avons contacté l'équipe du QCRI qui nous ont accordé un accès au corpus MGB-2 [QCRI, 2018]. Ce corpus se compose de 1200 heures et plus d'une centaine d'orateurs de programmes enregistrés de la chaîne télévisée Al-Jazeera. La distribution des orateurs est diverse en terme de genre. La plupart de ces programmes sont des interviews de différentes personnalités et contiennent donc des questions. Il est très important pour nous que le corpus contienne des questions. En effet, *ASeR-System* doit retranscrire ce type d'entrées pour qu'il soit testé, par la suite, avec un système de questions-réponses. Tous les programmes ont été sous-titrés manuellement.

Contrairement aux données du corpus d'essai, celles du corpus élargi sont formalisées d'une manière différente. La première chose que nous remarquons est les tailles des enregistrements qui sont d'une durée qui varie entre 25 et un peu plus d'une heure. La seconde est le format des fichiers qui contiennent les transcriptions qui sont au format XML suivant l'arborescence suivante :

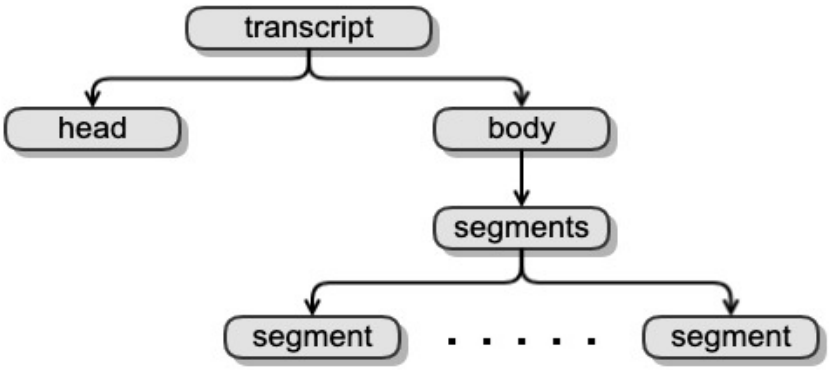

FIGURE 3.3 – Structure des fichiers XML du corpus élargi contenant les transcriptions

Afin de tirer partie de ces données, un pré-traitement est nécessaire. Nous découpons ce pré-traitement en :

- extraction des temps de début et de fin de chaque phrase prononcée par un orateur à partir du fichier XML,
- extraction de la transcription associée à chaque phrase prononcée par un orateur,
- découpage des enregistrements audio selon les temps de début et de fin de phrase, et

- constitution d'un corpus contenant des enregistrements de durée allant de 6 à 40 secondes environ ainsi que la transcription qui y est associée sous forme de fichiers texte.

L'algorithme 1 résume l'opération de collecte de données à partir du corpus élargi :

Algorithme 1 : Extraction des transcriptions et partitionnement des longs enregistrements

Input : repertoire_dataset
`/* repertoir_dataset : le chemin vers le répértoire du dataset */`
Output : `/* Pas de variables en Output */`

liste_pairs_audio_transcriptions ← liste_fichiers(repertoire_dataset)
pour *(audio, fichierXML) ∈ liste_pairs_ audio_ transcriptions* **faire**
    pour *segment ∈ fichierXML* **faire**
        date_debut ← attribut_start_time
        date_fin ← attribut_end_time
        transcription ← generer_transcription(segment)
        liste_transcriptions.ajouter(trancription)
    fin
    generer_fichier_transcriptions(liste_transcriptions)
    liste_enregistrements_audio ← Decouper_audio(audio, date_debut, date_fin)
    generer_fichier_audio(liste_enregistrements_audio)
fin

Maintenant que nous avons ces données à notre disposition, nous pouvons passer au pré-traitement pour qu'elles puissent être utilisées lors de l'apprentissage de nos modèles.

3.4.3 Pré-traitement des données

3.4.3.1 Extraction des caractéristiques des enregistrements

Afin d'utiliser nos enregistrements audio pour la phase d'apprentissage, nous devons extraire les caractéristiques de ces derniers. Il existe différentes manières d'effectuer cette opération mais dans le cadre de la reconnaissance de la parole nous nous intéressons aux caractéristiques MFCC. MFCC est l'acronyme de Mel Frequency Ceptral Coefficient introduit dans les années 1980s [Davis and Mermelstein, 1980]. Le processus par lequel il faut passer pour obtenir les caractéristiques MFCC est le suivant :

(i) Le signal audio passe par un filtre de pré-accentuation pour être découpé en un ensemble de spectres (qui peuvent être considérés comme des images ou plutôt des trames).

(ii) Une transformation de fourrier est ensuite appliquée à chaque image afin de calculer les fréquences des spectres qui sont appelés périodogrammes.

(iii) Calcul des Filter Bank [3] est effectué pour y appliquer ensuite une Transformée en Cosinus Discrète (TCD) aux groupes de filtres en conservant un certain nombre de coefficients résultants tandis que les autres sont ignorés.

(iv) L'étape finale consiste à normaliser les valeurs de ces coefficients.

Dans notre cas, après application du MFCC sur nos données, chaque enregistrement audio sera représenté par une matrice de 40 colonnes où chaque colonne est une caractéristique et chaque ligne est une partie du spectrogramme ; nous appelons cette partie timestep. Cette matrice n'étant pas normalisée, nous passons par une étape de normalisation.

3.4.3.2 Normalisation des spectrogrammes

La normalisation des données est une phase quasi nécessaire pour toute tâche d'apprentissage automatique et notre problématique ne fait pas exception à la règle. Il existe plusieurs méthodes pour normaliser un jeu de données ; nous nous intéressons à la méthode *Min-Max* qui consiste à calculer les valeurs minimum et maximum de chaque attribut du jeu de données pour ensuite appliquer la formule :

$$Normalise(X_i) = \frac{X_i - Min(X)}{Max(X) - Min(X)} \tag{3.1}$$

où :

X : est un attribut du spectrogramme, et
X_i : une valeur de X que nous voulons normaliser.

Dans le cas de notre problématique, le but de cette normalisation est de contenir toutes les valeurs entre 0 et 1 pour assurer une meilleure mise à jour des poids des réseaux de neurones lors de l'apprentissage.

3.4.3.3 Pré-traitement des transcriptions

Le second volet de nos données est l'ensemble des transcriptions. Chaque transcription est un texte écrit en utilisant la translittération Buckwalter [Buckwalter, 1990]. Cette translittération définit un mapping non ambigu de représentation de texte arabe en caractères latins étendus et vice-versa. La première étape du prétraitement consiste à remplacer les caractères présents dans les transcriptions et qui ne font pas partie de l'alphabet Buckwalter. Ces caractères sont les suivants :

- les nombres présents sous la forme de nombres arabes (écrits en graphie occidentale ou orientale) qu'il faut convertir en texte,
- le cas particulier des lettres de l'alphabet où certaines lettres qui devraient être en minuscule selon la table Buckwalter sont écrites en majuscule et où une conversion est nécessaire,

3. Filter Bank : Ensemble de 40 filtres appliqués aux periodogrammes pour extraire les bandes de fréquences.

- le caractère "%" qui doit être remplacé par des caractères alphabétiques, ainsi que
- les caractères "#", ";", "@" et "\\" qui n'apparaissent qu'un nombre dérisoire de fois sur tout le jeu de données.

Ce traitement des caractères spéciaux est important car des caractères superflus et sémantiquement redondants mais syntaxiquement différents augmentent la dimensionnalité des transcriptions. Ceci peut résulter par la suite en un sous-apprentissage. Notons ici, qu'a fin de pouvoir utiliser ces transcriptions lors de l'apprentissage, une conversion vers des valeurs numériques est requise. De ce fait nous présentons deux types d'encodage : encodage basé caractères et encodage basé mots.

3.4.3.4 Encodage des transcriptions basé caractères

La première étape de cet encodage consiste à attacher au début de la transcription le caractère spécial "\t" et le caractère spécial "\n" pour marquer le début et la fin de transcription, respectivement.

La conversion de chaque caractère en valeur numérique se fait en utilisant une table de mappage qui associe à chaque caractère une valeur selon l'ordre d'apparition de celui-ci dans le corpus de transcriptions. Maintenant que chaque transcription est sous forme de valeurs numériques, nous appliquons un encodage appelé *One Hot Encoding* [Zhang and LeCun, 2017] qui consiste à représenter chaque transcription sous forme de matrice binaire où chaque ligne représente un caractère de la transcription et où chaque colonne représente un caractère du jeu de caractères du corpus. Il s'agira par la suite de mettre à 1 chaque croisement de caractères entre lignes et colonnes. Pour illustrer cet encodage nous prenons l'exemple suivant :

Soit le jeu de caractères "a", "b", "x", "d", "e", "l", "o", "h", "r", "w", " " et la phrase "hello world" que nous voulons encoder. La figure 3.4 montre l'encodage One Hot de cette phrase.

	"a"	"b"	"x"	"d"	"e"	"l"	"o"	"h"	"r"	"w"	" "
"h"	0	0	0	0	0	0	0	1	0	0	0
"e"	0	0	0	0	1	0	0	0	0	0	0
"l"	0	0	0	0	0	1	0	0	0	0	0
"l"	0	0	0	0	0	1	0	0	0	0	0
"o"	0	0	0	0	0	0	1	0	0	0	0
" "	0	0	0	0	0	0	0	0	0	0	1
"w"	0	0	0	0	0	0	0	0	0	1	0
"o"	0	0	0	0	0	0	1	0	0	0	0
"r"	0	0	0	0	0	0	0	0	1	0	0
"l"	0	0	0	0	0	1	0	0	0	0	0
"d"	0	0	0	1	0	0	0	0	0	0	0

FIGURE 3.4 – Exemple de l'encodage basé caractères appelé One Hot Encoding

L'algorithme 2 résume l'encodage basé caractères des transcriptions :

Algorithme 2 : Encodage basé caractères des transcriptions

```
Input  : repertoire_dataset
         char_to_int_characters : table de conversion des caractères vers des
entiers
Output : /* Pas de variables en Output                                    */
liste_transcriptions ← recuperer_transcriptions(repertoire_dataset)
pour transcritpion ∈ liste_transcriptions faire
    pour index, caractere ∈ transcription faire
        encoder_caractere_en_input()
        si index > 1 alors
            encoder_caractere_en_target()
        fin
    fin
    decoder_input.ajouter(caractere_encode_en_input)
    decoder_target.ajouter(caractere_encode_en_target)
fin
enregistrer_fichier(decoder_input)
enregistrer_fichier(decoder_target)
```

L'encodage basé caractères est simple à implémenter mais présente des résultats qui sont moins performants qu'un encodage basés mots. En effet, des erreurs dans les suites de caractères donneraient naissance à des mots qui n'existent pas. C'est pour cela qu'il faut y associer un modèle de langage pour corriger ces erreurs. L'encodage basé mots quant à lui, en plus de présenter des résultats plus cohérents que ceux de l'encodage basé caractères, fait office de modèle de langage en assurant que les suites de caractères sont correctes pour les mots prédits correctement.

Nous présentons dans ce qui suit les différentes approches pour l'encodage basé mots.

3.4.3.5 Encodage des transcriptions basé mots

Tout comme pour l'encodage basé caractères, nous attachons au début de nos transcriptions le mot spécial SOS qui signifie "Start Of Sentence" et à la fin le mot EOS qui signifie "End Of Sentence" pour marquer respectivement le début et la fin d'une transcription.

L'encodage basé mots va vraisemblablement présenter des résultats très différents de l'encodage basé caractères et ce car le modèle devra distinguer et classifier un très grand nombre de mots en comparaison avec une cinquantaine de caractères pour le premier encodage.

Pour encoder des mots, nous pensons naturellement à l'encodage *One Hot Encoding* également appelé *Bag Of Words* dans le cas des mots. C'est une approche similaire à celle présentée dans la section 3.4.3.4 et qui consiste à représenter une transcription par une matrice où les lignes représentent les mots, séparés par le caractère blanc, et les colonnes représentent tous les mots distincts de notre jeu de

données. Nous illustrons cet encodage par un exemple réel de notre jeu de données. Prenons la phrase "m$AhdynA AlkrAm AlslAm Elykm" qui est la translittération de " مشاهدينا الكرام السلام عليكم" dont l'encodage est le suivant :

	m$AhdynA	AlkrAm	AlslAm	Elykm
m$AhdynA	1	0	0	0
AlAntxAbAt	0	0	0	0
AlkrAm	0	1	0	0
ywm	0	0	0	0
lltdAwl	0	0	0	0
Elykm	0	0	0	1
.			.	
.			.	
.			.	
AlslAm	0	0	1	0
fy	0	0	0	0
hnAk	0	0	0	0

FIGURE 3.5 – Exemple de l'encodage basé mots Bag Of Words

Cet encodage paraît adéquat à première vue mais pour un jeu de données aussi large que le notre où il existe plus de 150000 mots distincts, il est quasi-impossible, en terme de mémoire vive, d'encoder chaque transcription x sous forme d'une matrice de taille (x, 150000) d'où l'impossibilité d'utiliser cet encodage pour notre dataset.

Il existe néanmoins un autre encodage qui est l'encodage binaire où nous convertissons chaque mot en vecteur binaire. Cet encodage s'effectue en suivant les étapes suivantes :

- Collecte de la liste des mots distincts existant dans notre jeu de données.
- Création d'une table associant un indice unique à chaque mot.
- Création d'un vecteur binaire avec une taille suffisante pour encoder chaque indice tel que :

$$2^x < nbr\_mots\_distincts \tag{3.2}$$

où :

x est le nombre de bits du vecteur. Nous estimons ne pas avoir besoin de plus de 18 bits pour encoder chaque mot de notre jeu de données.

- Convertir l'indice de chaque mot en binaire et l'introduire dans le vecteur binaire.

Si nous prenons l'exemple précédent avec la phrase : "m$AhdynA AlkrAm AlslAm Elykm", l'encodage de cette dernière serait le suivant :

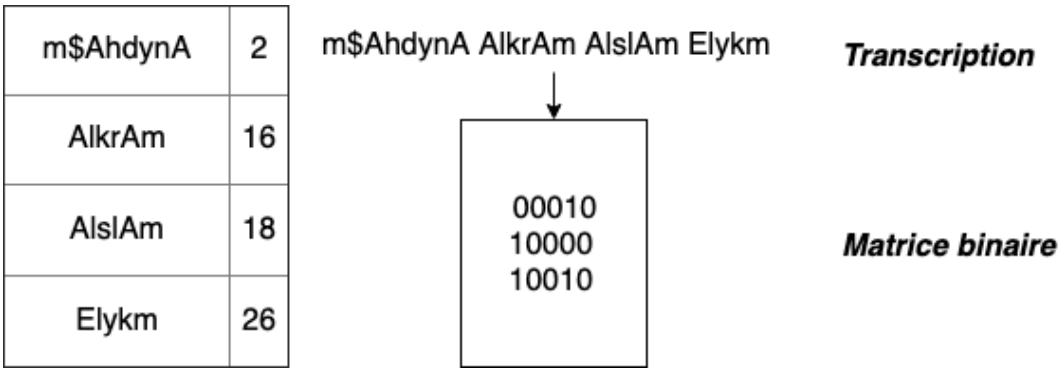

FIGURE 3.6 – Encodage binaire des mots du jeu de données

Bien que l'encodage binaire semble apporter la solution à notre problématique, il présente un désavantage majeur lors de l'apprentissage qui est le calcul d'une fausse précision. Si pour un vecteur binaire le modèle nous affiche, par exemple, une précision de 90%, notre modèle aura donc la faculté de prédire 9 bits sur 10 mais après reconversion du vecteur binaire prédit en mot nous nous rendrons compte que le mot deviendra un tout autre mot.

Afin de remédier à cette anomalie, nous présentons un nouvel encodage pour les mots. Le principe de cet encodage consiste à :

- calculer le nombre maximum de caractères par mot dans notre jeu de données,
- pour les transcriptions en entrée du modèle d'apprentissage, encoder chaque mot d'une transcription par un vecteur de taille « **Nombre de caractères du jeu de données** × **Nombre maximum de caractères par mot** » de telle sorte à encoder chaque caractère de chaque mot en utilisant un One Hot Encoding sur une partie de taille **Nombre de caractères du jeu de données** de ce vecteur, et
- pour les transcriptions en sortie du modèle d'apprentissage, encoder chaque mot de la transcription par **N** vecteurs où **N** représente le **nombre maximum de caractères par mot** et où chaque vecteur est de taille « **Nombre de caractères du jeu de données** ».

Afin de clarifier cet encodage qui peut paraître inhabituel au premier abord, nous présentons un exemple d'encodage avec le mot "hello", le jeu de caractères "a", "e", "h", "l", "b", "x", "o", " " et la taille maximale de mots de 7 comme le montre la figure 3.7.

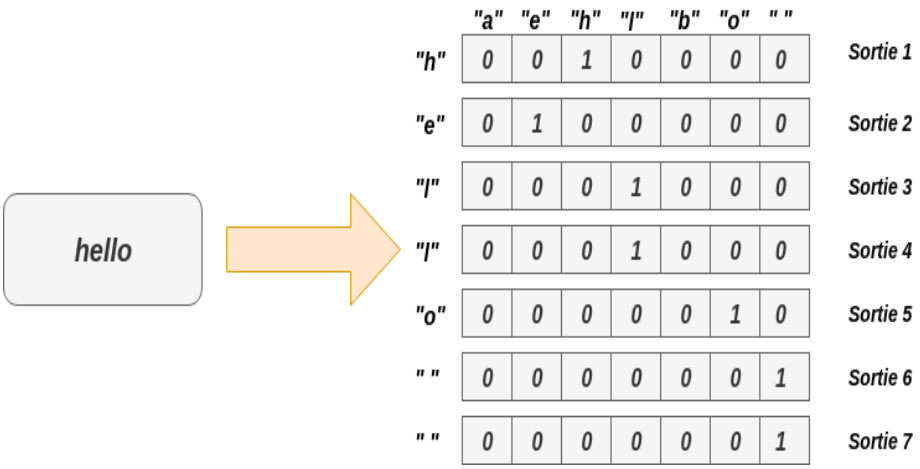

FIGURE 3.7 – Exemple Encodage basé mots à plusieurs sorties

L'algorithme 3 suivant résume cet encodage :

Algorithme 3 : Encodage basé mots des transcriptions

Input : repertoire_dataset
char_to_int_characters : table de conversion des caractères vers des entiers
max_car_mot : maximum de caracteres par mot de tout le corpus
nb_car : nombre de caracteres disctincts du corpus

Output : `/* Pas de variables en Output */`

```
liste_transcriptions ← recuperer_transcriptions(repertoire_dataset)
pour transcritpion ∈ liste_transcriptions faire
    liste_mots ← diviser_transcription_en_mots(transcription)
    pour mot ∈ liste_mots faire
        longueur_liste ← nb_car × max_car_mot
        encoded_word ← liste_de_zeros(longueur_liste)
        encoded_word_target ← matrice_de_zeros(nbr_car, max_car_mot)
        pour i, index_caractere ∈ mot faire
            encoder_caractere_en_input()
            si i<max_caracteres_par_mot alors
                encoder_fin_mot_input() si i > 1 alors
                    encoder_caractere_en_target()
                fin
            fin
        fin
    fin
    encoded_input.ajuoter(encoded_transcriptions_input)
    encoded_target.ajouter(encoded_transcriptions_target)
fin
enregistrer_fichier(decoder_input)
enregistrer_fichier(decoder_target)
```

En adoptant un tel encodage, nous évitons les difficultés liées à la gestion de la mémoire ainsi que l'erreur entre la précision et le WER pour l'encodage binaire.

Nous présentons dans le chapitre suivant l'implémentation d'un tel encodage ainsi que les performances achevées par ces deux encodages.

3.4.3.6 Teacher Forcing

Afin de mettre en place l'architecture que nous présentons dans la suite de ce chapitre, nous aurons, en plus des transcriptions encodées, une copie encodée de ces transcriptions à l'instant t+1. Ce principe s'appelle Teacher Forcing [Lamb et al., 2016], également appelé Professor Forcing. Cela permet à notre système de déduire quel caractère/mot avoir en sortie sachant l'enregistrement audio et la séquence de caractères/mots précédents.

3.4.3.7 Traitement des grandes tailles de données

Comme mentionné précédemment, nous aurons à effectuer notre apprentissage sur des jeux de données de tailles différentes afin d'évaluer les performances des différents modèles. Nous serons donc amenés à effectuer des apprentissages sur six heures, 60 heures, 260 heures et 1200 heures d'enregistrements audio en fin de parcours.

Gérer une telle quantité de données nous met face à deux problèmes principaux :

- impossibilité de charger le dataset nettoyé en mémoire pour l'encodage des transcriptions au vue de sa taille, et
- envoi de toutes les données à notre modèle pour l'apprentissage sachant que dix heures de dialogue équivaut à environ 500 Mo en mémoire.

Afin de remédier à ces complications, nous appliquons le principe de pagination. Nous commençons par la génération des fichiers binaires qui contiennent les spectrogrammes ainsi que leurs transcriptions. Au lieu de générer un fichier contenant toutes les données à notre disposition, nous générons un ensemble de fichiers en limitant le nombre d'heures de dialogue contenues dans chaque fichier. Nous prenons, pour la suite de ce projet, des fichiers de dix heures de dialogue chacun.

L'étape suivante consiste à créer à partir de ces partitions, des sous-partitions de telle sorte à avoir des fichiers binaires pour l'apprentissage et le test. Nous définissons ce nombre de sous-partitions de manière dynamique de telle sorte à avoir une utilisation minimum de la mémoire vive de l'utilisateur. Nous calculons le nombre de partitions à partir de la formule suivante :

$$nbr\_partitions = \frac{32}{memoire\_utilisateur} * duree\_dataset\_heures \tag{3.3}$$

Nous rencontrons une difficulté supplémentaire lors de l'apprentissage car nous devons envoyer toutes nos données à notre modèle pour qu'il puisse effectuer la mise à jour des poids sur l'ensemble de données mais là nous ne pouvons avoir toutes nos données en mémoire. C'est ici que nous faisons appel au principe de générateur de données qui se définit par les étapes suivantes :

- création d'une fonction chargeant en mémoire, à tour de rôle, les couples de fichiers contenant les spectrogrammes et transcriptions,
- création d'une table de hachage contenant comme clé les couples de nombre de timesteps des spectrogrammes et transcriptions, et comme valeur une liste des spectrogrammes et transcriptions ayant ces tailles, et
- envoi successif des listes de chaque clé à la fonction d'apprentissage.

Nous avons donc implémenté manuellement l'envoi des données à la fonction d'apprentissage pour permettre une gestion optimale de la mémoire en premier lieu et d'accélérer l'apprentissage en second lieu.

Maintenant que notre corpus est extrait et pré-traité nous passons à la conception des différentes architectures de modèles pour notre système.

3.5 Apprentissage du système

Nous nous attelons dans cette partie à la conception du modèle de notre système de reconnaissance de la parole. Comme mentionné précédemment, nous développons un système End-To-End qui, contrairement à l'approche basée reconnaissance de phonèmes, consistera en un module prenant en entrée un enregistrement audio et fournissant en sortie la transcription qui lui correspond. Dans le cas de la reconnaissance de la parole, nous avons affaire à des données de type séries temporelles où la taille de chaque enregistrement est variable. Comme tout exercice d'apprentissage, il existe plusieurs modèles et plusieurs architectures pour appréhender le problème. Pour les séries temporelles, nous nous intéressons aux modèle de type Séquence à Séquence.

3.5.1 Séquence à Séquence et Encodeur/Décodeur

Le modèle Séquence à Séquence, traduit de l'anglais Sequence to Sequence, fut introduit en 2014 par Google [Sutskever et al., 2014]. Ce modèle a pour but de mapper une entrée avec une sortie et où la longueur des entrées et celle des sorties peuvent être différentes.

Par exemple, traduire "What are you doing today ?" de l'anglais vers le français comporte une entrée de 5 mots et une sortie de 4 mots "Que faites-vous aujourd'hui ?". Comme les tailles des entrées et sorties sont différentes, ce qui est le cas pour les problèmes à données sous forme de séquences, nous ne pouvons utiliser un architecture séquentielle classique pour mapper chaque mot de la phrase en anglais à la phrase en français. Le modèle séquence à séquence est utilisé pour traiter des problèmes comme celui-ci.

Le modèle séquence à séquence se base sur le principe sous-jacent d'encodeur/-décodeur [Cho et al., 2014b] que nous représentons par l'illustration suivante :

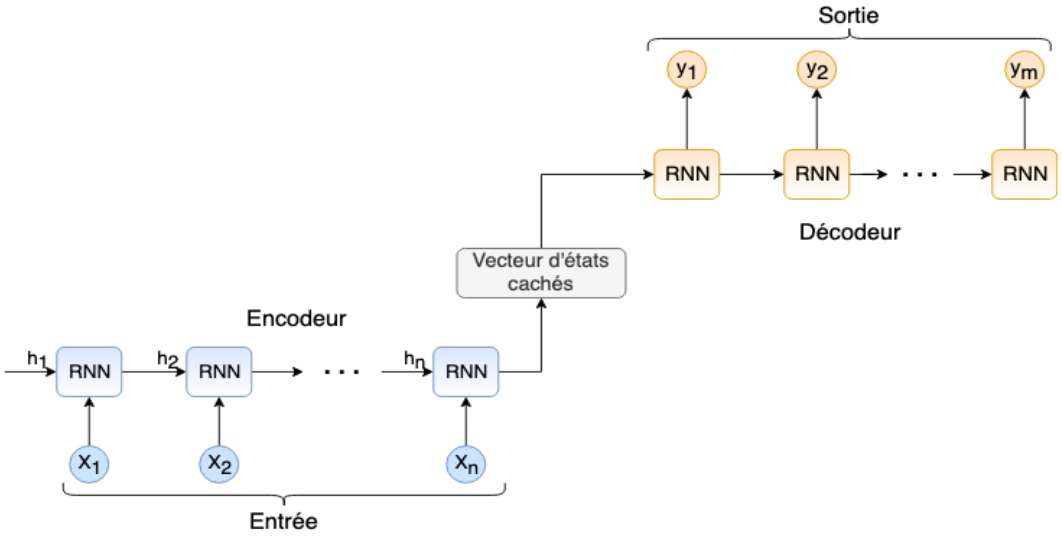

FIGURE 3.8 – Architecture de base d'un encodeur/décodeur

Comme illustré dans la figure 3.8, le modèle comprend deux blocs principaux qui sont l'encodeur et le décodeur ainsi qu'un vecteur intermédiaire qui est le vecteur d'états cachés.

3.5.1.1 Encodeur

L'encodeur est le premier bloc de ce modèle et est composé d'une pile de plusieurs unités récurrentes. Chacune de ces unités ne prend qu'un timestep de la séquence d'entrée et collecte les informations relatives à cet élément que nous appelons état caché (hidden state en anglais) et qui se calcule à partir de la formule suivante :

$$h_t = f(W^{(hh)} h_{t-1} + W^{(hx)} x_t) \tag{3.4}$$

où :
h : est l'état caché à timestep t.
x : est la donnée à un pas de temps t.
$W^{(hh)}$: représente les poids relatifs l'état caché.
$W^{(hx)}$: représente les poids relatifs à la sortie de l'encodeur.

L'encodeur a pour but de donner une représentation de la séquence de données afin de "l'encapsuler" dans le vecteur d'états cachés (Hidden States Vector en anglais), et va donc contenir l'état caché final produit par l'encodeur du modèle qui fera office d'état initial pour le décodeur.

Dans le cadre de notre projet, l'encodeur est la partie du modèle qui prend les enregistrements audio en entrée et apprend à regrouper les séquences d'un spectrogramme et produire une représentation vectorielle de celle-ci.

3.5.1.2 Décodeur

Tout comme l'encodeur, le décodeur se compose d'un ensemble d'unités récurrentes [Kostadinov, 2019] où :

- chaque unité prédit une sortie Y_t pour un pas de temps **t**, et
- chaque unité prend en entrée la donnée et un vecteur d'état caché de l'unité précédente (ou de l'encodeur si c'est la première unité) et génère à partir de cela un output et son propre vecteur d'états cachés pour l'unité suivante.

Le calcul de l'état caché se fait à partir de la formule suivante :

$$h_t = f(W^{hh} h_{t-1}) \tag{3.5}$$

où :
h : est l'état caché à un pas de temps t.
$W^{(hh)}$: représente les poids du réseau.

et l'output Y_t de l'unité récurrente pour un pas de temps **t** est calculé à partir de l'état caché selon la formule suivante :

$$Y_t = Softmax(W^s h_t) \tag{3.6}$$

où :
Y : est l'output de l'encodeur à un pas de temps t.
h : est l'état caché à un pas de temps t.
W : représente les poids du réseau.
Softmax : est une fonction d'activation qui attribue des probabilités décimales à chaque classe d'un problème à plusieurs classes. La somme de ces probabilités décimales doit être égale à 1 [Dunne and Campbell, 1997].

3.5.2 Discussion

L'intérêt du modèle encodeur/décodeur réside dans le fait de pouvoir associer des séquences de différentes longueurs en entrée et en sortie. Nous utilisons ce modèle pour le développement du système où :

- les entrées de l'encodeur sont nos enregistrements audio,
- les entrées du décodeur sont les transcriptions, et
- les sorties du décodeur sont les transcriptions à *t+1* selon le principe de Teacher Forcing que nous détaillons dans le chapitre suivant.

Nous passons à présent à la conception de l'architecture interne des unités récurrentes de notre modèle.

3.6 Architecture du modèle

Il existe plusieurs architectures possibles pour le développement de notre modèle, comme nous l'avons noté dans l'état de l'art. Chaque travail trouvé dans l'état de l'art présente une conception, un corpus et bien évidement des résultats différents, ce qui peut prêter à confusion.

Afin de mettre en place l'architecture idéale pour notre modèle, nous en expérimentons deux et pour chaque architecture nous prenons en compte deux variantes. Nous comparons ensuite l'évolution des courbes d'apprentissage de chaque architecture. Afin que notre système soit améliorable, il doit pouvoir prendre en compte l'augmentation (ou même la réduction) du volume de données. Ainsi, pour que cette comparaison soit précise, nous utilisons quatre jeux de données : six heures de dialogue, 60 heures, 260 heures et 1200 heures de dialogue.

Avant de présenter nos architectures nous commençons par présenter les couches que nous utilisons.

3.6.1 Présentation des couches utilisées

3.6.1.1 Long Short-Term Memory

Long-Short-Term-Memory, ou LSTM, sont un type particulier de RNNs et ont été introduits par [Hochreiter and Schmidhuber, 1997]. Les LSTMs ont été conçus pour éviter le problème de dépendance à long terme. En effet, il s'est avéré que les RNNs ne peuvent apprendre le séquencement d'informations à long terme. Comprendre et se souvenir du séquencement des informations quelque soit la taille de la séquence est l'avantage des LSTMs par rapport aux RNNs, et ce grâce au mécanisme des portes présentes dans une cellule LSTM. Ces portes peuvent apprendre les données importantes à conserver parmi les données d'entrée. En faisant cela, un LSTM peut transmettre des informations pertinentes tout au long de la séquence de données pour effectuer des prédictions.

La figure 3.9 présente le contenu d'une cellule LSTM ainsi que les opérations internes qu'y s'effectuent :

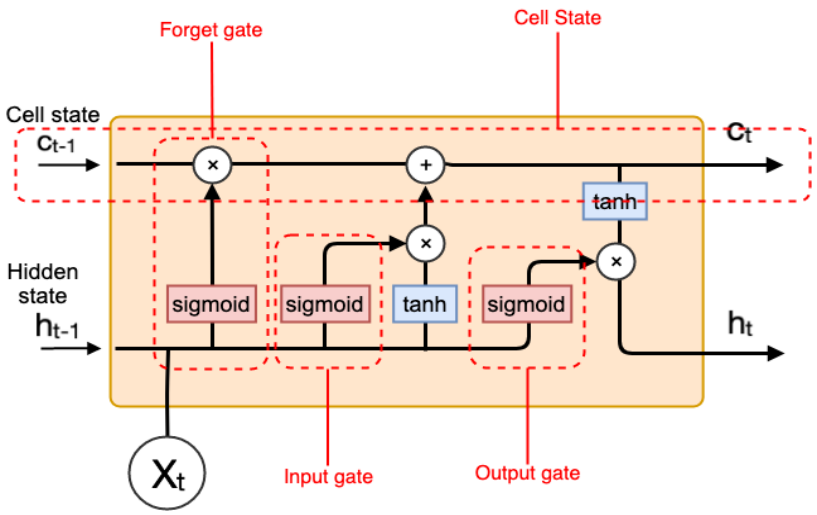

FIGURE 3.9 – Contenu d'une cellule LSTM [LST, 2018]

Comme le montre la figure 3.9, en plus du Hidden State qu'on retrouve dans une cellule RNN, il y a le Cell State qui est le concept de base des LSTMs. Le Cell State permet de garder en mémoire les informations pertinentes. Ces informations sont donc sauvegardés grâce aux trois portes d'une cellule LSTM [Nguyen, 2018a] qui sont :

- **Forget Gate** : pouvant être traduite en porte d'oubli, elle décide quelles informations doivent être conservées. Pour se faire, les informations contenus dans le vecteur d'états cachés précédent et les données en entrée sont additionnées et passent par la fonction sigmoïde et la valeur résultante est comprise entre 0 et 1. Plus la valeur se rapproche de 1 plus l'information est pertinente.

- **Input Gate** : L'input Gate, ou porte d'entrée, permet de mettre à jour le Cell State. Les valeurs du vecteur d'états cachés précédent et celles des donnée en entrée sont additionnées et passent par une fonction sigmoïde ce qui, comme pour la Forget Gate, permet de désigner les données pertinentes. En plus de la fonction sigmoïde, ces mêmes informations passent par une fonction tanh pour rendre les valeurs entre -1 et 1 afin de contribuer à la régulation du réseau. Les sorties de la fonction sigmoïde ainsi que tanh sont multipliées et ce résultat est additionné à la valeur contenue dans le Cell State.

- **Output Gate** : l'Output Gate, ou porte de sortie, décide quel devrait être le prochain vecteur d'états cachés. Premièrement, nous passons l'état caché précédent et l'entrée actuelle dans une fonction sigmoïde. Ensuite, nous passons le Cell State mis à jour par la fonction tanh. Nous multiplions la sortie tanh par la sortie sigmoïde pour décider quelles informations l'état caché doit contenir. C'est ainsi que sont générés les nouveaux Cell State et Hidden State qui seront utilisés pour le timestep suivant.

3.6.1.2 Gated Reccurent Unit

Introduit en 2014 par [Cho et al., 2014a], les Gated Reccurent Units (GRU) sont des unités récurrentes assez similaires à des LSTMs dans leur fonctionnement. Les principales différences sont que les GRUs n'ont pas de Cell State pour transferer l'information et n'ont que deux portes [Nguyen, 2018a] :

- **Update Gate** : agit de manière similaire aux Forget Gate et Input Gate d'un LSTM. Cette porte décide quelles informations doivent être gardées et quelles nouvelles informations ajouter ; et
- **Reset Gate** : utilisée pour décider de la quantité d'informations à oublier.

La figure 3.10 présente le contenu d'une unité GRU ainsi que les opérations internes qu'y s'effectuent.

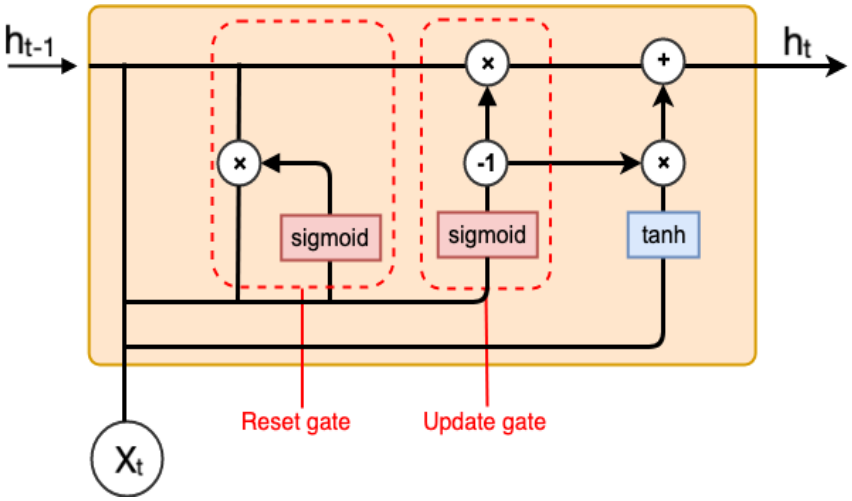

FIGURE 3.10 – Contenu d'une cellule GRU

Les GRUs effectuent moins de calculs matriciels que les LSTMs et sont donc un peu plus rapides lors de l'apprentissage.

3.6.1.3 Couches récurrentes bidirectionnelles

Les couches récurrentes bidirectionnelles sont une extension des RNNs traditionnels qui peuvent améliorer les performances du modèle en cas de problème de classification de séquences [Graves and Schmidhuber, 2005]. Dans les problèmes où tous les timesteps de la séquence d'entrée sont disponibles, comme le cas de notre problème, les RNNs bidirectionnels entraînent deux RNNs au lieu d'un seul. Le premier de ces RNN accomplit un apprentissage sur la séquence d'entrée de la manière usuelle tandis que le second le fait sur une copie triée dans le sens inverse de la séquence d'entrée. Cela peut fournir un contexte supplémentaire au réseau et permettre un apprentissage plus plus complet. Pour notre problématique, ce type de couches joue le rôle d'un modèle de langage.

3.6.2 Architectures de l'approche End-To-End pour ASeR-Sytem

Nous présentons dans cette partie les deux architectures où chaque architecture est basée sur un modèle encodeur/décodeur avec des RNNs mais aussi une variante à base de RNNs bidirectionnels pour chacune des deux architectures.

3.6.2.1 Architecture de base

Le modèle de base se compose de :

- **Encodeur** : composé d'une couche récurrente et renvoie les vecteur Hidden State.
- **Décodeur** : composé de plusieurs couches récurrentes.
- **Couche Fully Connected** : Dernière couche du modèle d'un réseau de neurones MLP complètement connecté qui prend en entrée l'output du décodeur.

La figure 3.11 résume l'architecture de ce modèle.

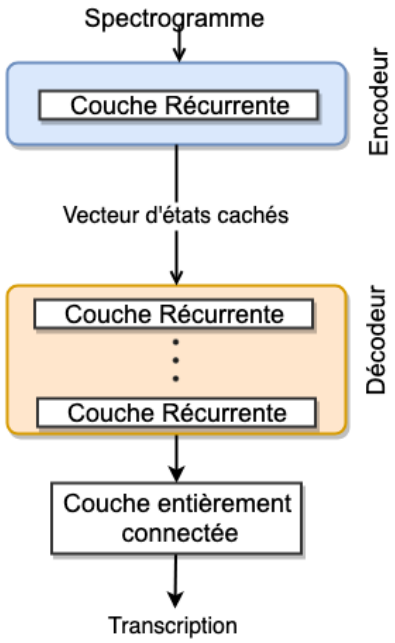

FIGURE 3.11 – Architecture de base de ASeR-System

3.6.2.2 Architecture avec couches convolutionnelles

Le modèle à couches convolutionnelles se compose de :

- **Couches convolutionnelles** : Se compose d'un ensemble de convolutionnelles et couches Pooling. Cette partie du modèle a pour but d'extraire les informations pertinentes du spectrogramme et de supprimer le bruit qui aurait échapper aux filtres du MFCC. La dernière couche génère des spectogrammes modifiés qui seront injectés en entrée de l'encodeur.

- **Encodeur** : composé d'une couche récurrente et renvoie le vecteur Hidden State.
- **Décodeur** : composé de plusieurs couches récurrentes. En plus de cela, le décodeur comporte une couche Dropout qui a pour but de minimiser les risques de sur-apprentissage en désactivant aléatoirement certains neurones des couches précédentes.
- **Couche Fully Connected** : Dernière couche du modèle, prend en entrée l'output du décodeur.

La figure 3.12 résume l'architecture du modèle.

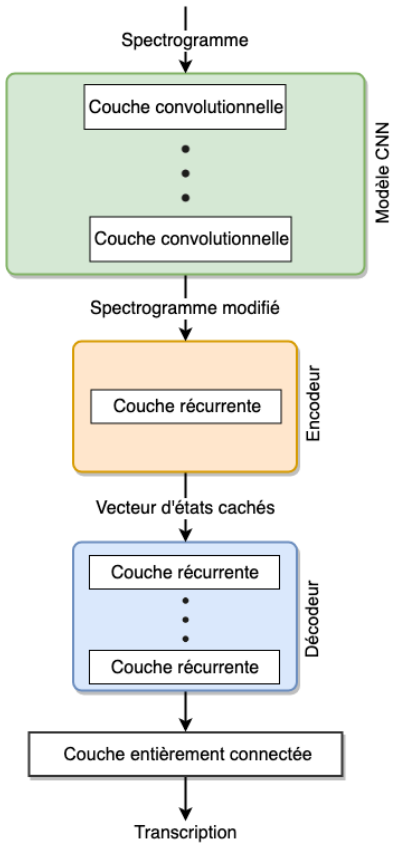

FIGURE 3.12 – Architecture de ASeR-System avec couches convolutionnelles

3.7 Utilisation du modèle d'apprentissage pour la reconnaissance

Après l'apprentissage de notre modèle, nous devrons tester celui-ci sur des données de test dans un premier temps ensuite des données réelles ou, autrement dit, tester nos propres enregistrements audio. Contrairement à un modèle d'apprentissage classique, le modèle séquence à séquence ne fait pas une simple prédiction de classes qui, dans notre cas, nécessiterait l'enregistrement audio et sa transcription

en entrée du modèle. C'est pour cela que nous effectuons une tâche d'inférence [Graves, 2013a]. Nous ne pouvons parler directement de prédiction car notre modèle se compose d'un encodeur et d'un décodeur. Le processus d'inférence se décompose en deux parties : la préparation du modèle après l'apprentissage et le décodage des transcriptions. La préparation du modèle se fait donc comme suit :

- Chargement des couches d'entrée et états cachés de l'encodeur à partir du modèle après apprentissage.
- Chargement des couches d'entrées, couches récurrentes ainsi que couches de sortie du décodeur.
- Génération d'un modèle pour l'encodeur et un modèle pour le décodeur et ce à partir des couches chargés préalablement.

Maintenant que l'encodeur et le décodeur sont prêts, nous passons au décodage des transcriptions. Il est à noter que cette partie de l'inférence est sensiblement la même pour un encodage basé caractères ou encore un encodage basé mots. Les étapes sont les suivantes :

- Préparation de la table qui associe cette fois un entier à chaque caractère ou mot selon l'encodage.
- Prédiction du vecteur d'états cachés à l'aide de l'encodeur. Ce vecteur est la représentation de l'enregistrement audio en entrée.
- Prédiction du caractère ou du mot correspondant en donnant en entrée du décodeur le vecteur d'états ainsi que le caractère ou mot actuel, la sortie est le caractère ou mot prédit ainsi qu'un vecteur d'états qui sera en entrée de la prochaine prédiction. Si la prédiction effectuée est celle du premier caractère, le vecteur d'états est le vecteur d'états cachés généré par l'encodeur.
- Répéter l'étape précédente tant que le caractère ou mot généré n'est pas fin de transcription.

3.8 Application de ASeR-System à un système de questions-réponses

ASeR-System peut être utilisé dans plusieurs domaines d'application du traitement automatique du langage naturel (TALN). Dans le cadre de ce projet, nous considérons les systèmes de questions-réponses comme domaine d'application.

L'équipe du LRIA, laboratoire de recherche en intelligence artificielle (Laboratory for Research in Artificial Inteligence) de l'université des sciences et de la technologie Houari Boumédiène a développé un système de questions-réponses nommé *AFaQ-System* qui a été mis à notre disposition. Comme nous l'avons mentionné dans les sections 1.4.2 et 1.4.3, un système de questions-réponses peut être classifié selon le domaine et/ou le type de questions qu'il accepte, le système de questions-réponses *AFaQ-System* est de type factoïde, il répond aux questions qui commencent par :

متى، كم ، أين، من. Un corpus [Aouichat and Guessoum, 2017] a été exclusivement développé dans le but d'être utilisé pour l'apprentissage des modèles des différents modules de ce système qui se compose de trois modules à savoir :

- le module de pré-traitement d'une question (Question preprocessing),
- le module de pré-traitement des documents (document preprocessing), et
- le module d'extraction de réponses (answer extraction).

La figure 3.13 illustre la conception de l'application développée dans le cadre de ce projet. Cette application permet d'utiliser *ASeR-System* avec *AFaQ-System*.

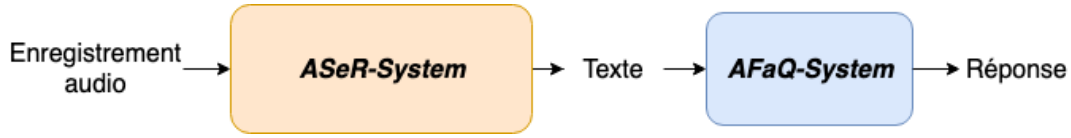

FIGURE 3.13 – Système de reconnaissance de la parole intégré avec un système de questions-réponses

3.9 Conclusion

Dans de ce chapitre, nous avons présenté les données à notre disposition ainsi que le pré-traitement mis en place afin d'en tirer profit pour l'apprentissage de nos modèles. Ensuite, nous avons présenté au total quatre architectures basées sur le principe d'encodeur/décodeur mais qui se composent de couches différentes. Le but principal de ce choix de conception est de garantir l'implémentation d'un modèle fiable et performant. Pour finir, nous avons présenté *AFaQ-System*, le système de questions-réponses que nous utilisons pour tester *ASeR-System*.

Dans le chapitre suivant, nous passons à l'implémentation de nos modèle ainsi que le pré-traitement des données au vu des différentes phases d'apprentissage. Une fois le choix du modèle effectué sur la base de l'évolution des performance de chaque architecture, nous passerons à l'apprentissage du modèle final et au test de notre système à travers un système de questions-réponses.

CHAPITRE 4

RÉALISATION DE ASER-SYSTEM

4.1 Introduction

Nous avons présenté précédemment les concepts de base d'un système de reconnaissance de la parole, les travaux qui ont présenté les résultats les plus performants ainsi que la conception de *ASeR-System*, notre système End-To-End de reconnaissance de la parole pour la langue arabe. En plus d'effectuer la tâche de reconnaissance, *ASeR-System* est un environnement de développement permettant d'intégrer le système de reconnaissance, d'améliorer sa performance ou encore de développer un nouveau système, pour une autre langue par exemple.

Dans ce chapitre, nous commençons par présenter notre environnement de travail, les outils utilisés pour le développement et l'apprentissage, l'envoi des données, les résultats d'apprentissage ainsi que les performances des différents modèles, le teste de notre système de reconnaissance de la parole sur un système de questions-réponses et l'organisation de l'environnement de développement.

4.2 Environnement et outils de travail

4.2.1 Matériel

La réalisation de ce projet s'est effectuée sur deux machines locales ainsi qu'un serveur cloud pour l'apprentissage des modèles. Nous avons opté pour un compte cloud Google Colaboratory [1]. Nous nous sommes tournés vers ce serveur cloud car il offre la puissance de calcul nécessaire en terme de GPU et plus particulièrement en terme de processeurs CUDA qui sont des unités de calcul massivement parallèles pour l'apprentissage profond. La configuration des deux machines locales ainsi que la machine virtuelle est présentée dans le tableau 4.1.

1. Google Colaboratory est un environnement de type "Jupyter Notebook" qui ne nécessite pas de configuration système et qui s'exécute entièrement sur le cloud [Google, 2017].

Paramétres	Machine 1	Machine 2	Google Colaboratory
Système d'exploitation	GNU/Linux Ubuntu	macOS Mojave 10.14.5	Jupyter notebook-based system
RAM	8GO	16GO	13GO
Processeur	Intel Core i7-8550 CPU	2,9 GHz Intel Core i5	2vCPU @ 2.2GHz
GPU	-	-	1xTesla K80 @ 3.7 GHz 2496 CUDA cores 12GB GDDR5 VRAM

TABLE 4.1: Caractéristiques des machines utilisées

4.2.2 Langage de programmation et logiciels

Afin d'implémenter notre système, il était nécessaire de choisir le langage de programmation ainsi que les différentes librairies que nous utilisons pour le développement et le test de notre système. Nous donnons dans cette section, une brève présentation de ces outils ainsi que leur utilité dans le cadre de ce projet.

4.2.2.1 Langage Python

Python est un langage de programmation interprété, interactif et orienté objet. Il intègre des modules, des exceptions, du typage dynamique, des types de données dynamiques de très haut niveau et des classes. Python combine une puissance remarquable avec une syntaxe très claire [van Rossum, 1989].

La syntaxe élégante et minimaliste, le grand nombre de bibliothèques scientifiques disponible en Open source ainsi que la grande communauté qui travaille dessus font de Python un langage idéal pour les scripts et le développement rapide d'applications dans de nombreux domaines et particulièrement le domaine de l'apprentissage automatique.

4.2.2.2 Bibliothèques utilisées

1. **Pydub :** bibliothèque libre et Open source implémentée en python conçue pour manipuler un audio avec une interface simple et facile de haut niveau [Robert, 2019]. Nous utilisons Pydub pour le découpage des enregistrements audio selon des temps de début et de fin de transcription.

2. **Librosa :** disponible en libre accès pour l'analyse musicale ainsi que l'analyse audio. il fournit les éléments de base nécessaires à la création de systèmes de récupération d'informations audio [McFee and Balke, 2019]. Nous utilisons

librosa pour générer les spectoragmmes MFCC à partir des enregistrements audio.

3. **NumPy (Numerical Python) :** est un package fondamental de calcul scientifique implémenté en Python et disponible en Open source. Ce package est essentiellement utilisé pour la manipulation des matrices ou tableaux multidimensionnels ainsi que des fonctions mathématiques opérant sur ces derniers. Au-delà de ses utilisations scientifiques évidentes, NumPy peut également être utilisé comme un conteneur multidimensionnel efficace de données génériques [Haldane and Terrel, 1995]. Nous utilisons Numpy pour encapsuler les données pour qu'elles soient utilisées lors de l'apprentissage.

4. **psutil (process and system utilities) :** est une bibliothèque multiplateforme permettant de récupérer des informations sur les processus en cours et l'utilisation du système en Python. Il est principalement utile pour la surveillance du système, l'établissement de profils et la limitation des ressources de processus et la gestion des processus en cours [Rodola, 2019]. Nous utilisons psutil pour lire la taille de la mémoire vive de l'utilisateur pour le partitionnement dynamique des datasets.

5. **matplotlib :** bibliothèque de traçage Python 2D qui produit des images de qualité dans une variété de formats papier et d'environnements interactifs sur toutes les plateformes [Silvester et al., 2003]. Nous utilisons Matplotlib pour dresser les différentes courbes de performance lors des apprentissages.

6. **TensorFlow :** plate-forme open source pour l'apprentissage automatique. Elle offre un écosystème complet et flexible d'outils, de bibliothèques et de ressources communautaires qui permet aux chercheurs de se familiariser avec les technologies de pointe et aux développeurs de créer et de déployer facilement des applications utilisant l'apprentissage automatique.

7. **Keras :** API de réseaux de neurones de haut niveau écrite en Python et qui enveloppe les fonctionnalités de la bibliothèque Tensorflow. Keras permet d'avoir un prototypage simple et rapide (convivialité, modularité et extensibilité), il prend en charge les réseaux convolutionnels et les réseaux récurrents, ainsi que les combinaisons des deux [Chollet et al., 2015a].

8. **Lang-trans :** bibliothèque Python de translittération principalement à partir de scripts non latins tels que l'arabe, le japonais, etc. [Aries, 2018].

9. **PyAudio :** bibliothèque python permettant d'effectuer différentes manipulation sur l'audio [Pham, 2017]. Nous utilisons PyAudio afin d'enregistrer un signal audio à partir du microphone de la machine.

10. **PyQT :** bibliothèque python permettant de développer des interfaces graphiques [Riverbank, 2018]. Nous l'utilisons pour développer notre application de test.

4.2.2.3 Formats de données

1. **Pickle :** module qui implémente des protocoles binaires pour sérialiser et désérialiser une structure d'objet Python. «Pickling» est le processus par lequel une hiérarchie d'objets Python est convertie en un flux d'octets et «unpickling» est l'opération inverse, par laquelle un flux d'octets (à partir d'un fichier binaire ou d'un objet de type octet) est reconverti en une hiérarchie d'objets [Python, 2019] .

2. **XML (eXtensible Markup Language) :** est une extension de format de fichier qui est similaire au HTML. Un fichier XML contient des symboles de balisage décrivant le contenu d'une page ou d'un fichier. XML est considéré comme extensible car les symboles de balisage sont illimités et se définissent d'eux-mêmes [Bray et al., 2006].

3. **wav :** Un fichier WAV est un fichier audio qui utilise un format audio numérique standard pour stocker des données. Il permet de sauvegarder les enregistrements audio avec différentes fréquences d'échantillonnage et différents débits binaires. Il est souvent sauvegardé dans un format stéréo à 44,1 KHz, 16 bits [Microsoft, 1991].

4.2.2.4 Logiciels

1. **PyCharm :** environnement de développement intégré utilisé pour programmer en Python. Il fournit une assistance et analyse de codage, une navigation fluide dans le code, la prise en charge de framework web, etc. [JetBrains, 2013].

2. **Sublime Text :** est un éditeur de texte générique codé en C++ et Python. Il est disponible sur les trois systèmes d'exploitation à savoir Windows, Mac OS et Linux. L'éditeur prend en charge 44 langages de programmation majeurs [Skinner, 2008].

3. **Git :** est un système de contrôle de versions distribué gratuit et Open source, conçu pour gérer tout projet en équipe, du plus petit au plus grand, avec rapidité et efficacité [Torvalds, 2005].

4.3 Préparation des données pour l'apprentissage

La préparation des données est une étape délicate qui affecte grandement les résultats de l'apprentissage selon la qualité du pré-traitement des données. Cela est d'autant plus vrai lorsque nous traitons des données de type séries temporelles et des corpus de très grande taille [Ding et al., 2008] comme c'est le cas pour le développement d'un système de reconnaissance de la parole.

Nous présentons dans cette section comment nous avons effectué les tâches suivantes :

- extraction des transcriptions et partitionnement des longs enregistrements audio en enregistrements plus courts pour le corpus élargi,
- conversion des enregistrements audio et transcriptions en fichiers binaires de dix heures de dialogue chacun, et
- encodage basé caractères et basé mots des transcriptions.

4.3.1 Traitement du corpus élargi

Comme mentionné dans le chapitre de conception, le corpus élargi est un corpus proposé en libre accès par l'équipe QCRI [QCRI, 2018]. Ce corpus contient un total de 2214 enregistrements audio d'une durée qui varie entre 20 minutes et un peu plus d'une heure chacun. Ces derniers sont des enregistrements d'émissions télévisées de la chaîne Al Jazeera et contiennent plusieurs centaines de phrases et donc plusieurs centaines de transcriptions pour chaque fichier. Nous retrouvons, associé à ces fichiers audio, leurs transcriptions sous forme de fichiers XML contenant les temps de début et de fin de chaque transcription, l'identité de l'orateur et la liste de mots d'une transcription. La figure 4.1 est un exemple de transcription :

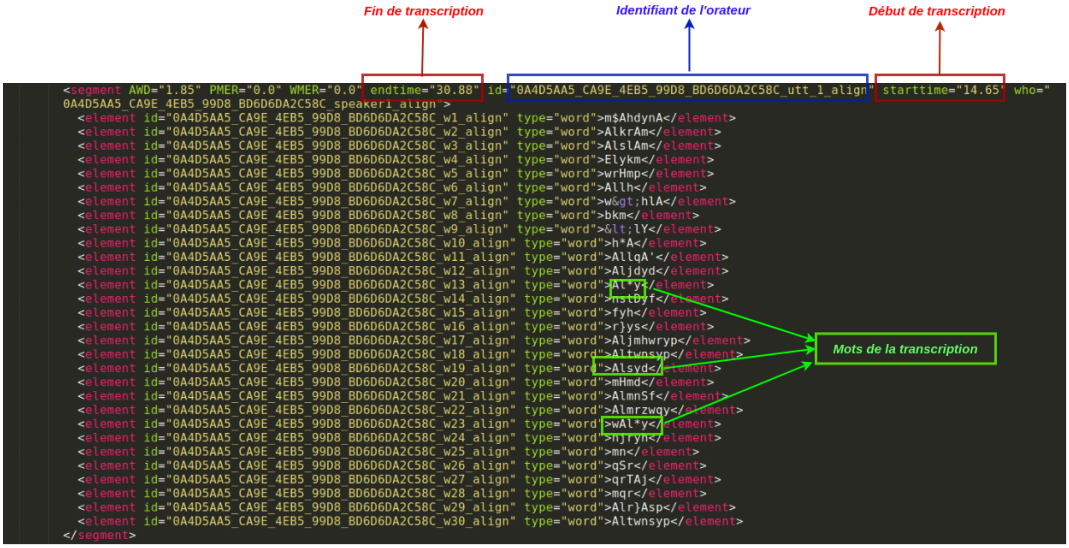

FIGURE 4.1 – Exemple de transcription dans un fichier XML du corpus élargi

Nous implémentons donc l'algorithme que nous avons présenté dans la section 3.4.2 en utilisant les méthodes de la bibliothèque Python "*XML*" ainsi que les méthodes de la bibliothèque "*Pydub*" pour effectuer le découpage des enregistrements audio selon les temps de début et fin d'enregistrement.

4.3.2 Conversion des fichiers audio et fichiers texte en dataset de dix heures

Afin de rendre l'exploitation des données plus simple, mais surtout plus rapide, nous créons une classe que nous nommons *"AudioInput"*. Cette classe aura pour

but de contenir le spectrogramme de l'enregistrement, sa transcription, son chemin ainsi que sa durée. Ensuite, nous parcourons chaque couple de répertoires audio et fichier de transcriptions et appliquons les modifications liées aux caractères qui n'appartiennent pas à la table Buckwalter [Buckwalter, 1990] que nous avons citées dans la section 3.4.3.3. Pour finir, nous définissons une limite de durée pour chaque dataset et générons des fichiers binaire de type *Pickle* contenant une liste d'objets *AudioInput* totalisant dix heures de dialogue.

Cette limite du nombre d'heures de dialogue pour chaque fichier Picke a pour but :

- de simplifier l'upload des fichiers Pickle sur le cloud,
- d'éviter les dépassements de taille de mémoire vive lors de la génération de ces fichiers car nous libérons l'espace mémoire après avoir généré les dix heures,
- de simplifier la séparation des datasets en données d'apprentissage et de test, et
- de faciliter la manipulation des datasets pour les 60 heures, 260 heures et 1200 heures d'apprentissage.

La génération de ce corpus est une opération qui a pris à la machine un total de quatre jours. Ce temps est due à la quantité de données à notre disposition, qui est de 118 GO, ainsi qu'à la durée de génération des spectrogrammes MFCC à l'aide de la bibliothèque *Librosa*. Il est à noter que cette opération n'aurait pas pu se faire sans partitionnement des données à cause de la mémoire vive qui aurait été saturée.

4.3.3 Encodage des transcriptions

Les réseaux de neurones ne traitent que les données numériques d'où la nécessité d'encoder les transcriptions sous forme de données numériques. Pour cela, deux approches s'offrent à nous : un encodage basé caractères et un encodage basé mots. Nous implémentons les algorithmes que nous avons présentés dans les sections 3.4.3.4 et 3.4.3.5 en utilisant la bibliothèque "*NumPy*" pour les transcriptions.Après encodage, l'ensemble des transcriptions est sous forme d'une matrice "*NumPy*" à 3 dimensions. La première dimension qui est un paramètre fixe représente le nombre de transcriptions ; la deuxième dimension, variable quant-à elle, représente le nombre de mots par transcription ; et, enfin, la troisième dimension qui est fixe, représente la taille de chaque mot selon l'encodage choisi.

4.4 Implémentation des modèles

Nous présentons dans cette section l'implémentation du modèle encodeur/décodeur, les différentes couches utilisées, l'environnement que nous avons mis en place pour l'apprentissage des modèles ainsi que l'envoi des données à l'algorithme d'apprentissage.

4.4.1 Modèle Encodeur/Décodeur

La base de notre système de reconnaissance de la parole est le modèle encodeur/-décodeur. Pour l'implémenter, nous utilisons l'API fonctionnelle de *Keras* [Keras, 2015] à travers la classe *Model*. L'intérêt principal de cette classe est de créer des modèles pouvant avoir plusieurs inputs et plusieurs outputs. Dans le cas de notre système, nous avons l'entrée de l'encodeur ainsi que l'entrée du décodeur en input et la sortie, ou les sorties pour la reconnaissance basée mots, du décodeur en output du modèle.

4.4.2 Couches utilisées

Nous avons utilisé les couches nativement fournies par **Tensorflow** pour l'implémentation de nos modèles. Ces couches sont les suivantes :

- **Input** : couche permettant de définir une donnée comme étant une donnée de type input pour l'API fonctionnelle de Keras. Cette couche prend en entrée la dimension des données en entrée et est insérée avant de définir les couches de l'encodeur et du décodeur.

- **GRU** : nous nous sommes tournés vers l'utilisation des couches GRU. Celles-ci elles présentent des résultats très semblables aux couches LSTM mais sont plus rapides pour l'apprentissage et l'inférence. Ces couches sont utilisées pour l'encodeur et le décodeur afin de capturer et comprendre les données séquentielles. Une couche GRU prend en entrée un certain nombre de paramètres ; le paramètre le plus important est sans doute "*latent_dim*" qui permet de spécifier la dimension de la représentation de l'entrée audio du modèle par l'encodeur. Il n'y a pas de mesure ou formule pour calculer la valeur idéale de ce paramètre ; le choix de celui-ci est donc, à première vue, expérimental.

- **Bidirectional** : ce n'est pas une couche mais une enveloppe de couche qui prend en paramètre une couche récurrente et effectue l'apprentissage dans un sens des données puis dans le sens contraire.

- **Conv1D** : couche convolutionnelle à une dimension qui a pour paramètres "*filters*" qui représente le nombre de filtres en sortie et le paramètre "*kernel_size*" qui, lui, représente la taille du masque de convolution.

- **MaxPooling1D** : prend en paramètre "*pool_size*" qui est la taille du vecteur qui se déplace dans le spectrogramme et retourne le maximum des valeurs contenues dans ce dernier.

- **Dense** : couche de sortie du modèle et a pour paramètres "*units*" qui est la dimension de la donnée en sortie ainsi que "*activation*" qui est la fonction d'activation de ce modèle. Pour notre problématique, nous utilisons la fonction d'activation "*Softmax*". Cette fonction retourne des valeurs positives dans

l'intervalle [0-1] ce qui, pour l'encodage One-Hot-Encoding, est nécessaire pour classifier les différents caractères.

4.4.2.1 Fonction d'erreur

Nous utilisons la fonction *categorical_ crossentropy* pour le calcul de l'erreur lors de l'apprentissage. Cette fonction d'erreur est utilisée lorsqu'il y a plusieurs classes à prédire et que toutes les classes ont la valeur 0 à l'exception de la classe à prédire qui prend la valeur 1.

4.4.2.2 Fonction d'apprentissage

Une fonction d'apprentissage, ou optimizer en anglais, sert à minimiser la fonction d'erreur. Cette fonction est un hyper-paramètre qui peut être défini :

- selon la problématique et les couches utilisées dans le modèle. Par exemple nous utilisons, en général, la fonction d'apprentissage *RMSprop* lorsque nous utilisons des récurrentes [Chollet et al., 2015b], et
- selon les résultats de l'expérimentation.

Nous nous intéressons, pour commencer, à la fonction d'apprentissage *RMSprop*, de la bibliothèque Keras [Chollet et al., 2015b] car celle-ci est, en théorie, convenable à la nature de notre problème.

La seconde fonction que nous considérons est la fonction *Adagrad* de Keras. *Adagrad* a des taux d'apprentissage qui sont spécifiques à chaque paramètre du modèle et sont adaptés en fonction de la fréquence de mise à jour des poids des couches. Cette particularité fait de cette fonction un bon candidat pour l'apprentissage d'un système de reconnaissance de la parole où les poids sont très fréquemment mis à jour.

Dans la suite de ce chapitre, nous expérimentons ces deux fonctions d'apprentissage pour définir celle qui répond le mieux à nos besoins.

4.4.3 Envoi des données pour l'apprentissage

Afin qu'un modèle effectue son apprentissage, des données d'entraînement et de test sont nécessaires. Lorsque nous effectuons l'apprentissage sur des données de type séries temporelles, il n'est pas possible d'envoyer les données d'entrée et de sorties directement en paramètre à la méthode *fit* de la classe *Model* de Keras. Celle-ci doit connaître la dimension des entrées et, dans notre cas, les données n'ont pas le même nombre de timesteps.

Pour remédier à ce problème, nous utilisons la méthode *fit_ generator* de la classe *Model* de Keras qui prend comme paramètre d'entrée une méthode que nous implémentons pour envoyer les données ayant le même nombre de timesteps à la fois.

L'algorithme 4 résume le fonctionnement de cette méthode.

Algorithme 4 : Algorithme de sélection des données qui ont le même nombre de timesteps

Input : liste_audio : liste des fichiers Pickle contenant les enregistrement audio
liste_transcriptions : liste des fichiers Pickle contenant les transcriptions
nombre_données : nombre total de données

```
tant que Vrai faire
    pour i ∈ [0, nombre_donnees] faire
        /* Récuperer les pairs (clé-valeur) tel que la clé est un tuple
           (timestep_audio, timestep_transcription) et la valeur est
           une liste contenant les spectrogrammes et transcriptions de
           même timestep que la clé.                                    */
        données ← pair(liste_audio[i], liste_transcriptions[i])
        pour (cle, valeur) ∈ donnees faire
            sortie ← (cle, valeur)
            pour element ∈ sortie faire
                /* Envoyer les inputs et les outputs ayant les mêmes
                   timesteps                                            */
                encoder_x.ajouter(element[0][0])
                decoder_x.ajouter(element[0][1])
                decoder_y.ajouter(element[1])
            fin
        fin
    fin
    /* Envoyer : traduite de "yield" qui envoie des données sans
       interrompre l'exécution de la fonction                           */
    envoyer [encoder_x, decodeur_x], decoder_y
fin
```

Le second souci que nous avons rencontré est le grand volume de données envoyé en entrée du modèle ce qui donne des itérations d'une longue durée. Afin de remédier à cette complication, nous précisons la taille du batch [2] et calculons la probabilité d'envoyer un ensemble de données par rapport à un autre ensemble. Pour calculer

2. Batch : nombre de données à être envoyées pour chaque itération.

ces probabilités nous avons défini la méthode 5 :

Algorithme 5 : calcul de probabilité d'envoi d'un ensemble de données d'apprentissage

Input : batch_size : nombre de données à envoyer dans un batch
dataset : Fichiers Pickle de spectrogrammes et transcriptions
Output : sortie : nombre d'instances à envoyer dans un batch

```
timesteps ← recuperer_timesteps_de_meme_taille(dataset)
taille_valeurs ← somme(valeurs(timesteps)
probas ← initiliser_dictionnaire()
/* Récuperer le dictionnaire des probas qui contient les probabilités
   d'avoir envoyer la clé                                              */
pour cle ∈ timesteps faire
  | proba[cle] ← taille(timesteps)/taille_valeurs
fin
cles_timesteps ← cles(timesteps)
nbr_parititons ← taille_valeurs/ batch_size + 1
pour i ∈ (0, nbr_paritions) faire
fin
/* Choisir au hasard un couple de timesteps selon la probabilité
   d'envoi de celui-ci                                                 */
r ← random()
pour cle ∈ cles_timesteps faire
  | si r < proba[cle] alors
  |   | sortie boucle
  | fin
  | r ← r - proba[cle]
fin
/* Retourner toutes les données ayant les timesteps selon la taille
   du batch                                                            */
sortie ← timesteps[cle]
n_batch_size ← min(batch_size, taille(sortie))
sortie ← random(sortie, n_batch_size)
Retourner : sortie
```

En sélectionnant les données à envoyer dans un batch aléatoirement, nous permettons au modèle d'effectuer un meilleur apprentissage. L'intuition derrière ce raisonnement est que lorsque les batchs sont envoyés au modèle dans le même ordre, celui-ci met à jour ses poids à la fin de l'itération en se basant toujours sur le même batch qu'il reçoit à la fin. Ainsi, envoyer les batchs de manière aléatoire permet au modèle de mettre à jour ses poids à la fin de l'itération sans effectuer son apprentissage sur les mêmes successions de données, améliorants ainsi les performances au test.

Nous passons à présent à la présentation de l'environnement utilisé ainsi que les différentes architectures avec leurs couches et leurs paramètres pour chaque corpus

4.4.4 Environnement d'apprentissage

Comme nous l'avons mentionné au début de ce chapitre, nous utilisons une machine Google Colaboratory [Google, 2017] qui est basée sur le cloud. Nous utilisons principalement cette machine pour sa carte graphique ainsi que son nombre élevé de Cuda Cores [3]. Afin de profiter de la puissance de calcul de la carte graphique, nous utilisons la couche récurrente *CuDNNGRU* à la place de *GRU*. Les couches *CuDNN* proposées par Keras permettent d'accélérer l'apprentissage des modèles à l'aide des GPU.

Google Colaboratory ne nous offre que la puissance de calcul graphique ; l'apprentissage s'effectue de la même manière que sur une de nos machines. Afin de travailler à partir de Google Colaboratory, nous passons par les étapes suivantes :

- créer un Jupyter Notebook à l'aide d'un compte Google,
- cloner le code source du système de reconnaissance de la parole à partir du projet Github que nous avons créé,
- se connecter à une instance Google Drive pour en copier les jeux de données,
- lancer le script python d'apprentissage à partir du code source disponible sur Google Colaboratory.

La figure 4.2 est un exemple d'environnement d'apprentissage avec Google Colaboratory.

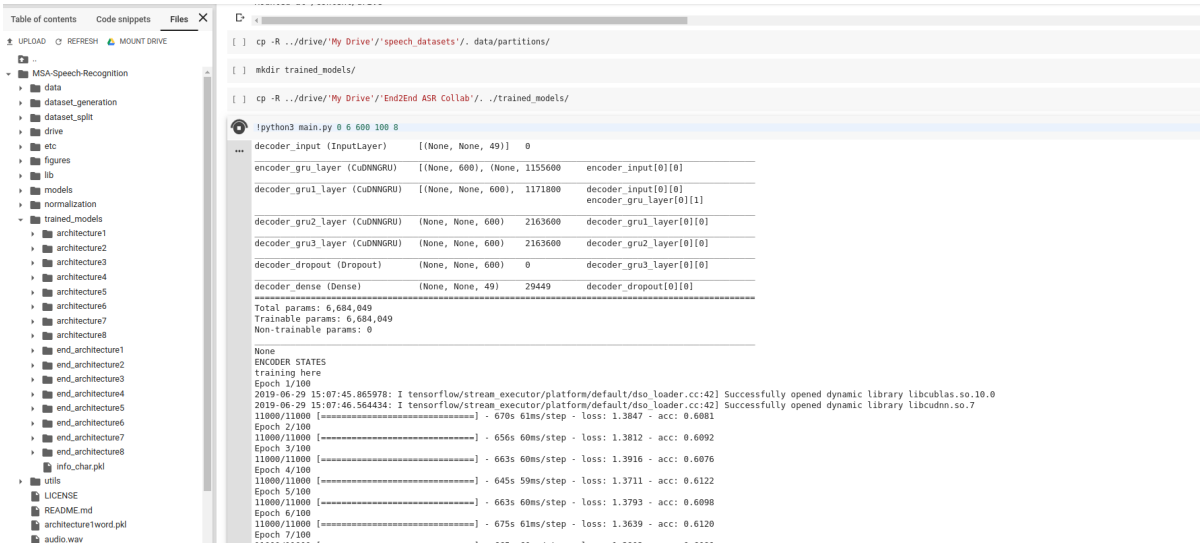

FIGURE 4.2 – Exemple d'un environnement d'apprentissage avec Google Colaboratory

Une difficulté que nous avons rencontrée est qu'une exécution à partir de Google Colaboratory ne dure que 12 heures et, une fois ce délai écoulé, toutes les données sur le disque sont effacées. Nous n'avions donc pas la possibilité d'enregistrer nos modèles pour poursuivre l'apprentissage. Afin de remédier à ce problème, nous sauvegardons

3. Cuda Core : Unité de calcul matricielle massivement parallèle dans les cartes graphiques

le modèle ainsi que l'historique des courbes d'apprentissage sur Google Drive et ce, de manière automatique en implémentant la méthode *on_ epoch_ end* dont l'appel s'effectue automatiquement à la fin de chaque itération.

4.5 Apprentissage et discussion des résultats

Nous passons à présent à l'apprentissage des modèles que nous avons définis dans la section 3.5 de ce document. Pour la préparation des modèles et leurs apprentissages nous utilisons les couches que nous avons définies précédemment en variant les paramètres de celles-ci.

Pour l'apprentissage basé mots, chaque couche de sortie a sa propre exactitude et sa propre erreur. Pour une meilleure lecture des résultats nous calculons les moyennes des exactitudes et des erreurs sur les six premières couches de sortie car *%70* des mots du corpus ont une taille égale ou inférieure à six caractères, ce qui est assez représentatif. Nous avons choisi de calculer cette moyenne sur six caractères car seulement *39%* des mots ont cinq caractères ou moins. En plus de cela, au delà de six caractères les exactitudes sont très élevées car elles représentent les exactitudes des caractères de fin de mots. Prendre en compte les exactitudes de ces couches de sortie risquerait de biaiser la performance rapportée du système.

Nous présentons dans cette section les architectures pour chacun des modèles avec les meilleures performances au test sous forme de tableaux récapitulatifs et courbes de performances.

4.5.1 Résultats d'apprentissage sur les différents corpus

Nous présentons et discutons dans ce qui suit les résultats obtenus par les différentes architectures en nous basant sur les trois corpus suivants :

- **Corpus d'essai** : ce corpus, que nous avons présenté dans la section 3.4.1, est un corpus de six heures qui a pour but de tester le fonctionnement des différentes architectures. Nous effectuons l'apprentissage de cinq heures de dialogue et le test sur une heure.
- **60 heures de dialogue** : pour ce corpus nous effectuons l'apprentissage sur 52 heures de dialogue et le test sur huit heures. Ce corpus contient 35111 mots distincts, une taille de mots maximale de 15 caractères et un jeu de 42 caractères distincts.
- **260 heures de dialogue** : ce jeu de données contient 85532 mots distincts, une taille maximale de 16 caractères par mot pour un total de 49 caractères distincts. Nous divisons ces 260 heures de dialogue en 230 heures d'apprentissage et 30 heures de test.

4.5.1.1 Apprentissage basé caractères

Le tableau 4.2 résume les résultats de l'apprentissage en utilisant les couches unidirectionnelles avec les deux optimizers *RMSprop* et *Adagrad*. Le tableau 4.3

quant à lui résume les résultats obtenus avec les couches bidirectionnelles avec ces mêmes optimizers.

Corpus	**CNN**	**Encodeur**	**Décodeur**	**Latent dim**	**optimizer**	**Exactitude (test)**	**CER**	**WER**
six heures	-	1 GRU	2 GRU	500	RMSprop	29%	71%	75%
					Adagrad	31%	69%	88%
six heures	2 Conv1D	1 GRU	2 GRU	400	RMSprop	27%	73%	87%
					Adagrad	32%	68%	87%
60 heures	-	1 GRU	2 GRU	500	RMSprop	42%	58%	72%
					Adagrad	42%	58%	72%
60 heures	2 Conv1D	1 GRU	2 GRU	400	RMSprop	40%	70%	73%
					Adagrad	43%	57%	68%
260 heures	-	1 GRU	3 GRU	550	RMSprop	51%	49%	54%
					Adagrad	68%	32%	41%
260 heures	3 Conv1D	1 GRU	3 GRU	500	RMSprop	49%	51%	53%
					Adagrad	71%	29%	38%

TABLE 4.2: Performances de l'apprentissage basé caractères avec couches unidirectionnelles

Corpus	**CNN**	**Encodeur**	**Décodeur**	**Latent dim**	**optimizer**	**Exactitude (test)**	**CER**	**WER**
six heures	-	1 GRU	2 GRU	500	RMSprop	26%	74%	89%
					Adagrad	26%	74%	91%
six heures	2 Conv1D	1 GRU	2 GRU	400	RMSprop	22%	78%	89%
					Adagrad	25%	75%	89%
60 heures	-	1 GRU	2 GRU	500	RMSprop	33%	67%	80%
					Adagrad	36%	64%	73%
60 heures	2 Conv1D	1 GRU	2 GRU	400	RMSprop	34%	77%	79%
					Adagrad	37%	73%	70%
260 heures	-	1 GRU	3 GRU	550	RMSprop	49%	51%	54%
					Adagrad	64%	36%	43%
260 heures	3 Conv1D	1 GRU	3 GRU	500	RMSprop	49%	51%	56%
					Adagrad	68%	32%	41%

TABLE 4.3: Performances de l'apprentissage basé caractères avec couches bidirectionnelles

Discussion des résultats : Nous remarquons tout de suite une grande différence entre les résultats obtenus pour chaque optimizer. La fonction *Adagrad* présente des

résultats nettement meilleurs que *RMSprop*. Nous nous intéressons donc aux résultats obtenus avec l'utilisation de *Adagrad*. Nous discutons ces résultats pour chaque corpus afin d'étudier l'évolution des performances des modèles avec l'augmentation du volume de données. Cette étude nous permettra de choisir, sur la base des résultats obtenus, le modèle adéquat pour l'apprentissage sur les 1200 heures de dialogue.

- **Corpus d'essai (six heures de dialogue)** : nous ne pouvons donner une interprétation digne d'intérêt aux résultats obtenus à partir du corpus d'essai. En effet, pour une problématique aussi complexe que la reconnaissance de la parole en utilisant un modèle End-To-End, cinq heures de dialogue ne sont pas représentatives provoquent clairement un sous apprentissage. Ce corpus a été utilisé uniquement dans le but de tester le bon fonctionnement de nos modèles. Nous passons à l'apprentissage sur 60 heures de dialogue pour obtenir des résultats plus représentatifs.

- **60 heures de dialogue :** la première chose que nous notons est une hausse des résultats d'apprentissage par rapport au corpus d'essai. Lors de cet apprentissage, l'exactitude de l'apprentissage pour chacun des modèles à atteint les 99%. Cependant comme le montre le tableau ci-dessus, les exactitudes de test ne dépassent pas les 43% ce qui est un signe de sur-apprentissage. Nous expliquons cela par le faible nombre de données d'apprentissage pour une problématique aussi complexe que la reconnaissance de la parole.

- **260 heures de dialogue :** avec 260 heures de dialogue, nous notons une nette amélioration des résultats d'apprentissage en comparaison avec les 60 heures. L'augmentation du volume de données a permis de minimiser le sur-apprentissage. En effet, les exactitudes d'apprentissage sont de 75% en moyenne. Nous remarquons également que le modèle à couches convolutionnelles a présenté les meilleurs résultats. Ceci est dû au traitement qu'effectuent les couches convolutionnelles sur les spectrogrammes avant que ceux-ci soient envoyés à l'encodeur. Notons cependant, que les modèles à couches bidirectionnelles ont présenté des résultats moin satisfaisants que les autres modèles. Ceci est dû à la quantité de données encore insuffisante ainsi qu'au temps que nous avons accordé aux apprentissages. Les modèles à couches bidirectionnelles prennent beaucoup plus de temps lors de l'apprentissage en comparaison aux autres modèles. Nous ne tentons pas d'aller plus loin dans l'apprentissage de ces modèles au vu de la limite de temps imparti pour la réalisation de ce projet.

4.5.1.2 Apprentissage basé mots

Nous adoptons donc pour cet apprentissage l'architecture à plusieurs couches de sortie où chaque couche n doit pouvoir reconnaître le *nième* caractère d'un mot. Ce nombre de couches est égal au nombre maximum de caractères par mot. Au vu de la supériorité de la fonction *Adagrad* par rapport à *RMSprop*, pour notre problématique du moins, nous effectuons nos apprentissages avec *Adagrad* uniquement.

Comme pour l'apprentissage basé caractères, le tableau 4.4 concerne les couches unidirectionnelles quant au tableau 4.5, il concerne les couches bidirectionnelles. Les

résultats d'apprentissage sont les suivants.

Corpus	CNN	Encodeur	Décodeur	Latent dim	Exactitude (test)	CER	WER
six heures	-	1 GRU	2 GRU	500	18%	78%	82%
	2 Conv1D	1 GRU	2 GRU	400	17%	79%	83%
60 heures	-	1 GRU	2 GRU	500	29%	66%	71%
	2 Conv1D	1 GRU	2 GRU	400	31%	67%	69%
260 heures	-	1 GRU	3 GRU	550	56%	39%	44%
	3 Conv1D	1 GRU	3 GRU	500	58%	36%	42%

TABLE 4.4: Performances de l'apprentissage basé mots avec couches unidirectionnelles

Corpus	CNN	Encodeur	Décodeur	Latent dim	Exactitude (test)	CER	WER
six heures	-	1 GRU	2 GRU	500	12%	83%	88%
	2 Conv1D	1 GRU	2 GRU	400	14%	83%	86%
60 heures	-	1 GRU	2 GRU	500	25%	64%	75%
	2 Conv1D	1 GRU	2 GRU	400	27%	71%	73%
260 heures	-	1 GRU	3 GRU	550	51%	44%	49%
	3 Conv1D	1 GRU	3 GRU	500	55%	39%	45%

TABLE 4.5: Performances de l'apprentissage basé mots avec couches bidirectionnelles

Discussion des résultats : Ici encore, nous discutons les résultats obtenus par les différents corpus à l'exception du corpus d'essai.

- **60 heures de dialogue** : nous remarquons une légère amélioration de la performance par rapport au corpus d'essai. Comme pour la reconnaissance basée caractères, l'exactitude d'apprentissage atteint les 99% pour chaque modèle. Nous nous retrouvons donc dans une situation de sur-apprentissage. Nous justifions ce résultat par le nombre insuffisant de données d'apprentissage. En effet, pour l'apprentissage basé mots, le modèle a n couches de sortie comme nous l'avons expliqué au début de cette section ce qui rend l'apprentissage sensiblement plus complexe. Nous ne pouvons espérer de bons résultats sans augmenter le volume de données.

- **260 heures de dialogue** : comme nous nous y attendions, nous notons une amélioration des performances qui est liée à l'augmentation du volume de données. Nous remarquons que le modèle à couches convolutionnelles a présenté

les meilleurs résultats d'apprentissage même si l'exactitude n'est que de 58% au test. Nous notons également un rapprochement entre l'exactitude d'apprentissage et de test minimisant ainsi le sur-apprentissage. Nous pensons que pour la reconnaissance basée mots, qui est de loin plus complexe que la reconnaissance basée caractères, un plus grand nombre de données est nécessaire. Nous étayons ce raisonnement par l'évolution de la performance lorsque nous sommes passé de 60 heures à 260 heures de dialogue.

Nous ne prenons pas la risque de lancer un apprentissage basé mots sur les 1200 heures de dialogue. Même si nous pensons que le modèle va enregistrer une performance nettement meilleure, nous ne pourrons en être sûrs sans expérimentation.

L'apprentissage sur le corpus élargi, après un bref test, prend entre 15 et 20 minutes par itération d'où l'impossibilité de lancer un apprentissage sans être sûr de la performance que peut apporter le modèle. Nous ne savons pas si cette quantité de données suffirait au modèle. Il nous semble donc plus adéquat de lancer un apprentissage basé caractères en introduisant un modèle de langage pour améliorer encore plus la performance du modèle.

4.5.2 Apprentissage sur le corpus élargi

4.5.2.1 Choix du modèle

Notre choix quant au modèle que nous utilisons pour le développement de *ASeR-System* s'est porté sur le modèle à couches convolutionnelles basé reconnaissance de caractères. Ce choix se justifie par les résultats prometteurs qu'a obtenu le modèle à couches convolutionnelles comparé aux autres modèles. Nous n'excluons pas la possibilité que les architectures à couches bidirectionnelles puissent présenter des résultats équivalents ou légèrement supérieurs aux résultats du modèle CNN mais nous ne pouvons déterminer à partir de quel volume de données ces modèles pourraient prendre le dessus sur le modèle CNN. Nous sommes également limité par la contrainte de temps car les modèles bidirectionnelles prennent plusieurs semaines d'apprentissage [Ahmed et al., 2016].

4.5.2.2 Apprentissage

Après le pré-traitement des données, nous nous sommes retrouvés avec 860 heures de dialogue que nous avons divisées en 780 heures pour l'apprentissage et 80 heures pour le test. Ce corpus se compose de 153500 mots distincts et 49 caractères distincts. Le tableau 4.6 détaille l'architecture que nous avons mise en place. Il est à noter que les résultats ne sont pas finaux et que l'apprentissage est toujours en cours.

Corpus	**CNN**	**Encodeur**	**Couches**	**Latent dim**	**Exactitude (test**	**CER**	**WER**
860 heures	4 Conv1D	1 GRU	4 GRU	500	74%	26%	17%

TABLE 4.6: Performance apprentissage modèle CNN sur le corpus élargi

Les résultats obtenu avec le corpus élargi sont prometteurs. Nous estimons qu'en ajoutant du temps à l'apprentissage du modèle, celui-ci pourrait achever une meilleure performance. Augmenter le volume des données est également un garant de l'amélioration de la performance de ce modèle et est tout l'intérêt de l'utilisation de l'approche End-To-End.

4.6 Environnement de développement pour la reconnaissance de la parole

Nous présentons dans cette section l'implémentation de l'environnement de développement pour la reconnaissance de la parole ainsi que l'ensemble des paramétrages que nous avons mis en place. Pour le développement de cet environnement, nous nous sommes inspirés de la convention PEP8 en matière d'organisation pour simplifier à tout utilisateur l'intégration de nos modèles ou encore l'apprentissage de nouveaux modèles. Le schéma 4.3 résume l'organisation de l'environnement :

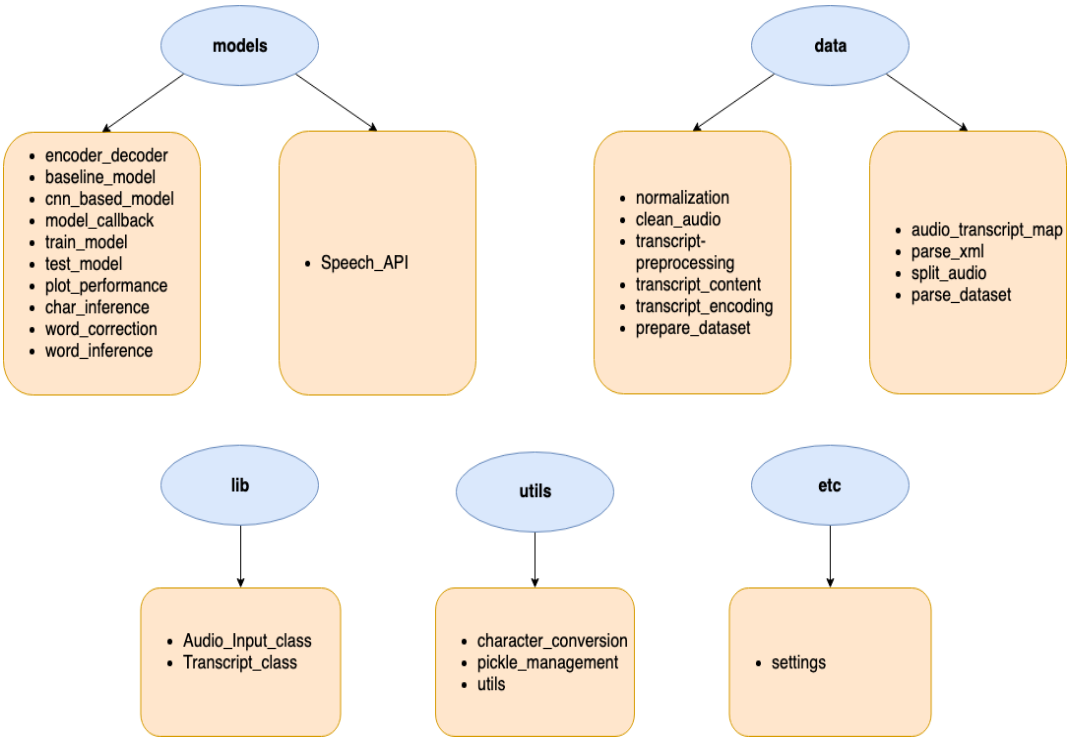

FIGURE 4.3 – Disposition des packages de l'environnement de reconnaissance de la parole

Nous avons implémenté l'environnement de développement de telle sorte à permettre à l'utilisateur d'utiliser chaque module indépendamment des autres modules. les modules sont les suivants :

- **models :** Ce module se divise en deux modules :

1. **Speech_API** : module contenant un système de reconnaissance de la parole End-To-End prêt à intégrer à n'importe quelle application d'un simple appel de fonction.
2. **Apprentissage :** Permettant d'effectuer l'apprentissage d'un modèle en second lieu. L'utilisateur a la liberté de modifier les couches utilisées pour l'apprentissage. Il est cependant à noter que tout changement dans les couches de l'architecture implique un changement au niveau de l'inférence.

- **data :** Ce module se décompose à son tour en deux modules :
  1. **Génération de dataset :** Module semi-automatisé permettant de générer à partir d'enregistrements audio et transcriptions en format texte des fichiers Pickle, selon un nombre d'heures que peut définir l'utilisateur s'il le souhaite, associant à chaque enregistrement audio un spectrogramme MFCC ainsi que sa transcription. Ce module est semi-automatisé car les jeux de données ne sont pas toujours idéalement présentés (comme ce fut le cas pour le QCRI Speech Dataset).
  2. **Pré-traitement des données :** Ce module permet un certain nombre d'opérations sur les données tel que la normalisation des spectrogrammes, la suppression des enregistrements de mauvaise taille (d'une durée trop courte ou trop longue) ainsi que plusieurs types d'encodages des transcriptions comme nous l'avons présenté dans la section 3.4.3.3.
- **lib :** Module contenant les différentes classes utilisées.
- **utils :** Contenant différentes méthodes pour la gestion des répertoires et fichiers, conversion des caractères de Buckwalter vers l'arabe et vice versa, conversion binaire et bien d'autres.
- **etc :** Contenant les chemins relatifs et variables globales utilisées pour le prétraitement et l'apprentissage.

Nous avons également mis en place une documentation détaillée concernant le fonctionnement de l'environnement de développement qui est disponible sur Github avec le système. Cette documentation est également disponible dans l'annexe de ce mémoire.

4.7 Intégration du Système de questions-réponses

Après avoir développé *ASeR-System*, il est temps de le tester sur un vrai domaine d'application. Comme nous l'avons mentionné dans la section 3.8, nous avons choisi le domaine des systèmes de questions-réponses et plus particulièrement *AFaQ-System*.

AFaQ-System est basé sur un ensemble des paramètres que nous avons la possibilité de modifier pour effectuer différents tests. Ces paramètres sont les suivants :

- –**Mode :** définit le mode de recherche. Deux possibilités s'offrent à nous : le mode "Online" pour la recherche de réponses via le Web et le mode "Offline" dans le cadre de l'utilisation d'un corpus local.

- –**Source :** une chaîne de caractères précisant le chemin vers le corpus.
- –**Seuil :** représente le seuil de sélection des phrases pertinentes.
- –**SentenceSelection :** définit le nom du modèle à utiliser dans la phase de sélection des phrases.
- –**Model_Path :** chaîne de caractères pour le chemin vers le modèle.
- –**NbrPassage :** le nombre des phrases à sélectionner.

Dans notre cas, nous gardons les valeurs par défaut pour tous les paramètres cités sauf pour le –**mode** que nous mettons à Online pour ne pas être restreints à un corpus local.

L'intégration *AFaQ-System* avec *ASeR-System* fut très simple, il nous a suffit d'appeler la fonction principale *ArabicQuestionAnsweringSystem* du système de quesions-réponses en lui donnant en entrée la question qui représente la transcription de notre audio pour avoir la réponse en format texte.

4.8 Application de ASeR-System sur un système de questions-réponses

Afin de tester le bon fonctionnement de notre système de reconnaissance de la parole, nous avons développé une application Desktop avec la bibliothèque *PyQT*. Cette application a pour objectif :

- d'effectuer un enregistrement sonore,
- d'utiliser la Speech_API de *ASeR-System* afin de générer la transcription de l'enregistrement sonore,
- de trouver une réponse à la demande de l'enregistrement sonore en utilisant *AFaQ-system*, et
- permettre à l'utilisateur de contribuer au projet en enregistrant un audio correspondant à une transcription afin d'augmenter la taille du corpus et ainsi, améliorer la performance de AeER-System.

Comme cette application est à but de test uniquement, nous avons développé une interface graphique simple et intuitive permettant de réaliser les tâches ci-dessus. La figure 4.4 présente la fenêtre principale de l'interface :

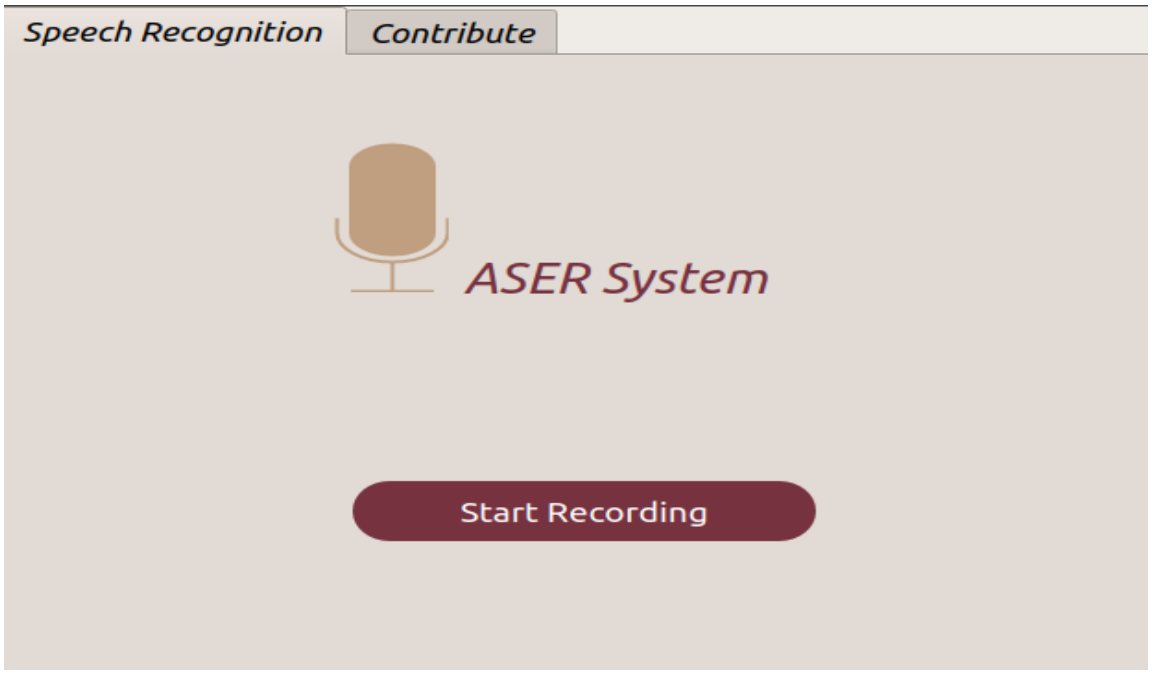

FIGURE 4.4 – Fenêtre principale de l'application

Ensuite, après avoir enregistré une question, *ASeR-System* effectue la reconnaissance du dialogue contenu dans l'enregistrement et génère une transcription.

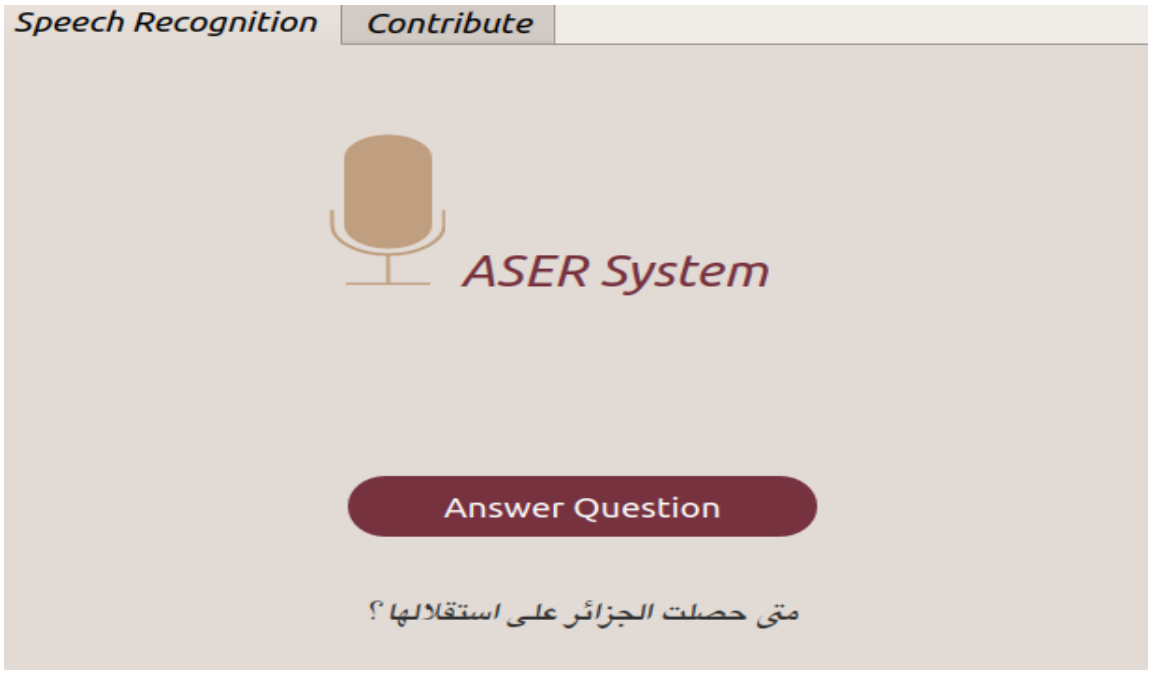

FIGURE 4.5 – Affichage de la transcription

Une fois la reconnaissance effectuée, l'utilisateur peut décider d'afficher une réponse à sa question et c'est là où nous faisons appel à *AFaQ-System* pour trouver

une réponse à cette question.

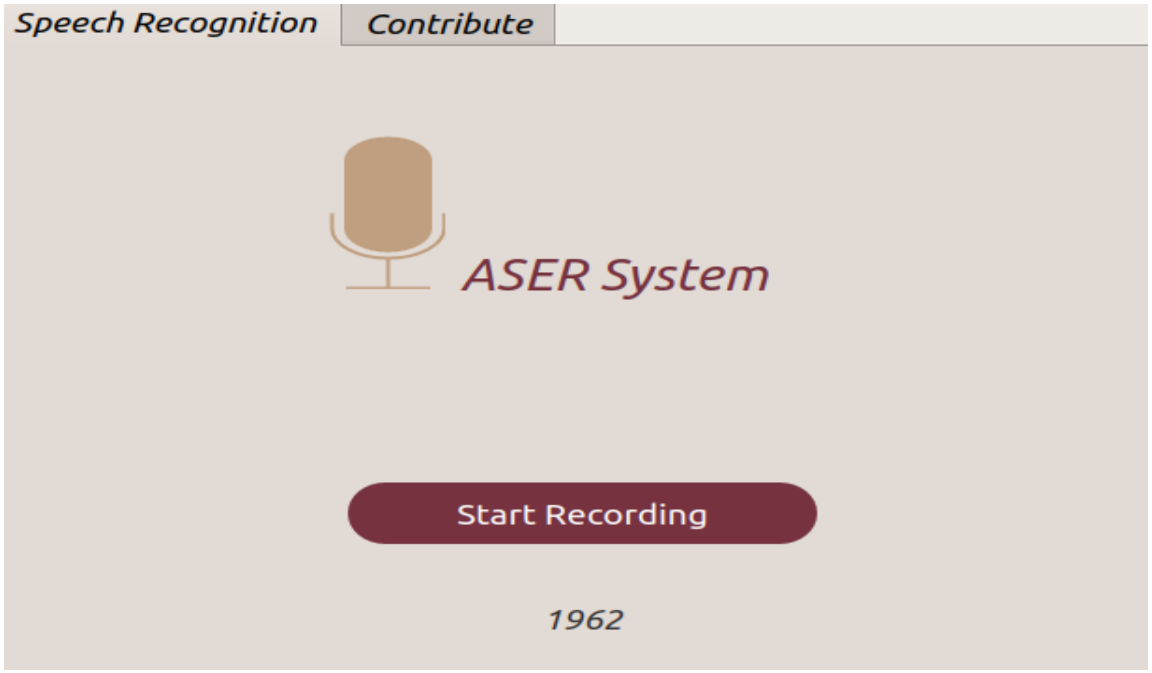

FIGURE 4.6 – Réponse à la question

Nous avons également introduit un onglet contribution. Cette partie de l'application nous aidera à améliorer la performance de *ASeR-System*. Nous générons aléatoirement des transcriptions à l'utilisateur pour que celui-ci enregistre un audio correspondant à ces transcriptions.

FIGURE 4.7 – Onglet contribution de l'application

Nous conservons ces enregistrements dans une base de données MySQL pour une utilisation future.

FIGURE 4.8 – Génération d'une transcription pour la contribution

Cette application est un exemple concret de l'utilisation du système de reconnaissance de la parole pour n'importe quel cas de figure tel qu'un système de questions-réponses.

4.9 Conclusion

À travers ce chapitre, nous avons présenté l'implémentation des différents modules de *ASeR-System* en passant par l'environnement de travail, les différentes librairies utilisées mais aussi la génération des données d'apprentissage, la normalisation et la préparation des différents modèles. Nous avons également discuté les résultats d'apprentissage obtenus par les différents modèles et particulièrement le modèle final que nous avons choisi pour notre système de reconnaissance de la parole.

Nous avons conclu avec la présentation de l'application que nous avons réalisée pour tester *ASeR-System* sur un système de questions-réponses. Cette application montre bien que notre environnement de développement permet de créer des systèmes de reconnaissance de la parole qui peuvent être utilisés dans tout type d'application de traitement automatique du langage naturel.

CONCLUSION GÉNÉRALE ET PERSPECTIVES

La motivation principale de ce travail a été de créer un système de reconnaissance de la parole pour la langue arabe en se basant sur plusieurs représentations des données et plusieurs architectures de modèles. Ce système a été par la suite utilisé dans une application avec un système de questions-réponses. Nous avons également créé un environnement de développement en offrant une base solide pour le pré-traitement des données et l'apprentissage des modèles permettant ainsi aux chercheurs de pousser la recherche dans le domaine.

La première étape de ce travail a été l'étude du fonctionnement général des systèmes de reconnaissance de la parole en mettant en avant les différentes approches pour développer de tels systèmes. Ensuite, nous avons introduit les systèmes de questions-réponses, leurs architectures et leur fonctionnement car ces systèmes sont un excellent cas d'application pour la reconnaissance de la parole, *i.e.*, sur lesquels une couche de reconnaissance de la parole peut être placée.

Une grande partie de notre travail fut l'étude de l'état de l'art en ce qui concerne les systèmes de reconnaissance de la parole. Cette étude a permis d'identifier les travaux les plus prometteurs en la matière et, plus particulièrement, les systèmes End-To-End qui sont encore récents et en pleine ascension. Nous avons également étudié et introduit les techniques avancées d'apprentissage profond qui sont utilisées dans ces systèmes.

Vint ensuite la conception de *ASeR-System*, notre système de reconnaissance de la parole pour la langue arabe avec les étapes de collecte et de nettoyage des données. Par la suite, nous avons approfondi le pré-traitement de ces données en proposant une représentation basée caractères et une représentation basée mots avec plusieurs encodages. Pour finir, nous avons conçu non pas une, mais quatre architectures de modèles que nous proposons dans notre environnement de développement.

Après l'étape de conception, ce fut le tour de l'implémentation de *ASeR-System* en passant par le pré-traitement des données, l'apprentissage des modèles sur plusieurs tailles de corpus, l'étude et la discussion des résultats obtenus. Nous l'avons testé sur un système de questions-réponses à travers une application développée à cet effet. Nous avons également implémenté les différents modules de notre environnement de développement de systèmes de reconnaissance de la parole.

Les principaux freins à ce projet furent le nombre limité, comparé à d'autres travaux, de données à notre disposition, le manque de moyens en terme de RAM et de GPU ainsi que le manque de temps pour l'apprentissage profond qui s'avère très complexe et nécessite des semaines d'apprentissage sur les données. Nous avons néanmoins mis en place tous les outils nécessaires en terme de modèles et de gestion dynamique de la mémoire pour permettre d'améliorer ou de recréer les modèles déjà existants en utilisant d'autres jeux de données plus élaborés.

Il est important de noter que notre travail s'inscrit dans le cadre d'un projet de recherche et reste donc perfectible. En effet, plusieurs points peuvent être améliorés pour avoir des systèmes de reconnaissance de la parole arabe aussi performants que possible. Les perspectives de ce projet sont, par ordre décroissant d'importance, les suivantes :

- apprentissage des modèles et particulièrement les modèles bidirectionnels pour une période de temps supplémentaire ;
- envoi de l'enregistrement audio sous forme de plusieurs découpes d'enregistrements pour afficher à l'utilisateur les résultats de la reconnaissance au fur et à mesure de l'enregistrement ;
- ajout du mécanisme d'attention ;
- développement et traitement d'un corpus de plusieurs milliers d'heures de dialogue pour parfaire l'apprentissage des modèles ;
- traitement des noms propres dans les corpus qui peuvent être écrits dans un alphabet qui n'est pas l'alphabet Buckwalter comme ce fut le cas avec le QCRI Corpus ;
- génération automatisée de corpus d'enregistrements audio et de transcriptions quelle que soit leur présentation comme fonctionnalité de l'environnement de développement ; et
- implémentation d'une fonction d'apprentissage PSO (particle Swarm Optimization) basée sur le principe d'intelligence en essaim. Le but de cet algorithme d'apprentissage est de trouver les minima locaux (voir globaux) de la fonction d'erreur de manière plus efficace lors de la rétro-propagation [Settles and Rylander, 2002].

Nous nous intéressons particulièrement à la perspective d'introduction du mécanisme d'attention. Nous sommes confiants quant à l'impact positif de celui-ci sur l'apprentissage des modèles. Nous avons poussé notre investigation de cette perspective et nous pouvons l'expliquer ci-dessous.

Le mécanisme d'attention a été introduit par [Bahdanau et al., 2014] et, récemment, les systèmes End-to-End basés sur l'attention ont été appliqués à une grande variété de tâches telles que la synthèse de l'écriture manuscrite [Graves, 2013b], la traduction automatique [Bahdanau et al., 2014], la génération de légendes d'images [Xu et al., 2015], la classification d'objets visuels [Mnih et al., 2014] ainsi que la reconnaissance de la parole.

Le but premier de l'apparition de ce mécanisme est de permettre au modèle encodeur/décodeur de mémoriser de plus grandes séquences de données ce qui peut s'avérer fort utile lors du processus de reconnaissance de la parole. À la manière d'un

LSTM ou GRU qui génère les vecteurs Hidden States, une couche d'attention génère un vecteur de contexte entre l'encodeur et le décodeur. Ce vecteur prend toutes les sorties de l'encodeur en entrée pour calculer la distribution de probabilités, ou poids d'attention, des timesteps de l'enregistrement audio aux entrées du décodeur qui sont des caractères (ou des mots si nous utilisons un encodage mot par mot). Il est ainsi possible pour le décodeur de capturer des informations globales sur l'alignement des données de l'encodeur avec celles du décodeur. Sans mécanisme d'attention, il est difficile d'effectuer cet alignement uniquement sur la base des vecteurs d'états générés par l'encodeur [SyncedReview, 2017].

La figure 4.9 illustre le fonctionnement du mécanisme d'attention pour la tâche de reconnaissance de la parole à partir d'un spectrogramme.

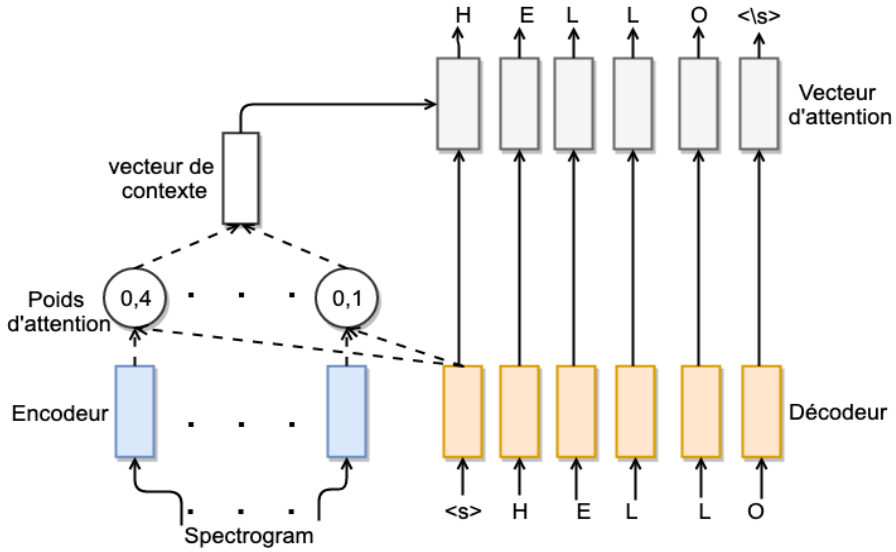

FIGURE 4.9 – Mécanisme d'attention

Nous pourrions introduire ce mécanisme en ajoutant une couche d'attention au modèle de base qui permettrait de relier les sorties de l'encodeur à celles du décodeur. Nous pensons que cette approche pourrait présenter un meilleur apprentissage. La figure 4.10 schématise l'architecture d'un tel modèle.

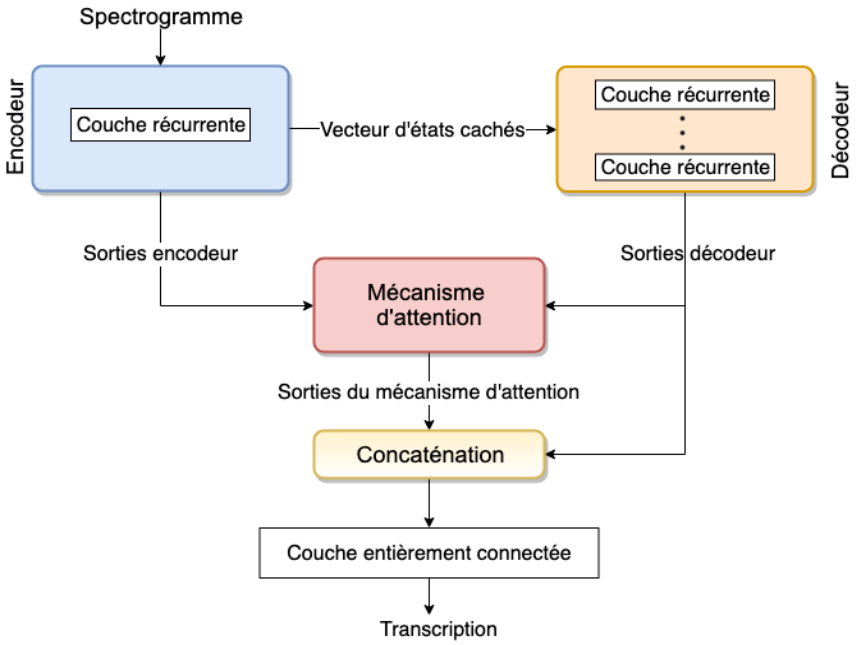

FIGURE 4.10 – Architecture d'un modèle End-To-End avec mécanisme d'attention

Pour clore, nous pouvons dire que la réalisation de ce projet nous a permis de maîtriser des domaines importants de l'intelligence artificielle. À travers ce travail nous avons :

- maîtrisé les principes de la reconnaissance automatique de la parole mais aussi les principes des systèmes de questions-réponses avec de potentielles applications très importantes que ce soit pour le web, applications mobiles ou autres plateformes, et
- approfondi nos connaissances en apprentissage automatique et maîtrisé de nouvelles approches plus flexibles et plus performantes nous permettant ainsi d'appréhender un grand éventail de problématiques d'intelligence artificielle.

En plus des compétences informatiques acquises, nous avons appris à gérer un projet, à rédiger un mémoire de qualité et à travailler en équipe tout en respectant les différentes conventions qui sont utilisées dans le monde professionnel. Au terme de ce projet, les enseignements s'avèrent être précieux et nous seront utiles tout au long de nos parcours respectifs.

BIBLIOGRAPHIE

[LST, 2018] (jully 6th 2018). Lstm and its equations. https://medium.com/@divyanshu132/lstm-and-its-equations-5ee9246d04af. Visited may 28th 2019.

[Abouenour, 2014] Abouenour, L. (2014). *Three-levels Approach for Arabic Question Answering Systems*. PhD thesis, UNIVERSITE MOHAMMED V DE RABAT, ECOLE MOHAMMADIA D'INGENIEURS.

[Ahmed et al., 2016] Ahmed, A., Hifny, Y., Shaalan, K., and Toral, S. (2016). Lexicon free arabic speech recognition recipe. In *International Conference on Advanced Intelligent Systems and Informatics*, pages 147–159. Springer.

[Ahmed et al., 2018] Ahmed, A., Hifny, Y., Shaalan, K., and Toral, S. (2018). *End-to-End Lexicon Free Arabic Speech Recognition Using Recurrent Neural Networks*, pages 231–248.

[Akbaraly, 2019] Akbaraly, M. (January 27th 2019). La synthèse vocale : un outil utile pour tous les utilisateurs. https://blog.ipedis.com/synthese-vocale-outil-experience-utilisateur, note =Visited February 18th 2018,.

[Al-Anzi and AbuZeina, 2018] Al-Anzi, F. and AbuZeina, D. (2018). Literature survey of arabic speech recognition. In *2018 International Conference on Computing Sciences and Engineering (ICCSE)*, pages 1–6. IEEE.

[Amazon, 2016] Amazon (November 30th, 2016). Amazon polly. https://aws.amazon.com/polly/. visited January 24th 2019.

[Amrouche et al., 2017] Amrouche, A., Falek, L., and Teffahi, H. (2017). Design and implementation of a diacritic arabic text-to-speech system. *International Arab Journal of Information Technology*, 4.

[Aouichat and Guessoum, 2017] Aouichat, A. and Guessoum, A. (2017). Building talaa-afaq, a corpus of arabic factoid question-answers for a question answering system. In *International Conference on Applications of Natural Language to Information Systems*, pages 380–386. Springer.

[Aries, 2018] Aries, A. (2018). trans-lang. https ://github.com/kariminf/lang-trans. Visited june 5th 2019.

[Arpabet, 2017] Arpabet (May 2017). Arpabet. https://ipfs.io/ipfs/QmXoypizjW3WknFiJnKLwHCnL72vedxjQkDDP1mXWo6uco/wiki/Arpabet.html.

[Azure, 2019] Azure, M. (2019). Microsoft speech api. https://azure.microsoft.com/en-us/services/cognitive-services/speech-to-text/. Visited January 23rd 2018.

[Bahdanau et al., 2014] Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv :1409.0473*.

[Banerjee, 2018] Banerjee, S. (May 23rd 2018). An introduction to recurrent neural networks. ://medium.com/explore-artificial-intelligence/an-introduction-to-recurrent-neural-networks-72c97bf0912.

[Belenko and Balakshin, 2017] Belenko, M. and Balakshin, P. (2017). Comparative analysis of speech recognition systems with open code. - , (04 (58) 4) :13–18.

[Bray et al., 2006] Bray, T., Textuality, Netscape, Paoli, J., and Sperberg-McQueen (2006). Xml. https ://www.w3.org/TR/2006/REC-xml11-20060816/sec-well-formed. Visited April 16th 2019.

[Buckwalter, 1990] Buckwalter, T. (1990). Arabic transliteration/encoding chart. http://languagelog.ldc.upenn.edu/myl/ldc/morph/buckwalter.html. Visited April 9th 2019.

[Capti, 2016] Capti (2016). Capti voice. https://www.captivoice.com/capti-site//. visited January 24th 2019.

[Cho et al., 2014a] Cho, K., Van Merriënboer, B., Bahdanau, D., and Bengio, Y. (2014a). On the properties of neural machine translation : Encoder-decoder approaches. *arXiv preprint arXiv :1409.1259*.

[Cho et al., 2014b] Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014b). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv :1406.1078*.

[Chollet et al., 2015a] Chollet, François, et al. (2015a). Keras. https://keras.io. Visited April 29th 2019.

[Chollet et al., 2015b] Chollet, François, et al. (2015b). Keras. https://keras.io/optimizers/. Visited May 15th 2019.

[Christensson, 2014] Christensson, P. (January 10th 2014). Speech recognition definition. https://techterms.com, note =Visited 2019, Jun 25,.

[Davis et al., 1952] Davis, Biddulph, and Balashek (1952). Audrey speech recognizer.

[Davis and Mermelstein, 1980] Davis, S. and Mermelstein, P. (1980). Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE transactions on acoustics, speech, and signal processing*, 28(4) :357–366.

[Deng et al., 2013] Deng, l., Hinton, G., and Kingsbury, B. (2013). New types of deep neural network learning for speech recognition and related applications : An overview. pages 8599–8603.

[Ding et al., 2008] Ding, H., Trajcevski, G., Scheuermann, P., Wang, X., and Keogh, E. (2008). Querying and mining of time series data : experimental comparison of representations and distance measures. *Proceedings of the VLDB Endowment*, 1(2) :1542–1552.

[Dunne and Campbell, 1997] Dunne, R. A. and Campbell, N. A. (1997). On the pairing of the softmax activation and cross-entropy penalty functions and the derivation of the softmax activation function. In *Proc. 8th Aust. Conf. on the Neural Networks, Melbourne*, volume 181, page 185. Citeseer.

[Fearn, 2018] Fearn, N. (2018). Best text to speech software. `https://www.techradar.com/news/best-text-to-speech-software`. [posted May 24th 2018].

[Ghahramani, 2001] Ghahramani, Z. (2001). An introduction to hidden markov models and bayesian networks. *IJPRAI*, 15 :9–42.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

[Google, 2017] Google (2017). Google colaboratory. `https://colab.research.google.com/notebooks/welcome.ipynb`. Visited April 29th 2019.

[Google, 2019a] Google (2019a). Cloud speech-to-text. `https://cloud.google.com/speech-to-text`. Visited January 24th 2018.

[Google, 2019b] Google (2019b). Google ai technique reduces speech recognition errors by 29%. `https://venturebeat.com/2019/02/21/google-ai-technique-reduces-speech-recognition-errors-by-29/`. Consulté le 24/02.2019.

[Graves, 2013a] Graves, A. (2013a). Generating sequences with recurrent neural networks. *arXiv preprint arXiv :1308.0850*.

[Graves, 2013b] Graves, A. (2013b). Generating sequences with recurrent neural networks. *arXiv preprint arXiv :1308.0850*.

[Graves and Jaitly, 2014] Graves, A. and Jaitly, N. (2014). Towards end-to-end speech recognition with recurrent neural networks. In *International conference on machine learning*, pages 1764–1772.

[Graves et al., 2013] Graves, A., Mohamed, A.-r., and Hinton, G. (2013). Speech recognition with deep recurrent neural networks. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, 38.

[Graves and Schmidhuber, 2005] Graves, A. and Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, 18(5-6) :602–610.

[Guyon and Yao, 1999] Guyon, X. and Yao, J. (1999). On the underfitting and overfitting sets of models chosen by order selection criteria. *Journal of Multivariate Analysis*, 70 :221–249.

[H. Juang and Rabiner, 2005] H. Juang, B. and Rabiner, L. (2005). Automatic speech recognition - a brief history of the technology development.

[Haldane and Terrel, 1995] Haldane, A. and Terrel, A. (1995). Numpy. `https://www.numpy.org/`. Visited March 21st 2019.

[Hamdi, 2012] Hamdi, A. (2012). Apport de la diacritisation de l'analyse morpho-syntaxique de l'arabe (apport of diacritization in arabic morpho-syntactic analysis)[in french]. In *Proceedings of the Joint Conference JEP-TALN-RECITAL 2012, volume 3 : RECITAL*, pages 247–254.

[Henrik et al., 2016] Henrik, B., Joseph W., R., and Mark, F. (September 2016). *Real-World Machine Learning*. Manning Editions.

[Hinton et al., 2012] Hinton, G., Deng, l., Yu, D., Dahl, G., Mohamed, A.-r., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T., and Kingsbury, B. (2012). Deep neural networks for acoustic modeling in speech recognition. *Signal Processing Magazine, IEEE, vol. 29, no. 6*.

[Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8) :1735–1780.

[IBM, 1962] IBM (1962). Ibmshoebox. `https://www.ibm.com/ibm/history/exhibits/specialprod1/specialprod1_7.html`. [Visited November-03-2018].

[J. Keogh and Pazzani, 2002] J. Keogh, E. and Pazzani, M. (2002). Derivative dynamic time warping. *First SIAM International Conference on Data Mining*, 1.

[JetBrains, 2013] JetBrains (2013). Pycharm. https ://www.jetbrains.com/pycharm/. Visited March 2nd 2019.

[Jurafsky and Martin, 2008] Jurafsky, D. and Martin, J. (2008). *Speech and Language Processing : An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, volume 2.

[Kaldi, 2011] Kaldi (2011). The kaldi speech recognition toolkit. `http://kaldi-asr.org/`. Visited January 24th 2018.

[Katz et al., 2006] Katz, B., Borchardt, G. C., and Felshin, S. (2006). Natural language annotations for question answering. In *FLAIRS Conference*, pages 303–306.

[Keras, 2015] Keras (2015). Keras documentation, functional api. https ://keras.io/getting-started/functional-api-guide/. Visited April 2nd 2019.

[Kostadinov, 2019] Kostadinov, S. (Feb 4th 2019). Understanding encoder-decoder sequence to sequence model. `https://towardsdatascience.com/understanding-encoder-decoder-sequence-to-sequence-model-679e04af4\346`. Visited April 13th 2019.

[Lamb et al., 2016] Lamb, A. M., Goyal, A. G. A. P., Zhang, Y., Zhang, S., Courville, A. C., and Bengio, Y. (2016). Professor forcing : A new algorithm for training recurrent networks. In *Advances In Neural Information Processing Systems*, pages 4601–4609.

[Lecun et al., 1998] Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86 :2278 – 2324.

[Linguatec, 2016] Linguatec (2016). Voice reader home. `https://www.linguatec.de/en/text-to-speech/voice-reader-home-15`. visited January 24th 2019.

[M. Katz, 1987] M. Katz, S. (1987). Estimation of probabilities from sparse data for the language model component of a speech recognizer. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 35 :400 – 401.

[Matarneh et al., 2017] Matarneh, R., Maksymova, S., Lyashenko, V., and Belova, N. (2017). Speech recognition systems : A comparative review.

[Mccowan et al., 2004] Mccowan, I., Moore, D., Dines, J., Gatica-Perez, D., Flynn, M., Wellner, P., and Bourlard, H. (2004). On the use of information retrieval measures for speech recognition evaluation.

[McFee and Balke, 2019] McFee, B. and Balke, S. (2019). Librosa. `https://github.com/librosa/librosa`. Visited March 21st 2019.

[Microsoft, 1991] Microsoft (1991). wav. https ://fileinfo.com/extension/wav. Visited march 21st 2019.

[Microsoft, 2017] Microsoft (2017). Microsoft's speech recognition is now as good as a human transcriber. `https://futurism.com/microsofts-speech-recognition-is-now-as-good-as-a-human-transcriber`. Consulté le 24/02.2019.

[Mishra and Jain, 2015] Mishra, A. and Jain, S. (2015). A survey on question answering systems with classification. *Journal of King Saud University - Computer and Information Sciences*, 28.

[Mitchell, 1997] Mitchell, T. (1997). *Machine Learning*. McGraw-Hill Science/Engineering/Math.

[Mnih et al., 2014] Mnih, V., Heess, N., Graves, A., et al. (2014). Recurrent models of visual attention. In *Advances in neural information processing systems*, pages 2204–2212.

[Mohamed et al., 2015] Mohamed, R., Ragab, M., Abdelnasser, H., El-Makky, N. M., and Torki, M. (2015). Al-bayan : A knowledge-based system for arabic answer selection. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, pages 226–230.

[NaturalReader, 2017] NaturalReader (2017). Natural reader. `https://www.naturalreaders.com/`. visited January 24th 2019.

[Nguyen, 2018b] Nguyen, M. (2018b). Illustrated guide to recurrent neural networks. `https://towardsdatascience.com/introduction-to-recurrent-neural-networks-rnn-with-dinosaurs-790e7\4e3e6f6`. visited March 24th 2019.

[Nguyen, 2018a] Nguyen, M. (September 2018a). Illustrated guide to lstm's and gru's : A step by step explanation. `https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-4\4e9eb85bf21`. Visited April 16th 2019.

[Nuance, 2019] Nuance (2019). Dragon mobile sdk, inc. `https://www.nuance.com/dragon/for-developers/dragon-software-developer-kit.html`. Visited January 24th 2018.

[O. Arik et al., 2017] O. Arik, S., Chrzanowski, M., Coates, A., Diamos, G., Gibiansky, A., Kang, Y., Li, X., Miller, J., Raiman, J., Sengupta, S., and Shoeybi, M. (2017). Deep voice : Real-time neural text-to-speech.

[O'Shea and Nash, 2015] O'Shea, K. and Nash, R. (2015). An introduction to convolutional neural networks.

[Peixeiro, 2019] Peixeiro, M. (April 15th 2019). Introduction to recurrent neural networks (rnn). `https://towardsdatascience.com/`

introduction-to-recurrent-neural-networks-rnn-with-dinosaurs-790e7\
4e3e6f6. [visited March 24th 2019].

[Pham, 2017] Pham, H. (2017). Pyaudio. http ://-people.csail.mit.edu/hubert/pyaudio/. Visited june 5th 2019.

[Pinola, 2011] Pinola, M. (November 4th 2011). History of voice recognition : from audrey to siri. https://www.itbusiness.ca/news/history-of-voice-recognition-from-audrey-to-siri/15008.

[Python, 2019] Python (2019). pickle. https ://-docs.python.org/3/library/pickle.html. Visited March 21st 2019.

[Qatab and N. Ainon, 2010] Qatab, B. and N. Ainon, R. (2010). Arabic speech recognition using hidden markov model toolkit(htk). volume 2, pages 557 – 562.

[QCRI, 2017] QCRI (2017). Automatic dialect detection repository. https://github.com/qcri/dialectID/tree/master/data. Visited April 6th 2019.

[QCRI, 2018] QCRI (2018). Automatic speech recognition. https://arabicspeech.org/resources. Visited April 6th 2019.

[Rabiner, 1989] Rabiner, L. (1989). A tutorial on hidden markov models and selected applications on speech recognition. *Proceedings of the IEEE*, 77 :257 – 286.

[Rabiner and Juang, 1986] Rabiner, L. and Juang, B. (1986). An introduction to hidden markov models. *IEEE ASSP Magazine*.

[Rashidul Hasan et al., 2004] Rashidul Hasan, M., Jamil, M., Rabbani, G., and Rahman, M. S. (2004). Speaker identification using mel frequency cepstral coefficients. *Proceedings of the 3rd International Conference on Electrical and Computer Engineering (ICECE 2004)*.

[Reddy and Madhavi, 2017] Reddy, A. C. O. and Madhavi, K. (2017). A survey on types of question answering system. *IOSR Journal of Computer Engineering (IOSR-JCE) e-ISSN :2278-0661, p-ISSN :2278-8727, Volume 19, Issue*.

[Reynolds, 2009] Reynolds, D. (2009). Gaussian mixture models. *MIT Lincoln Laboratory*.

[Reynolds and Rose, 1995] Reynolds, D. and Rose, R. (1995). Robust text-independent speaker identification using gaussian mixture speaker models. *Speech and Audio Processing, IEEE Transactions on*, 3 :72 – 83.

[Riverbank, 2018] Riverbank (2018). Pyqt. https ://riverbankcomputing.com/software/pyqt/intro. Visited june 5th 2019.

[Robert, 2019] Robert, J. (2019). Pydub. https://github.com/jiaaro/pydub. Visited March 21st 2019.

[Rodola, 2019] Rodola, G. (2019). psutil. https ://github.com/giampaolo/psutil. Visited May 2nd 2019.

[S Jurafsky and Martin, 2000] S Jurafsky, D. and Martin, J. (2000). *Speech and Language Processing*.

[Salvi, 1999] Salvi, G. (1999). Developing acoustic models for automatic speech recognition in swedish. *The European Student Journal of Language and Speech*.

[Settles and Rylander, 2002] Settles, M. and Rylander, B. (2002). Neural network learning using particle swarm optimizers. *Advances in information science and soft computing*, pages 224–226.

[Shalev-Shwartz and Ben-David, 2013] Shalev-Shwartz, S. and Ben-David, S. (2013). Understanding machine learning. from theory to algorithms. *Understanding Machine Learning : From Theory to Algorithms*.

[Silvester et al., 2003] Silvester, S., McDougall, D., Firing, E., et al. (2003). Matplotlib. https ://matplotlib.org/. Visited May 2nd 2019.

[Skinner, 2008] Skinner, J. (2008). Sublime text. https ://www.sublimetext.com/. Visited March 21st 2019.

[Sphinx, 2017] Sphinx, C. (2017). Cmu sphinx open source speech recognition toolkit. `https://cmusphinx.github.io/`. Visited January 24th 2018.

[Strik and Cucchiarini, 1999] Strik, H. and Cucchiarini, C. (1999). Modeling pronunciation variation for asr : A survey of the literature. *Speech Communication*, 29(2-4) :225–246.

[Stupina et al., 2016] Stupina, A., Zhukov, E., Ezhemanskaya, S. N., Karaseva, M. V., and Korpacheva, L. N. (2016). Question-answering system.

[Sutskever et al., 2014] Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

[SyncedReview, 2017] SyncedReview (September 5th 2017). A brief overview of attention mechanism. `https://medium.com/syncedreview/a-brief-overview-of-attention-mechanism-13c578ba9129`. Visited April 19th 2019.

[T. Lowerre, 1976] T. Lowerre, B. (1976). The harpy speech recognition system, phd thesis. *Tech rep*.

[Torvalds, 2005] Torvalds, L. (2005). Git. https ://git-scm.com/. Visited May 2nd 2019.

[Touzet, 2019] Touzet, C. (2019). Les reseaux de neurones artificiels introduction au connexionnisme cours, exercices et travaux pratiques.

[Trigui et al., 2010] Trigui, O., Belguith, L. H., and Rosso, P. (2010). Defarabicqa : Arabic definition question answering system. In *Workshop on Language Resources and Human Language Technologies for Semitic Languages, 7th LREC, Valletta, Malta*, pages 40–45.

[Tulyakov, 2014] Tulyakov, S. (2014). Hidden markov models. *venu@cubs.buffalo.edu*.

[van Rossum, 1989] van Rossum, G. (1989). Python. https ://docs.python.org/3/faq/general.htmlwhat-is-python. Visited April 17th 2019.

[Wang et al., 2017] Wang, Y., Skerry-Ryan, R., Stanton, D., Wu, Y., J. Weiss, R., Jaitly, N., Yang, Z., Xiao, Y., Chen, Z., Bengio, S., Le, Q., Agiomyrgiannakis, Y., Clark, R., and A. Saurous, R. (2017). Tacotron : Towards end-to-end speech synthesis. pages 4006–4010.

[Xu et al., 2015] Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhudinov, R., Zemel, R., and Bengio, Y. (2015). Show, attend and tell : Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057.

[Yu and Deng, 2014] Yu, D. and Deng, L. (November 11th 2014). *Automatic Speech Recognition, A Deep Learning Approach*. Springer Publishing Company.

[Zhang and LeCun, 2017] Zhang, X. and LeCun, Y. (2017). Which encoding is the best for text classification in chinese, english, japanese and korean ? *arXiv preprint arXiv :1708.02657*.

ANNEXE A

DOCUMENTATION

A.1 models

A.1.1 Speech_API

ASER_system est la fonction principale de ce module qui contient un système de reconnaissance de la parole pour la langue arabe, ce dernier peut être intégré à n'importe quelle application.

A.1.2 Apprentissage

A.1.2.1 encoder_decoder

Nous commençons par vous présentez 4 types d'encodeurs :

1. **Encodeur LSTM :**

   ```
   get_encoder_states_LSTM(encoder_inputs, latent_dim, return_sequences=False)
   ```

   Fonction implémentant un encodeur avec une couche LSTM

2. **Encodeur GRU :**

   ```
   get_encoder_states_GRU(encoder_inputs, latent_dim, return_sequences=False)
   ```

   Fonction implémentant un encodeur avec une couche GRU

3. **Encodeur Bi-LSTM :**

encoder_bi_LSTM(encoder_inputs, latent_dim, return_sequences=False)

Fonction implémentant un encodeur avec une couche LSTM bidirectionnelle

4. **Encodeur Bi-GRU :**

encoder_bi_GRU(encoder_inputs, latent_dim, return_sequences=False)

Fonction implémentant un encodeur avec un couche GRU bidirectionnelle

Toutes ces fonctions ont les mêmes arguments en entrée et renvoie la même chose en sortie.

Arguments

- encoder_inputs : 3D Numpy Array, les données en entrèe pour l'encodeur.
- latent_dim : Entier positif, dimensionnalité de l'espace de sortie.
- return_sequences=False : Booléen. Indique s'il faut renvoyer la dernière sortie de la séquence de sortie ou la séquence complète du LSTM.

Retourne

- encoder_states : le vecteur d'états cachés renvoyé par la couche LSTM

Dans ce qui suit, nous présentons 4 types de décodeurs :

1. **Décodeur LSTM :**

get_decoder_outputs_LSTM(encoder_inputs, latent_dim, return_sequences=False)

Fonction implémentant un décodeur avec trois couches LSTM

2. **Décodeur GRU :**

get_decoder_outputs_GRU(encoder_inputs, latent_dim, return_sequences=False)

Fonction implémentant un décodeur avec trois couches GRU

3. **Décodeur Bi-LSTM :**

decoder_for_bidirectionnal_encoder_LSTM(encoder_inputs, latent_dim, return_sequences=False)

Fonction implémentant un décodeur avec trois couches LSTM bidirectionnelles

4. **Décodeur Bi-GRU :**

decoder_for_bidirectionnal_encoder_GRU(encoder_states, decoder_inputs, latent_dim)

Fonction implémentant un décodeur avec trois couches GRU bidirectionnelles

Toutes ces fonctions ont les mêmes arguments en entrée et renvoie la même chose en sortie.

Arguments

- encoder_inputs : 3D Numpy Array, les données en entrèe pour l'encodeur.
- decoder_inputs : 3D Numpy Array. Les données en entrée du décodeur
- latent_dim : Entier positif, dimensionnalité de l'espace de sortie.

Retourne

- decoder_states : le vecteur d'états cachés renvoyé par la couche LSTM
- decoder_outputs : 3D Numpy array. Représente le résultat obtenu par le décodeur

A.1.2.2 baseline_model

1. **Modèle Baseline en utilisant des couches GRU**

   ```
   train_baseline_seq2seq_model_GRU(mfcc_features, target_length, latent_dim, word_level)
   ```

   Modèle de base implémenté en utilisant des couches GRU pour l'encodeur ainsi que le décodeur

2. **Modèle Baseline en utilisant des couches GRU bidirectionnelles**

   ```
   train_bidirectional_baseline_seq2seq_model_GRU(encoder_inputs, latent_dim, return_sequences=False)
   ```

   Modèle de base implémenté en utilisant des couches GRU bidirectionnelles pour l'encodeur ainsi que le décodeur

3. **Modèle Baseline en utilisant des couches LSTM**

   ```
   train_baseline_seq2seq_model_LSTM(encoder_inputs, latent_dim, return_sequences=False)
   ```

   Modèle de base implémenté en utilisant des couches LSTM pour l'encodeur ainsi que le décodeur

4. **Modèle Baseline en utilisant des couches LSTM bidirectionnelles**

   ```
   train_bidirectional_baseline_seq2seq_model_LSTM(encoder_inputs, latent_dim, return_sequences=False)
   ```

   Modèle de base implémenté en utilisant des couches LSTM bidirectionnelles pour l'encodeur ainsi que le décodeur

   Arguments

- mfcc_features : 3D Numpy Array
- target_length : Entier positif, taille de l'output du décodeur
- latent_dim : Entier positif, dimensionnalité de l'espace de sortie.
- word_level : Booléen, 0 si les transcriptions sont basées caractères, 1 sinon.

Retourne

- model : Définit une instance de model séquence à séquence
- encoder_states : le vecteur d'états cachés renvoyé par la couche LSTM

A.1.2.3 cnn_based_model

1. **Modèle avec couche convolutionnelles et Encodeur/Décodeur avec des couches GRU**

   ```
   train_cnn_seq2seq_model_GRU(mfcc_features, target_length, latent_dim, word_level)
   ```

   Modèle implémenté en utilisant des couches GRU pour l'encodeur ainsi que le décodeur

2. **Modèle avec couche convolutionnelles et Encodeur/Décodeur avec des couches GRU bidirectionnelles**

   ```
   train_bidirectional_baseline_seq2seq_model_GRU(encoder_inputs, latent_dim, return_sequences=False)
   ```

   Modèle de base implémenté en utilisant des couches GRU bidirectionnelles pour l'encodeur ainsi que le décodeur

3. **Modèle avec couche convolutionnelles et Encodeur/Décodeur avec des couches LSTM**

   ```
   train_baseline_seq2seq_model_LSTM(encoder_inputs, latent_dim, return_sequences=False)
   ```

   Modèle de base implémenté en utilisant des couches LSTM pour l'encodeur ainsi que le décodeur

4. **Modèle avec couche convolutionnelles et Encodeur/Décodeur avec des couches LSTM bidirectionnelles**

   ```
   train_bidirectional_baseline_seq2seq_model_LSTM(encoder_inputs, latent_dim, return_sequences=False)
   ```

   Modèle de base implémenté en utilisant des couches LSTM bidirectionnelles pour l'encodeur ainsi que le décodeur
   Arguments

- mfcc_features : 3D Numpy Array
- target_length : Entier positif, taille de l'output du décodeur
- latent_dim : Entier positif, dimensionnalité de l'espace de sortie.
- word_level : Booléen, 0 si les transcriptions sont basées caractères, 1 sinon.

Retourne

- model : Définit une instance de model séquence à séquence
- encoder_states : le vecteur d'états cachés renvoyé par la couche LSTM

A.1.2.4 model_callback

```
ModelSaver(model_name, model_path, encoder_states, drive_instance, word_level=True, output_length=16)
```

Classe permettant de sauvegarder le modèle après cheque itération lors de l'apprentissage

Arguments

- model_name : chaine de caratères, le nom du modèle.
- model_path : chaine de caratères, le chemin vers le modèle.
- encoder_states
- word_level : Booléen, 0 si les transcriptions sont basées caractères, 1 sinon.
- output_length : entier positif, taille de l'output.

Méthodes

- on_epoch_end(epoch, logs=None)

A.1.2.5 train_model

```
Seq2SeqModel(latent_dim=300, epochs=50, model_architecture=5, data_generation=True, word_level=False)
```

Classe permettant de faire le traitement nécessaire pour l'apprentissage d'un modèle seéquence à séquence.

Arguments

- latent_dim : entier positif,
- epochs : entier positif,
- model_architecture : entier positif
- data_generation : Booléen,
- word_level : Booléen, 0 si les transcriptions sont basées caractères, 1 sinon.

Méthodes

- test_model()
- train_model()
- validation_generator()
- split_data_generator_dict_test()
- split_data_generator_dict(batch_size)
- split_data_generator_dict_word_level(batch_size)
- split_data_generator_dict_word_level_test()
- get_test_data(audio_file, transcripts_file)
- get_data(audio_file, transcripts_file)

A.1.2.6 plot_performance

```
plot_train_loss_acc(model_hist_path, word_level)
```

Fonction permettant d'afficher les courbes d'appentissage et de test (loss et accuracy)

Arguments

- model_hist_path : le chemin vers l'historique du modèle
- word_level : Booléen, 0 si les transcriptions sont basées caractères, 1 sinon.

A.1.2.7 char_inference

```
Char_Inference()
```

Fonction permeettant l'inférence en utilisant un modèle basé caractères.

A.1.2.8 word_inference

```
Word_Inference(model_path, latent_dim)
```

Fonction permeettant l'inférence en utilisant un modèle basé mots.

Arguments

- model_path : chaine de caractères, chemain vers le modèle choisi.
- latent_dim : entier positif,

A.1.2.9 word_correction

```
correct_word(word_to_correct)
```

Fonction permettant la correction d'un mot en utilisant un modèle de language.
Arguments

- word_to_correct : chaine de caractères, le mot à corriger.

A.2 data

A.2.1 Dataset generation

A.2.1.1 audio_transcript_map

```
map_audio_transcripts()
```

Fonction permettant de mapper chaque audio avec sa transcription.

A.2.1.2 parse_xml

```
generate_transcriptions_file(transcriptions_desc, output_path)
```

Fonction permettant la génération d'un fichier des transcriptions à partir d'un XML
Arguments

- transcriptions_desc
- output_path

A.2.1.3 split_audio

```
split_audio(audio_entry, transcriptions_desc, audio_output_dir)
```

Fonction permettant de diviser un audio en plusieurs sous-audio.
Arguments

- audio_entry
- transcriptions_desc
- audio_output_dir

A.2.1.4 parse_dataset

```
generate_dataset()
```

Fonction permettant la génération d'un fichier Pickle contenant toutes les informations nécessaire et les données du dataset

A.2.2 pré-traitement des données

A.2.2.1 normalisation

```
normalize_encoder_input(dataset)
```

Fonction permettant la normalisation des entrés de l'encodeur
Arguments

- dataset

A.2.2.2 transcript_preprocessing

```
transcript_preprocessing(transcription, special_characters_table)
```

Fonction permettant de faire le pré-traitement des transcriptions
Arguments

- transcription
- special_characters_table

A.3 lib

A.3.1 AudioInput

```
AudioInput(path, transcript)
```

Classe permettant de représenter un audio, ses caractéristiques ainsi que sa transcription
Arguments

- path
- transcript

A.3.2 Transcript

```
Transcript(start_time, end_time, content)
```

Arguments

- start_time
- end_time
- content

A.4 utils

A.4.1 character_convesion

1. Conversion d'une phrase de la langue arabe au buckwalter :

   ```
   arabic_to_buckwalter(arabic_sentence)
   ```

   Arguments

   - arabic_sentence

2. Conversion d'une phrase du buckwalter à l'arabe :

   ```
   buckwalter_to_arabic(buckwalter_sentence)
   ```

   Arguments

   - buckwalter_sentence

3. Conversion des chiffres en lettres :

   ```
   convert_numeral_to_written_number(number)
   ```

   Arguments

   - number

4. Conversion ...

   ```
   numerical_to_written_numbers_table(inf_number=0,
   sup_number=10001)
   ```

   Arguments

   - inf_number=0
   - sup_number=10001

A.4.2 pickle_management

```
generate_pickle_dataset(threshold)
```

Arguments

- threshold

A.5 etc

Toutes les constantes sont enregistrées dans settings, nous retrouvant :

- DATA_PATH : Chemin vers le dossier *data*.
- PICKLE_FILE_PATH : Chemin vers le fichier pickle.
- PICKLE_PARTITIONS_PATH : Chemin vers le dossier *partitions*.
- PICKLE_PAD_FILE_PATH
- GENERATED_DATA_PATH : Chemin vers le dossier generated_data.
- GENERATED_DATA_WAV_PATH : Chemin vers le dossier wav de generated_data.
- GENERATED_DATA_TRANSCRIPTS_PATH : Chemin vers le dossier transcripts de generated_data.
- DISTINCT_CHARACTERS_PATH : Chemin vers le fichier contenant les caractères distincts du corpus.
- NORMALIZATION_PATH : Chemin vers le dossier *normalisation*
- TRAINED_MODELS_PATH : Chemin vers le dossier *trained_models*
- DATASET_SPLIT_PATH : Chemin vers le dossier *dataset_split*
- DATASET_SPLIT_TRAIN_PATH : Chemin vers le dossier *train* de *dataset_split*
- DATASET_SPLIT_TEST_PATH : Chemin vers le dossier *test* de *dataset_split*
- MFCC_FEATURES_LENGTH
- CHARACTER_SET : Ensemble de caractères du corpus.
- WORD_SET : Ensemble des mots du corpus.
- LONGEST_WORD_LENGTH : Taille du plus grand mot.
- WORD_TARGET_LENGTH
- TOTAL_SAMPLES_NUMBER

Résumé

La reconnaissance automatique de la parole est une branche de l'intelligence artificielle qui permet la transcription du langage naturel humain exprimé par la parole. Un système de reconnaissance de la parole prend en entrée un signal audio capturé par un microphone pour produire une sortie sous forme de texte qui sera par la suite traité selon le besoin.

Dans le cadre de ce projet, nous avons conçu et implémenté *ASeR-System*, un système de reconnaissance de la parole pour la langue arabe. Pour la conception de ce système, nous avons opté pour une approche End-to-End en utilisant des techniques d'apprentissage profond. Nous avons proposé différentes architectures de modèles à savoir le modèle de base, le modèle à couches convolutionnelles et ce en utilisant deux types d'encodage, l'encodage basé mots et l'encodage basé caractères .L'apprentissage a été effectué sur le corpus QCRI contenant 1200 heures de dialogue des programmes d'Al-Jazeera TV intégrant plus d'une centaine de locuteurs.

Par la suite nous avons développé une application qui intègre la reconnaissance de la parole à un système de questions-réponses. L'application permet à un utilisateur de poser une question oralement, ensuite ASeR-System la transformant en texte, avant d'utiliser un système de questions-réponses pour obtenir la réponse à la question.

Abstract

Automatic speech recognition is a branch of artificial intelligence that allows the transcription of human natural language expressed by speech. A speech recognition system inputs an audio signal captured by a microphone to produce a text output which will subsequently be processed as needed.

As part of this project, we designed and implemented ASeR-System, a speech recognition system for the Arabic language. For the design of this system, we went for an End-to-End approach using deep learning techniques. We have put in place different model architectures namely the baseline model and the convolutional layers based model while using for each, two types of encoding, word-based encoding and character-based encoding. Learning has been done on the QCRI corpus which consists of 1200 hours of Al-Jazeera TV programmes that feature more than a hundred speakers.

Subsequently we developed an application that uses speech recognition with a question-answering system. The application allows a user to ask a question orally, then ASeR-System transforms it into text, before using the question-answering to get the answer to the question.

ملخص

التعرف التلقائي على الكلام هو فرع من الذكاء الاصطناعي الذي يسمح بنسخ اللغة (أي التمثيل النصي للغة) الطبيعية للإنسان التي يعبر عنها بالكلام أو تحت التعبير الصوتي. يقوم نظام التعرف على الكلام بإدخال إشارة صوتية يتم التقاطها بواسطة ميكروفون لإنتاج نص يتم معالجته لاحقًا حسب الحاجة.

كجزء من هذا المشروع ، قمنا بتصميم وتنفيذ ASeR-System ، وهو نظام التعرف على الكلام للغة العربية. لتصميم هذا النظام ، اخترنا أسلوب End-to-End باستخدام تقنيات التعلم العميق. لقد اقترحنا تصميمات مختلفة للنماذج وهي النموذج الأساسي ونموذج الطبقة التلافيفية وهذا باستخدام نوعين من الترميز ، الترميز القائم على الكلمات والترميز القائم على الأحرف. وقد تم تنفيذ التعلم على مدونة QCRI التي تحتوي على 1200 ساعة من الحوارات من برامج قناة الجزيرة التلفزيونية والتي شارك فيها أكثر من مائة متحدث.

بعد ذلك قمنا بتطوير تطبيق يدمج التعرف على الكلام مع نظام الأسئلة والأجوبة. يسمح التطبيق للمستخدم بطرح سؤال شفهيًا ، ثم يقوم ASeR-System بتحويله إلى نص ، قبل استخدام نظام الأسئلة والأجوبة للحصول على إجابة على السؤال.