Sommaire

Table des figures

Liste des tableaux

Déclaration de non-plagiat

Nous, soussignés :

M. Wissam BENHADDAD et
M. Yasser BOURAHLA,

auteurs de ce mémoire de fin d'études de master N° 061/2019, intitulé :

Exploration de la reconnaissance de la parole pour le développement d'aspects d'un assistant de manipulation de l'ordinateur.

déclarons, chacun, 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.

Signatures :

Yasser **Wissam**

Alger, Dimanche 30 Juin 2019

Remerciements

Nous voudrions commencer par remercier DIEU, le Miséricordieux, qui nous a donné les moyens et la force d’aller jusqu’au bout de ce projet.

Nous tenons ensuite à remercier nos deux promoteurs Monsieur GUESSOUM Ahmed et Madame AKLI Karima pour nous avoir guidés à travers toutes les étapes de notre projet, et pour tous les encouragements et l’aide précieuse qu’ils nous ont prodigués tout au long du travail.

Nous sommes également reconnaissants envers Madame KHELLAF, présidente du jury, et Madame AMROUS, membre du jury, pour avoir accepté de consacrer leur précieux temps afin de juger notre travail.

Enfin, nous adressons nos plus sincères remerciements à toutes les personnes qui nous ont aidés ou encouragés de près ou de loin.

Yasser & Wissam

Dédicaces

Je dédie ce modeste travail à mes parents qui m'ont toujours soutenu et qui ont toujours tout sacrifié pour que je puisse réussir dans ma vie.

Je dédie ensuite cet humble travail à tous mes amis et surtout Yasser, Sofiane, Adel et Ghiles. À mes proches qui ne sont plus là, qu'ils reposent en paix. Je tiens également à dédier ce projet à nos chers promoteurs Monsieur Ahmed Guessoum et Madame Karima Akli qui ont redoublé d'efforts pour nous assister, nous conseiller et surtout nous apporter leur immense savoir afin de réaliser ce travail.

Je dédie finalement ce travail à mon binôme qui n'a jamais cessé de me proposer son aide dans les moments les plus difficiles et qui a redoublé d'efforts pour la réalisation de ce projet.

Que le bon Dieu vous garde tous.

Wissam

Dédicaces

Je tiens à dédier cet humble travail à mes parents, que je ne remercierai jamais assez ; mon frère, qui est la personne qui m'a inspiré le plus ; et ma sœur, qui m'a beaucoup appris et qui a toujours été présente avec moi, ainsi qu'au reste de ma famille qui m'ont soutenu au cours de la réalisation de ce projet.

Je le dédie ensuite à tous mes amis qui m'ont accompagné tout au long de mon parcours.

Je tiens également à dédier ce projet à nos chers promoteurs Monsieur Guessoum et Madame Akli qui ont tout donné pour nous aider à le réaliser. Ils ont été plus que promoteurs ; des exemples que j'admire et que je souhaite suivre dans ma vie.

Je finis par dédier ce travail à mon très cher binôme sans qui ce travail n'aurait pas été possible.

Yasser

Résumé

Les récents progrès de l'apprentissage automatique montrent clairement que l'intelligence artificielle dominera l'avenir. L'interaction homme-machine a particulièrement attiré l'attention des plus grandes industries informatiques du monde. Ceci est particulièrement vrai avec la poussée de l'apprentissage profond qui a permis aux ordinateurs d'atteindre un niveau élevé de compréhension de la langue.

Néanmoins, ces grandes avancées n'ont pas encore atteint le niveau de communication humain et semblent être très en retard. Les ordinateurs sont soit limités à la gestion de tâches spécifiques, telles que des assistants intelligents, soit dépourvus de compréhension linguistique réelle, tels que les chat-bots.

À travers ce travail, nous nous familiarisons avec les systèmes de gestion de dialogue vocal et leurs composants. En tirant partie des approches d'intelligence artificielle, allant des modèles acoustiques, à la génération de langage naturel, tout en passant par la compréhension du langage naturel et la gestion de dialogue ; notre solution peut aider l'utilisateur à manipuler l'ordinateur à l'aide de directives vocales.

Mots clés : Assistant personnel intelligent, Apprentissage par renforcement, Apprentissage profond, Intelligence artificielle, Interaction homme-machine, Compréhension du langage naturel, Classification d'intention, Reconnaissance automatique de la parole, Gestion de dialogue, Bases de connaissances.

ملخص

أوضحت التطورات الحديثة في مجال التعلم الآلي أنّ الذكاء الاصطناعي سيهيمن على حياة الإنسان في المستقبل القريب. وعلى وجه الخصوص، ألقى عمالقة الإعلام الآلي تركيزًا خاصًا على مجال التفاعل بين الإنسان والآلة. فقد حصل ذلك تزامنًا مع تطور التعلم العميق الذي مهد الطريق أمام أجهزة الكمبيوتر للوصول إلى مستوى عالٍ من فهم اللغة.

ومع ذلك ، فإن هذه التطورات العظيمة لم تصل بعد إلى المستوى المنشود و تبقى بعيدة عن طرق تواصل البشر الطبيعية فيما بينهم إذ تقتصر أجهزة الكمبيوتر إما على إدارة مهام محددة مثل المساعدين الأذكياء أو تفتقر إلى الفهم الفعلي للّغة.

سنتعرف من خلال هذا العمل على أنظمة الحوار ونفهم مكوناتها الأساسية. لنستطيع بذلك و باستخدام أساليب الذكاء الاصطناعي، من النماذج الصوتية إلى توليد النص مرورًا بفهم اللغة و إدارة الحوار ، أن نقدم حلاً يمَّكِن المستخدم من التحكم في الكمبيوتر عن طريق توجيهات شفوية.

الكلمات الدالة : مساعد شخصي ذكي ، التعلم العميق ، الذكاء الاصطناعي ، التفاعل بين الإنسان والحاسوب ، فهم اللغة الطبيعية ، تصنيف النوايا ، التعرف التلقائي على الكلام ، إدارة الحوار ، قواعد المعرفة.

Abstract

Recent advances in machine learning are making it clear that the near future will be dominated by artificial intelligence. In particular, human-machine interaction received a great focus from the world's biggest computer science industries. This is especially true with the surge of deep learning which paved the way for computers to reach a high level of language understanding.

Nonetheless, these great advances did not yet reach human level communications and seem to be lagging behind. Computers are either restricted to the management of specific tasks like smart assistants or lack the actual language understanding like chatbots.

Through this work, We familiarize ourselves with spoken dialogue systems and understand their components. By taking advantage of artificial intelligence approaches, ranging from acoustic models, to natural language generation, while going through natural language understanding and dialogue management; our solution can assist the user with computer manipulation using spoken directives.

Keywords : Smart Personal Assistant, Deep Learning, Reinforcement Learning, Artificial Intelligence, Human-Computer Interaction, Natural Language Understanding, Intent Classification, Automatic Speech Recognition, Dialog Management, Knowledge Bases.

Introduction générale

> *"Nous entrons dans un nouveau monde. Les technologies d'apprentissage automatique, de reconnaissance de la parole et de compréhension du langage naturel atteignent un niveau de capacités supérieur. Le résultat final est que nous aurons bientôt des assistants intelligents pour nous aider dans tous les aspects de nos vies."*
>
> – Amy Stapleton, Opus Research 2015

Plus de quarante ans se sont écoulés depuis la présentation du premier assistant virtuel commandé vocalement par la compagnie IBM (VanDijck, 2005). Déjà à cette époque là, ce fut présenté comme une révolution technologique. Un programme qui pouvait reconnaître 16 mots et les chiffres de 0 à 9. Quelques décennies plus tard, nous nous retrouvons avec des assistants capables de reconnaître, comprendre et parler plusieurs langues. Ces assistants intelligents sont la nouvelle génération d'intelligence artificielle capable de s'intégrer dans nos vies personnelles et professionnelles. Les assistants virtuels intelligents (Smart Personal Assistants, SPA) sont un exemple de cette intelligence artificielle.

Un domaine d'application pour les SPAs qui a particulièrement émergé est celui de l'aide à la manipulation d'un ordinateur personnel. Selon certains experts (Milhorat et al., 2014 ; Trappl, 2013 ; Knote et al., 2018), l'époque où nous utilisons encore le clavier, la souris et l'écran est une étape de transition. Le futur se trouve dans l'exploitation de la parole comme moyen de communication principal avec les machines ; et ce futur est proche. La course à la réalisation d'assistants qui excellent dans plusieurs domaines a commencé il y a quelques années avec l'entrée de grandes compagnies comme Google et Apple dans le secteur (Tulshan and Namdeorao Dhage, 2019). Par la suite, de très grands efforts ont été fournis dans le but d'améliorer l'expérience d'utilisation de ces assistants. Les chercheurs ainsi que les industriels se sont orientés vers cette solution très rapidement, notamment en investissant dans la recherche de solutions exploitant l'apprentissage automatique (Tulshan and Namdeorao Dhage, 2019).

Nous sommes sans doute actuellement entrain de vivre une époque importante de l'intelligence artificielle, avec le développement de l'apprentissage automatique, surtout dans le domaine de la reconnaissance automatique de la parole, de la compréhension automatique du langage naturel et plus récemment de l'apprentissage par renforcement (Knote et al.,

2018). L'accomplissement de notre rêve de converser avec une machine semble à la fois proche en termes de temps, mais loin en termes de progrès. Les utilisateurs réguliers de produits technologiques sont confrontés à une technologie nouvelle mais prometteuse. Ce secteur d'activités peut donc s'avérer très prolifique si les efforts fournis sont suffisamment conséquents.

C'est dans ce contexte que s'inscrivent les travaux de notre projet de fin d'études de master. En effet, pour ne pas être en marge de l'évolution dans ce secteur qu'est la personnalisation des services électroniques, cette thématique avec l'application de techniques d'apprentissage automatique, qui sont la base de nombreux systèmes actuels, Google Assistant, Apple Siri,Microsoft Cortana (López et al., 2017), et des systèmes Open source comme Rasa (Bocklisch et al., 2017), devrait être un défi pour l'Algérie en général et pour nous en particulier. Très motivés par l'envie de nous initier à ce domaine, nous espérons ainsi apporter notre modeste contribution. Nous pensons aussi que les plus ambitieux des projets commencent avec des contributions à petites échelles. Notre assistant aura pour but d'améliorer l'expérience d'utilisation d'un ordinateur, ceci en effectuant des tâches rudimentaires, efficacement et sans réel effort de la part de l'utilisateur hormis l'énonciation de la requête. En utilisant des techniques d'intelligence artificielle d'actualité comme la reconnaissance automatique de la parole, la compréhension du langage naturel et l'apprentissage par renforcement, ce projet se veut assez ambitieux et vise à faire gagner du temps à tout utilisateur d'un ordinateur de bureau ou portable.

Dans cette optique, nous nous sommes inspirés dans notre étude des travaux de la littérature sur les assistants personnels intelligents. Nous passerons en revue les aspects théoriques qui sont utilisés dans des solutions considérées comme état de l'art du domaine. Nous nous baserons sur ces techniques pour la conception des modules de notre système tout en les adaptant à nos besoins.

Ce mémoire se constitue de quatre chapitres. Le premier chapitre sera consacré à la présentation des assistants virtuels intelligents. Le deuxième chapitre se focalisera sur l'étude des travaux existants liés à la thématique de notre sujet comme l'utilisation de l'apprentissage profond, l'apprentissage par renforcement et les ontologies pour le développement d'un assistant virtuel intelligent. Le troisième chapitre traitera de l'étude conceptuelle. Le quatrième et dernier chapitre présentera notre système Speact avec son évaluation, une synthèse des résultats obtenus, et notre interface pour l'application. Enfin, nous clôturerons ce travail avec une conclusion générale et les perspectives envisageables.

Chapitre 1

Assistants virtuels intelligents

1.1 Introduction

Depuis la commercialisation du premier ordinateur grand public (Xerox PARC Alto) en 1973, le monde découvrit pour la première fois ce qui allait devenir l'apparence basique de chaque ordinateur moderne. En effet, la compagnie Xerox fut la première à proposer une interface graphique dotée de fenêtres, d'icônes et d'une souris pour se déplacer et d'un clavier pour écrire du texte. Bien que basique, cette idée lança alors plusieurs autres grandes marques sur le même chemin (IBM, Apple, Compaq ...). Par la suite, beaucoup ont essayé d'améliorer la façon dont l'homme utilisait sa machine : souris plus précise, écran doté d'une plus grande résolution, clavier plus enrichi, voire même l'introduction des écrans tactiles dans certains systèmes embarqués.

Cependant, certains voyaient encore cette façon d'utiliser la machine comme trop primitive, et peu intuitive. En effet, laissez un enfant devant un ordinateur et il prendrait un bon moment pour apprendre à éditer ne serait ce qu'un simple fichier. Pour citer Donald A. Norman (Norman, 2002) :

> "We must design for the way people behave, not for how we would wish them to behave."

que nous pouvons traduire par :

> "Nous devons concevoir selon le comportement des utilisateurs, et non pas selon la façon dont nous voudrions qu'ils se comportent."

L'humanité a fait beaucoup de chemin depuis les années 70, l'utilisation d'un ordinateur de nos jours avec les moyens classiques (souris, clavier, écran ...) est devenue une tâche triviale, voire même une seconde nature. Cependant, l'effort d'utiliser les outils communs reste lui très présent.

La plus naturelle et plus ancienne façon de communiquer pour l'homme a toujours été la parole. Le développement de langues toutes aussi riches et complexes les unes que les autres a permis à l'humanité d'enrichir sa panoplie de moyens de communications. L'avancement le plus naturel pour cette façon de communiquer serait donc de l'étendre aux machines que l'homme a su construire et améliorer au fil des années.

Motivé par cette manière que l'on a de communiquer entre nous, et épaulé par les récentes technologies telles que l'apprentissage automatique, le traitement automatique du langage naturel et l'intelligence artificielle, les plus brillants des chercheurs ont entamé leurs travaux dans cette toute nouvelle direction.

Les Assistants Virtuels Intelligents (Smart Personal Assistant, SPA) sont donc le produit de plusieurs années de recherche, visant tout d'abord à faciliter certaines tâches pour l'utilisateur. Les premiers SPAs étaient conçus comme des agents de conversation ou Chatbots, limités dans leurs actions et dépendant toujours d'un moyen de communication textuel. Ce n'était pas la forme désirée du SPA. Avec l'avancement des recherches sur la reconnaissance automatique de la parole (Automatic Speech Recognition, ASR) et l'émergence de l'apprentissage automatique, les tout premiers assistants virtuels utilisant l'ASR étaient spécialisés dans certains domaines comme les systèmes médicaux d'aide à la décision. Il a ensuite été plus aisé de briser la barrière et de réaliser ce qui était encore une esquisse d'un SPA personnalisé. Aujourd'hui, et ce depuis l'avènement de l'apprentissage profond et la popularisation des Smartphones, de nouveaux SPAs comme Apple Siri (voir 1.5.2), Google Assistant (voir 1.5.1) et Amazon Alexa (voir 1.5.3) ont fait leur apparition, offrant de plus en plus de services personnalisés et spécifiques à chaque utilisateurs.

Dans la suite de ce chapitre nous essayerons de mieux détailler ce qu'est un SPA, ce qui est demandé d'un tel système, ses domaines d'application, en enchaînant par une description d'une pseudo-architecture potentielle de ce système, pour enfin conclure sur les limitations actuelles et les motivations de ce projet.

1.2 L'importance du contexte pour un Assistant Virtuel Intelligent (SPA)

Informellement, un SPA est un type d'agent logiciel qui peut effectuer certaines tâches et proposer des services dédiés aux utilisateurs qui vont d'une simple tâche comme ouvrir une fenêtre, lancer une application ..., à la réalisation de requêtes un peu plus complexes comme réserver une table dans un restaurant en passant un appel vocal. Pour répondre efficacement à toutes sortes de requêtes, un SPA se doit donc de garder trace du contexte courant de sa conversation avec l'utilisateur. Il doit disposer d'un système capable d'enregistrer les informations pertinentes et de savoir les réutiliser, mais aussi de pouvoir déduire lesquelles de ces informations sont manquantes. On parle ici de Context-Awarness ou Sensibilité au contexte, comme vu dans (Knote et al., 2018).

D'après (Knote et al., 2018) et (Abowd et al., 1999), *Day* et *Abwod* définissent un contexte comme suit :

> "A context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves."

qui peut être traduit par :

> “Un contexte est une information qui peut être utilisée pour caractériser l’état d’une entité. Une entité peut être une personne, une place ou un objet, considéré comme pertinent à l’interraction entre l’utilisateur et l’application, ainsi qu’à ces deux derniers eux mêmes.”

Il en découle que pour parvenir à développer un système qui réponde aux besoins individuels et spécifiques de chaque personne, modéliser et prendre en compte le contexte semble être une solution prometteuse.

1.3 Caractéristiques principales d’un SPA

À partir de (Knote et al., 2018), nous pouvons dégager certaines caractéristiques principales qui peuvent être vues comme primordiales pour qualifier un assistant virtuel d’intelligent.

1.3.1 Sensibilité au contexte

Comme vu précédemment dans la section 1.2, le contexte peut être interprété comme tout aspect d’une entité (position d’un objet, couleur d’un objet, température d’une chambre, etc.). Un assistant dit intelligent doit donc être capable de capturer le concept du contexte, d’utiliser et de traiter toute information catégorisée comme contextuelle. Pour être plus précis, un SPA doit être sensible à l’évolution du contexte courant, par le biais de capteurs optiques, de microphones, ou tout ce qui pourrait amener l’utilisateur à faire évoluer la requête qu’il a émise. L’assistant devra donc proposer un système de mise à jour du contexte pour éliminer les informations inutiles et garder celles qui pourraient aider à répondre à la requête de l’utilisateur.

1.3.2 Évolutivité

Comme vu dans la section 1.2, un SPA peut être vu comme un type d’agent. Pour rappel, d’après *Russel* et *Norvig* dans (Russell and Norvig, 2003), un agent est une entité autonome pouvant interagir avec son environnement afin d’accomplir certaines tâches et peut être de plusieurs types :

- Agent à réflexes simples : agent exécutant ses actions à base de règles conditionnelles simples, c.à.d Si *Condition* alors *exécuter actions*. Ils sont ainsi très simplistes et limités dans la portée de leurs actions.
- Agent basé modèle : il est semblable semblable aux agents à réflexes simples. Il est doté d’un modèle interne complexe censé représenter le monde extérieur auquel il a accès. Les actions sont appliquées de la même manière que le précédent type d’agents.
- Agent à but(s) : ce type représente une amélioration des agents simples puisqu’il est doté d’un ensembles d’états buts à atteindre d’une façon ou d’une autre.

- Agent à utilité : il s'agit ici d'agents à buts qui tentent d'aboutir à leurs buts d'une manière optimisée (intelligente) utilisant une fonction de mesure adéquate pour le choix des différents états à atteindre.
- Agent apprenant : agent à utilité enrichi par un module d'apprentissage qui sert de juge pour répondre aux "critiques" des actions qu'il entreprend. Le terme agent évolutif est aussi employé.

Pour ce qui est des SPAs, les systèmes plus récents peuvent être considérés comme des agents apprenants, répondant de ce fait à la contrainte évolutive imposée. On peut donner comme exemple Amazon Alexa qui améliore son module de reconnaissance de la parole après chaque réponse non réfutée par l'utilisateur. Cependant, le domaine de l'auto-évolution des systèmes intelligents est encore un domaine nouveau qui est aidé par les récentes avancées dans l'apprentissage automatique (Knote et al., 2018).

1.3.3 Multimodalité

Afin d'assurer une aisance d'utilisation, les SPAs sont fréquemment amenés à récupérer les requêtes ou données en entrée de la manière la plus naturelle possible, par exemple par le biais de la parole. Cependant, pour garantir une expérience d'utilisation adéquate, l'assistant sera souvent confronté à récupérer ces requêtes de différentes manières, que ce soit à travers une interface graphique, écran tactile ou à travers un texte brut tapé au clavier, voire même à travers des expressions faciales ou des états cognitives/émotionnels (Dingler, 2016), pour ensuite produire une réponse qui elle aussi pourrait éventuellement être de la forme textuelle, sonore ou les deux. Cette capacité à recevoir en entrée et/ou produire une sortie de plusieurs façons différentes est appelée la multimodalité (Luger and Sellen, 2016). Cette caractéristique permet de masquer à l'utilisateur toute la complexité d'acquisition de ses requêtes.

1.3.4 Anthropomorphisme

Plusieurs auteurs tendent à attribuer une grande importance à l'anthropomorphisme des SPAs (Trappl, 2013), qui est :

> "Un mécanisme qui pousse les êtres humains à induire qu'une entité non-humanoïde possède des caractéristiques et comportements propres à l'homme."(Purington et al., 2017)

Ce comportement humanoïde pousserait donc l'utilisateur à se sentir plus à l'aise avec l'assistant, le conduisant ainsi à adopter une façon de communiquer plus humaine et moins structurée qu'avec les autres machines. Ceci est une caractéristique majeure d'un SPA se disant personnalisé.

1.3.5 Flexibilté et Environnement Multi-Plateforme

Malgré leurs récentes prouesses, certains SPAs sont encore restreints à un écosystème fortement dépendant du fabricant. Cowan et al. mentionnent dans (Cowan et al., 2017)

que Apple Siri est limité à l'environnement constitué des produits de la firme à la pomme, n'ouvrant par défaut que les applications de cette dernière quand une requête lui est transmise. C'est un comportement que les assistants devraient éviter, car une indépendance des plateformes utilisées est, certes, très complexe à instaurer, mais offre plus de possibilités aux utilisateurs et aux développeurs pouvant ainsi exploiter la puissance de certaines plateformes (Smartphones, TV connectées, etc.). Avec l'émergence de l'IoT (Internet of Things) et des maisons intelligentes par exemple, c'est un tout nouveau terrain de jeu qui est présenté aux SPAs, offrant plus d'opportunités pour les utilisateurs.

1.4 Domaines d'applications des SPAs

Après avoir vu les différents aspects que les SPAS doivent traiter, nous nous intéresserons maintenant aux types de services et applications fournis pour démontrer qu'ils peuvent bel et bien faciliter certaines tâches à l'homme.

1.4.1 Vie quotidienne

À la base, les SPAs étaient destinés à un usage très personnel comme la gestion des achats dans les supermarchés, ou des guides touristiques de plusieurs destinations de voyage. Cette spécificité a commencé à s'estomper petit à petit avec l'émergence de nouveaux systèmes dédiés à des applications plus générales, comme les maisons intelligentes ou les assistants de planification de tâches. Ceci a permis de mettre encore plus l'accent sur cet aspect de convivialité que les tout premiers SPAs ont tenté de perfectionner. Ainsi, ces assistants spécialisés dans des domaines restreints, tourisme, shopping, détente ..., ont été regroupés dans un seul système plus polyvalent, capable de répondre à des besoins quotidiens divers et variés, allant même à fournir une assistance aux personnes âgées pour leur faciliter les tâches rudimentaires devenues trop fatigantes.

1.4.2 Assistance professionnelle

Les SPAs ont aussi une place dans le monde professionnel. Dans (Imtiaz et al., 2014), il est cité que dans les situations où la marge d'erreur est très petite, par exemple dans les système de manufacturing [1], l'assistance d'un SPA est nécessaire, servant d'une aide à l'humain pour la prise de décision.

Par exemple, dans un environnement de travail hétérogène , nouveaux/anciens employés, hiérarchies des postes..., les SPAs pourraient décharger les employés les plus expérimentés de la tâche d'assister les nouveaux arrivants, pour ainsi aider ces derniers dans leurs tâches et permettre aux autres de se focaliser sur les leurs.

1. Manufacturing ici dans le sens chaîne de montage industrielle, par exemple dans des usines.

1.4.3 Apprentissage Assisté par Ordinateur

Les SPAs peuvent aussi être utiles dans l'enseignement académique que professionnel. D'une part, ils pourraient occuper plusieurs rôles dans les établissements scolaires pour les corrections automatiques de copies, l'enseignement interactif ... (Janson and de Gafenco, 2015) et, d'autre part, accompagner les employés durant leur formation professionnelle.

Ainsi, en considérant les caractéristiques d'un SPA, la sensibilité au contexte est reliée aux expériences antérieures de l'apprenant, permettant au SPA d'adapter son processus d'enseignement en conséquence.

En ce qui concerne l'aspect évolutif du SPA, il lui permet de préférer une approche d'enseignement à une autre selon les résultats de ses apprenants.

1.5 Exemples de SPAs

Pour illustrer la puissance des SPAs les plus récents, nous présentons dans cette section les quatre produits qui dominent le marché courant.

1.5.1 Google Assistant

Lancé en 2016 sous forme d'un chatbot intégré dans l'application Google Allo, Google Assistant s'est vu intégré directement sur les système d'exploitation Android, que ce soit sur smartphones ou tablettes, et plus récemment sur Google Home [2]. Google Assistant est un assistant à tout faire, Il a été développé par les ingénieurs de Google dans le but de faciliter la recherche sur internet, la planification des tâches, l'ajustement des réglages de l'appareil, etc. Son point fort est sa capacité à engager une conversation bi-directionnelle avec l'utilisateur, assurant ainsi une interaction personnalisée variant d'un utilisateur à un autre. Cette capacité lui permet par exemple de proposer certains résultats de recherche selon les précédentes interactions avec l'utilisateur ou de lui proposer une activité si ce dernier lui mentionne qu'il s'ennuie (voir les figures 1.1 et 1.2).

Google Duplex

Une des nouveautés impressionnante de Google Assistant est la fonctionnalité Google Duplex. Toujours en phase de développement, ce module est capable de passer des appels a de vraies personnes et d'avoir une conversation avec elles afin de réaliser une tâche demandée par l'utilisateur comme par exemple réserver une chambre d'hôtel, une table au restaurant, etc.

2. Appareil servant à contrôler les composants d'une smart-house ainsi que l'utilisation des différents services de Google

1.5.4 Microsoft Cortana

Cortana est la tentative de la part de Microsoft d'intégrer un assistant dans son système d'exploitation Windows 10 et WindowsPhone. Il propose divers services de base tel que planifier des tâches, exécuter des commandes via la parole, et pour répondre à des questions, analyser des résultats de recherche sur le moteur de recherche de Microsoft, Bing.

1.6 Conclusion

À travers les sections précédentes, nous avons présenté un certain nombre d'aspects d'un assistant virtuel intelligent. Nous avons donc pu apprécier la potentielle puissance d'un tel système s'il venait à être perfectionné d'avantage.

En effet, en examinant les domaines d'applications, il est facile de déduire que le recours à un SPA peut grandement faciliter certaines tâches, que ce soit celles qui sont les plus triviales pouvant retarder d'autres tâches plus importantes, ou bien celles qui doivent faire appel à la précision et à la grande capacité de calcul des machines, assurant ainsi des résultats précis et rapidement délivrés.

À la fin de ce chapitre nous pouvons donc mettre en valeur la place primordiale que pourraient avoir les SPAs s'ils arrivaient à maturité, c.à.d briser la barrière qui sépare les humains de la machine, parvenant ainsi à faire partie de la vie quotidienne des utilisateurs.

Dans le prochain chapitre, nous allons principalement aborder les aspects techniques des différents composants du SPA que nous allons réaliser.

Chapitre 2

Composants de base d'un assistant personnel

2.1 Introduction

Dans ce chapitre, nous détaillerons un peu plus l'aspect technique d'une architecture typique d'un SPA (Speech Personal Assistant). Nous commencerons d'abord par définir des notions de base. Nous ne traiterons que celles qui ont été les plus utilisées dans les travaux que nous avons examiné. La suite du chapitre sera organisée en sections qui décrirons chacune le fonctionnement d'une partie du SPA, en citant les travaux et références qui relatent de cette dernière. Nous terminerons sur une conclusion qui introduira le chapitre suivant.

2.2 Schéma global d'un SPA

Durant nos lectures des différents travaux sur le domaine, nous avons pu dresser un schéma assez général qui englobe les principaux modules d'un SPA comme illustré par la figure 2.1.

Le processus peut être résumé en des étapes cruciales (qui seront détaillées dans les sections suivantes) :

- L'utilisateur énonce une requête en utilisant sa voix. Le signal est ensuite transformé en sa version textuelle.
- Étant sous un format textuel brut, la requête ne peut pas être interprétée ou comprise par la machine. Une représentation sémantique est générée qui regroupera les principales informations contenues dans la requête, à savoir l'intention de l'utilisateur et ses arguments.
- Au fur et à mesure que l'utilisateur énonce des requêtes, le système doit pouvoir être capable d'utiliser le contexte du dialogue pour interagir avec ce dernier afin

> "How can we build computer systems that automatically improve with experience, and what are the fundamental laws that govern all learning processes ?" (Mitchell, 2006)

que nous pouvons traduire par :

> "Comment pourrions nous développer un système informatique qui pourrait s'améliorer à travers l'expérience, et quelles seraient les lois qui régiraient le processus d'apprentissage ?"

D'après (Mitchell, 2006), l'apprentissage automatique est un paradigme qui stipule qu'un système informatique peut apprendre à effectuer un ensemble de tâches T, sachant qu'il dispose d'un ensemble de données E, tout en améliorant sa performance P.

Il existe plusieurs sous-catégories d'apprentissage automatique. Elles diffèrent principalement par la manière dont le système apprend, du type de données sur lesquelles il apprend. ainsi que du but de son apprentissage (classification, régression, etc.). Nous pouvons citer les catégories suivantes :

- **Apprentissage supervisé** : Les algorithmes de cette catégorie ont besoin d'une assistance externe. Les données doivent être séparées en deux parties (ensemble d'apprentissage et de test). Un *label* (ou classe) est associé à chaque instance pour permettre à l'algorithme de calculer un certain taux d'erreur qu'il essayera de minimiser au fur et à mesure qu'une nouvelle instance lui est présentée (Kotsiantis et al., 2006). Idéalement, le système pourra apprendre une certaine fonction $\hat{f} : X \to Y$ qui liera les entrées X aux sorties Y en minimisant l'erreur E_Y.
- **Apprentissage non supervisé** : Dans ce cas, les algorithmes ne disposent pas d'un étiquetage des données. Ils essayeront donc d'apprendre des *pattern* ou motifs fréquents pour grouper les données similaires. De tels algorithmes ne se préoccupent pas de la classe, mais de la similarité entre un groupe de données et un autre (Barlow, 1999).
- **Apprentissage par renforcement** : Cette dernière catégorie regroupe des algorithmes qui apprennent par un système de *trial and error* (Essais et erreur). C'est à dire qu'en interagissant avec l'environnement, l'agent apprenant (l'algorithme) apprendra à accomplir une tâche précise en exécutant des actions qui modifierons son état et celui de son environnement (voir la section 2.6.3.3).

Dans les sections suivantes, nous nous intéresserons à quelques types d'algorithme d'apprentissage automatique.

2.3.2 Réseaux de neurones artificiels

Un réseau de neurones artificiels est une structure d'appariement non-linéaire entre un ensemble de caractéristiques en entrée et sortie ; ces réseaux sont inspirés de la façon dont les systèmes nerveux biologiques fonctionnent. Ils permettent de modéliser les relations sous-jacentes des données. Ils sont composés d'un nombre arbitrairement large de petites unités de calcul interconnectées appelées neurones, qui traitent l'information d'une manière parallèle dans le but de résoudre un problème bien spécifique (Sonali, 2014). Ils ont notamment connu un très grand succès pendant les dernières années dans différents

2.3.2.2 Réseaux de neurones profonds

Les réseaux de neurones à une seule couche sont dits *shallow*. Historiquement, ils présentaient l'avantage d'être assez rapides durant la phase d'apprentissage. Cependant les limites computationnelles d'antan se sont vite vues brisées avec le développement de processeurs plus puissants. De plus, avec l'explosion des données sur internet, toutes les conditions nécessaires étaient réunies pour la mise en place d'architectures plus complexes. Les réseaux de neurones profonds sont une adaptation des réseaux multi-couches classiques avec généralement plus de 2 couches cachées.

Même si le principe reste le même, l'apprentissage profond est puissant car les architectures les plus complexes permettent d'extraire automatiquement les caractéristiques qui sont importantes mais non visibles ; c'est le cas des réseaux de neurones convolutifs et récurrents (LeCun et al., 2015).

2.3.2.3 Réseaux de neurones récurrents

Un aspect que les réseaux de neurones (profonds ou pas) ne peuvent capturer est la notion de séquentialité. En effet beaucoup de problèmes qui sont de nature séquentielle ne peuvent être modélisés par les architectures dites classiques, comme l'analyse d'un texte. L'introduction d'une notion de séquence permet donc de capturer des dépendances entre certains états et leurs voisins, on parle ici de contexte (Lipton, 2015).

Les réseaux de neurones récurrents (Recurrent Neural Networks, RNNs) sont des réseaux de neurones *feedforward* dont certaines connections en sortie sont réintroduites comme entrées dans le réseau durant une étape ultérieure du processus d'apprentissage (voir figure 2.3). Ceci introduit la notion de temps dans l'architecture. Ainsi à un instant t, un neurone récurrent recevra en entrée la donnée $x^{(t)}$ ainsi que la valeur de l'état caché $h^{(t-1)}$ résultante de l'étape précédente du réseau, la valeur en sortie $\hat{y}^{(t)}$ est calculée en fonction de l'état caché $h^{(t)}$. Les équations suivantes montrent les calculs effectués (Lipton, 2015) :

$$h^{(t)} = \tanh(W^{hx} \times x^{(t)} + W^{hh} \times h^{(t-1)} + b_h) \tag{2.2}$$

$$\hat{y}^{(t)} = softmax(W^{hy} \times h^{(t)} + b_y) \tag{2.3}$$

où W^{hx} est la matrice de poids entre la couche d'entrée et la couche cachée. W^{hh} est la matrice de poids de récurrence (c.à.d, c'est la matrice qui, quand multipliée par le vecteur d'état caché $t-1$, donnera le vecteur d'état t). W^{hy} est la matrice de poids entre la couche cachée et le couche de sortie. Les deux vecteurs b_h et b_y sont les vecteurs de biais et $\times$ est l'opération du produit matriciel (Lipton, 2015).

Un des principaux problèmes que rencontrent les RNNs est celui du "*Vanishing gradient*" traduit par le "*Problème de disparition du gradient*" (Hochreiter, 1998). Les relations à long terme entre les séquences ne sont donc pas capturées. Ainsi, pour remédier à ce problème, des architectures de réseaux de neurones dotées d'un module de mémoire ont étés introduites.

- i_t est appelé le vecteur d'entrée de la cellule LSTM.
- W_i est la matrice de poids entre les entrées (plus précieusement, l'état précédent et la nouvelle donnée) et le vecteur d'entrée de la cellule LSTM.
- b_i est un vecteur de biais.
- $\tilde{C}_t$ est le vecteur d'état interne candidat.
- W_C est la matrice de poids entre le vecteur d'entrée de la cellule LSTM et le vecteur d'état interne candidat.
- b_C est un vecteur de biais.
- $\tanh$ est la fonction Tangente Hyperbolique.

- **Porte d'oubli (Forget gate) :** De manière similaire, cette porte permet de spécifier au fur et à mesure de l'apprentissage les informations à oublier, qui sont donc peu importantes (Hochreiter and Schmidhuber, 1997 ; Lipton, 2015).

$$f_t = \sigma(W_f \times [h_{t-1}, x_t] + b_f) \tag{2.5}$$

où :

- f_t est appelé le vecteur d'oubli de la cellule LSTM.
- W_f est la matrice de poids entre les entrées et le vecteur d'oubli de la cellule LSTM.
- b_f est un vecteur de biais.

- **État interne de la cellule (Internal cell's state) :** C'est une sorte de convoyeur qui fait circuler l'information à travers la cellule. Cet état est mis à jour à travers la combinaison des vecteurs C_{t-1} et $\tilde{C}_t$ filtrés par les portes f_t et i_t (Hochreiter and Schmidhuber, 1997).

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{2.6}$$

où :

- C_{t-1} est le vecteur d'état interne de la cellule LSTM à l'instant $t-1$.

- **Porte de sortie (Output gate) :** Pour renvoyer un résultat comme état caché du réseau, la cellule filtre son vecteur d'état interne C_t et le combine avec la donnée en entrée et l'état du réseau précédent pour ne laisser passer que certaines informations (Olah, 2015 ; Hochreiter and Schmidhuber, 1997 ; Lipton, 2015).

$$\begin{aligned} o_t &= \sigma(W_o \times [h_{t-1}, x_t] + b_o) \\ h_t &= o_t * \tanh(C_t) \end{aligned} \tag{2.7}$$

où :

- o_t est un vecteur de sortie temporaire de la cellule LSTM à l'instant t.
- W_o est la matrice de poids entre les entrées et le vecteur de sortie temporaire de la cellule LSTM.
- b_o est un vecteur de biais.

pliquée pour engendrer un spectre de magnitude qui sera ensuite passé à un module de transformation en spectre de Mel (Mel-Spectrum) qui est une sorte de changement d'échelle. Finalement, une transformation inverse de fourrier est appliquée pour engendrer le Mel-Cpestrum qui est le vecteur d'attributs que nous recherchions (Narang and Gupta, 2015). Un tel processus peut utiliser plusieurs techniques pour la mise à l'échelle : MFCC (Mel-frequency cepstrum) (Chauhan and Desai, 2014), LPC (Linear Predictive coding) (O'Shaughnessy, 1988) et RASTA (RelAtive SpecTrAl) (Rahali et al., 2014), chacune possédant ses avantages et ses inconvénients.

2.4.2 Modélisation acoustique et modélisation du lexique

C'est le cœur du système de reconnaissance. Le but est de construire un modèle permettant d'identifier une séquence de phonèmes à partir d'une séquence d'observations de vecteurs d'attributs. Ceci peut être fait en utilisant un modèle HMM et en disposant d'un corpus de fichiers audio accompagnés de leurs transcriptions en phonèmes et en texte (Ghai and Singh, 2012 ; Rabiner and Juang, 1986), ou bien un réseau de neurones profonds (Yu and Deng, 2015), ou encore une hybridation de ces derniers (Deng et al., 2013). Le modèle acoustique procède après la reconnaissance de la séquence de phonèmes au décodage de ce dernier. Ceci est effectué en utilisant un dictionnaire linguistique qui transcrit chaque mot du vocabulaire en phonèmes qui le constituent. Ceci peut conduire à un cas d'ambiguïté où plusieurs mots peuvent avoir la même transcription phonétique (ou bien aucun mot ne correspond à cette séquence). Pour lever cette ambiguïté, un modèle de langue est nécessaire pour décider quelle est la séquence de mots la plus probable qui coïncide avec la séquence de phonèmes observée.

2.4.3 Modélisation de la langue

Cette étape consiste à modéliser les aspects linguistiques de la langue que le système essaye de traiter. Souvent ces aspects sont spécifiques au domaine d'application afin de restreindre l'espace de recherche des mots reconnaissables par le système, puis de déterminer la séquence de mots en sortie la plus plausible. Les modèles utilisés sont basés soit sur des grammaires à contexte libre dans le cas où les séquences de mots reconnaissables sont peu variées et peuvent être modélisées par le biais de règles de la langue (McCandless, 1994) ; ou alors dans un cadre plus récent, sur des modèles probabilistes basés sur les N-grammes.

Très souvent, quand il est nécessaire de traiter un ensemble de mots, phrases ou bien toute entité atomique qui constitue une séquence, il est intéressant de pouvoir assigner une probabilité de vraisemblance à une séquence donnée (par exemple une séquence de mots).

Dans un contexte textuel, un N-gramme est une suite de N mots [1] consécutifs qui forment une sous-chaîne S' d'une chaîne de caractères S. Un exemple d'un ensemble de bi-grammes

1. Dans certains cas, un N-gramme peut être aussi une suite de lettres/caractères

(2-grammes) pour la phrase "Ouvre le fichier *home*" serait donc : (Ouvre,le), (le,fichier), (fichier,*home*).

Ainsi, en disposant d'un corpus de textes assez large et diversifié et d'une méthode de comptage efficace, il serait possible de calculer la vraisemblance d'apparition d'un mot w après une certaine séquence de mots t sous forme d'une probabilité P (Jurafsky and Martin, 2008).

$$P(w|t) = \frac{Comptage(t, w)}{Comptage(t)} \tag{2.9}$$

Disposant d'un volume de données textuelles assez conséquent, l'utilisation de modèles basés sur les N-grammes a prouvé son utilité (Jurafsky and Martin, 2008 ; Yu and Deng, 2015 ; Roark et al., 2007) .

2.5 Compréhension du langage naturel

Après avoir obtenu la requête de l'utilisateur, le module qui prend le relais est celui de la compréhension du langage naturel (Natural Langage Understanding, NLU). Ce domaine fait partie du traitement automatique du langage naturel (Natural Langage Processing, NLP). Il se focalise principalement sur les tâches qui traitent le niveau sémantique, voire même pragmatique, de la langue tel que la classification de texte, l'analyse de sentiments ou encore le traitement automatiques des e-mails. Dans le contexte d'un SPA, le but final d'un module de compréhension du langage naturel est de construire une représentation sémantique de la requête de l'utilisateur. Étant donné une telle requête préalablement transformée en texte brut dans un langage naturel, le module essayera d'en extraire une information primordiale qui est l'intention du locuteur.

2.5.1 L'intention

L'intention est un élément clé dans notre travail. Nous allons donc la définir et expliquer pourquoi elle a été choisie comme abstraction sémantique d'une requête d'un utilisateur.

Une intention (ou intent en anglais) est une représentation sémantique d'un but ou d'un sous-but de l'utilisateur. Plusieurs requêtes de l'utilisateur peuvent avoir le même intent, ce qui rend l'utilisation d'une telle représentation essentielle pour que la machine puisse mieux comprendre l'utilisateur. La motivation vient du fait que la machine ne peut pas comprendre une requête formulée dans un langage non-formel. Il a fallu trouver un moyen de communication universel entre la machine et l'utilisateur de telle sorte que l'un puisse "comprendre" l'autre . L'intermédiaire entre ces deux parties est un traducteur bidirectionnel d'une requête en langage naturel vers une requête dans un langage formel et inversement.

Le choix de l'intention doit être minutieusement étudié au préalable pour ne pas trop subdiviser un type d'intention, tout en gardant un certain degré de spécificité. À titre d'exemple, si deux intentions $Ouvrir\_fichier$ et $Ouvrir\_document$ existent, on retrouve ici

2.6.2.3 Suivi de l'état du gestionnaire de dialogue avec réseaux de neurones profonds

Récemment, des approches utilisant les réseaux de neurones profonds (voir 2.3.2.2) ont fait leur apparition. En effet, l'utilisation des architectures profondes permet de capter des relations complexes entre les caractéristiques d'un dialogue et, ainsi, mieux estimer l'état du système. Le réseau de neurones estime les probabilités de toutes les valeurs possibles d'un emplacement de la trame sémantique (Henderson et al., 2013). En conséquence, il peut être utilisé comme modèle de suivi d'état pour un processus partiellement observable.

2.6.3 Politique de gestion de dialogue

La première partie est dédiée au module qui suit l'état du système de dialogue. Dans cette partie, nous allons présenter des approches proposées afin d'arriver au but du MDP, c'est à dire quelles actions prendre pour maximiser la somme des récompenses obtenues.

2.6.3.1 Gestion de dialogue avec une base de règles

Les premières approches utilisaient des systèmes de règles destinés à un domaine bien spécifique. Elles étaient déployées dans plusieurs domaines d'application pour leur simplicité. Cependant, le travail manuel nécessaire reste difficile à faire et, généralement, n'aboutit pas à des résultats flexibles qui peuvent suivre le flux du dialogue convenablement (Lee et al., 2010).

2.6.3.2 Gestion de dialogue par apprentissage

La résolution d'un MDP revient à trouver une estimation de la fonction de récompense afin de pouvoir choisir la meilleure action. La majorité des approches récentes utilise l'apprentissage par renforcement dans le but d'estimer la récompense obtenue par une action et un état donnés. Cette préférence par rapport aux approches supervisées revient à la difficulté de produire des corpus de dialogues (Henderson et al., 2008), encore moins des corpus annotés avec les récompenses à chaque transition. Néanmoins, il existe des approches de bout en bout qui exploitent des architectures avec réseaux de neurones profonds et traitent le problème comme Seq2Seq [2] (voir figure 2.16) afin de produire directement une sortie à partir des informations reçues de l'utilisateur (Wen et al., 2017 ; Serban et al., 2016).

2.6.3.3 Apprentissage par renforcement

L'apprentissage par renforcement est une approche qui a pour but d'estimer une politique d'actions à prendre dans un environnement donné. Cette politique doit maximiser

2. Les modèles Seq2Seq sont des architectures de réseaux de neurones profonds qui prennent en entrée une séquence et produisent en sortie une autre séquence en utilisant les réseaux de neurones récurents (RNN).

cette approche, à chaque fois que l'agent prend une action, il compare la récompense reçue avec celle estimée par le réseau de neurones, et applique ensuite l'algorithme de rétro-propagation pour corriger les erreurs du réseau.

Le système de dialogue est modélisé par un MDP. Seulement, ce dernier inclut l'utilisateur comme partie de l'environnement. Par conséquent, pour appliquer l'apprentissage par renforcement, il est nécessaire qu'un utilisateur communique avec le système pour qu'il apprenne. D'où la nécessité d'utiliser un **simulateur d'utilisateur** qui est un programme capable de se comporter comme un humain interagissant avec un système de dialogue. Le simulateur doit pouvoir estimer la satisfaction de l'utilisateur après une interaction. En d'autres termes, il doit comporter une fonction de récompense. Il existe plusieurs méthodes pour créer un simulateur d'utilisateur :

- Les simulateurs d'utilisateur basés règles : dans ce cas une liste de règles est écrite manuellement et le simulateur doit les suivre pour communiquer avec le système (Schatzmann et al., 2007).
- Les simulateurs d'utilisateur basés n-grammes : ils traitent un dialogue comme une séquence d'actions en prenant les n-1 actions précédentes pour estimer l'action la plus probable que peut prendre le simulateur à partir des statistiques tirées d'un corpus de dialogues (Georgila et al., 2005).
- Les simulateurs d'utilisateur basés HMM : les états du modèle sont les états du système et les observations sont ses actions. Ainsi un HMM peut estimer l'état le plus probable du système pour prendre une action. Il existe d'autres variantes, IHMM et IOHMM, qui incluent les probabilités conditionnelles des actions de l'utilisateur sachant l'état du système ou l'action du système directement dans le modèle HMM (Cuayáhuitl et al., 2005).
- Les simulateurs d'utilisateur avec apprentissage par renforcement : Comme le gestionnaire de dialogue, le simulateur apprend par renforcement au même temps. Dans ce cas la fonction de récompense pour les deux agents peut être apprise à partir des dialogues humain-humain (Chandramohan et al., 2011).

2.7 Génération du langage naturel (Natural Language Generation, NLG)

Le domaine de la génération automatique du langage naturel est l'un des domaines dont les limites sont difficiles à définir (Evans et al., 2002). Il est vrai que la sortie d'un tel système est clairement du texte. Cependant, l'ambiguïté se trouve dans ses entrées, c'est-à-dire, sur quoi se basera le système pour générer le texte. D'après (Reiter and Dale, 1997), la génération du langage naturel est décrite comme étant le sous domaine de l'intelligence artificielle qui traite la construction des systèmes de génération de texte à partir d'une représentation non-linguistique de l'information. Celle-ci peut être une représentation sémantique, des données numériques, une base de connaissances ou même des données visuelles (images ou vidéos). Ceci dit, d'autres travaux, comme (Labbé and Portet, 2012),

utilisent les mêmes techniques pour des entrées linguistiques. Enfin, la génération du langage naturel peut être très proche de la gestion de dialogue (Dethlefs, 2014). En effet, le texte généré doit prendre en compte l'historique de la conversation et le contexte de l'utilisateur.
Il existe six tâches trouvées fréquemment dans les systèmes de génération de texte (Reiter and Dale, 1997) ; nous les présentons dans cette section.

2.7.1 Détermination du contenu

Cette partie consiste à sélectionner les informations de l'entrée dont le système veut transmettre leur contenu sous forme de texte naturel à l'utilisateur. En effet, les données en entrée peuvent contenir plus d'informations que ce que l'on désire communiquer (Yu et al., 2007). De plus, le choix de l'information peut aussi dépendre de l'utilisateur et de ses connaissances (Dethlefs, 2014). Ceci requiert de mettre au point un système qui détecte les informations pertinentes à l'utilisateur.

2.7.2 Structuration de texte

Après la détermination du contenu, le système doit ordonner les informations à transmettre. Ceci dépend grandement du domaine d'application qui peut exiger des contraintes d'ordre temporel ou de préférence par importance des idées. Les informations à transmettre en elles-mêmes sont souvent reliées par sens, ce qui implique une certaine structuration de texte à respecter.

2.7.3 Agrégation de phrases

Certaines informations peuvent être transmises dans une même phrase. Cette partie introduit des notions de la linguistique afin que le texte généré soit plus lisible et évite les répétitions. Un exemple de cela peut être la description de la météo à Alger au cours de la matinée :

- Il va faire 16° à Alger à 7h.
- Il va faire 17° à Alger à 8h.
- Il va faire 18° à Alger à 9h.
- Il va faire 18° à Alger à 10h.

Ceci peut être agrégé en un texte plus compact : "La température moyenne à Alger sera de 17.25°entre 7h et 10h."

2.7.4 Lexicalisation

Le système choisit les mots et les expressions à utiliser pour communiquer le contenu des phrases sélectionnées. La difficulté de cette tâche revient à l'existence de plusieurs

manières d'exprimer la même idée. Cependant, certains mots ou expressions sont plus appropriés en certaines situations que d'autres. En effet, "inscrire un but" est une façon inadéquate d'exprimer "un but contre son camp" (Gatt and Krahmer, 2018).

2.7.5 Génération d'expressions référentielles (REG)

Cette partie du système se focalise sur la génération d'expressions référentielles qui peuvent être entre autres : des noms propres, groupes nominaux ou pronoms et ceci a pour but d'identifier les entités du domaine. Cette tâche semble être très proche de la lexicalisation dans le sens où elle aussi a pour but de choisir les mots et les expressions ; elle s'avère néanmoins plus délicate dû à la difficulté de confier suffisamment d'information sur l'entité afin de la différencier des autres (Reiter and Dale, 1997). Le système doit faire un choix de l'expression référentielle en se basant sur plusieurs facteurs. Par exemple "Mohammed", "Le professeur" ou "Il" faisant référence à la même personne, le choix entre eux dépend de comment l'entité a été mentionnée auparavant, si elle l'a été, et des détails qui l'ont accompagnée.

2.7.6 Réalisation linguistique

La dernière tâche consiste à combiner les mots et expressions sélectionnés pour construire une phrase linguistiquement correcte. Ceci requiert l'utilisation des bonnes formes morphologiques des mots, les ordonner, et éventuellement l'addition de certains mots du langage afin de réaliser une structure de phrase grammaticalement et sémantiquement correcte. Plusieurs méthodes ont été proposées, principalement les méthodes basées sur des règles manuellement construites (modèles de phrases, systèmes basés grammaires) et des approches statistiques (Gatt and Krahmer, 2018).

Modèles de phrases : La réalisation se fait en utilisant des modèles de phrases prédéfinis. Il suffit de remplacer des espaces réservés par certaines entrées du système. Par exemple, une application dans un contexte météorologique pourrait utiliser le modèle suivant : la température à [ville] atteint [température]° le [date].

Cette méthode est utilisée lorsque les variations des sorties de l'application sont minimales. Son utilisation a l'avantage et l'inconvénient d'être rigide. D'un côté, il est facile de contrôler la qualité des sorties syntaxiquement et sémantiquement tout en utilisant des règles de remplissage complexes (Theune et al., 2001). Cependant, lorsque le domaine d'application présente beaucoup d'incertitude, cette méthode exige un travail manuel énorme, voire impossible à faire, pour réaliser une tâche pareille. Bien que certains travaux ont essayé de faire un apprentissage de modèles de phrases à partir d'un corpus (Angeli et al., 2012), cette méthode reste inefficace lorsqu'il s'agit d'applications qui nécessitent un grand nombre de variations linguistiques.

Systèmes basés grammaire : La réalisation peut se faire en suivant une grammaire du langage. Celle-ci contient les règles morphologiques et de structures de la langue, no-

Chapitre 3

Conception du système

3.1 Introduction

Dans ce chapitre, nous allons présenter en détail les étapes de conception de notre système no. De prime abord une architecture générale est introduite puis décortiquée. Ensuite, chaque module du système sera détaillé du point de vue des composants qui le constituent. Une conclusion viendra ensuite clôturer ce chapitre.

3.2 Architecture du système

Comme montré dans la figure 3.1 et comme cité dans le chapitre précédent (voir 2.2) le système Speact se présente comme l'interconnexion de cinq parties dont une interface [1] et quatre modules internes communiquant entre eux. Chaque module forme ainsi un maillon d'une chaîne qui représente une partie du cycle de vie du système. L'architecture de Speact est un pipeline (chaîne de traitement) de processus qui s'exécutent de manière indépendante mais qui font circuler un flux de données entre eux dans un format préalablement établi (voir 2.1). Nous pouvons séparer ces parties en deux catégories : la partie utilisateur et la partie interne du système que nous présentons dans la figure 3.1.

3.2.1 Partie utilisateur

Cette partie représente ce que l'utilisateur peut voir comme entrée/sortie et les interfaces qui lui sont accessibles. Puisque l'assistant est un processus qui communique majoritairement avec l'utilisateur à travers des échanges verbaux, nous avons pensé à implémenter l'interface du système comme un processus qui s'exécute en arrière plan et qui attend d'être activé, dans notre cas par un événement physique, c.à.d un clic sur un bouton/icône ou raccourci clavier. L'assistant pourra ensuite répondre en affichant un texte à l'écran qui sera vocalement synthétisé et envoyé à l'utilisateur via l'interface de sortie de

1. Par interface nous entendons le sens abstrait du terme et non obligatoirement le sens interface graphique.

Chaque batch (lot) de données reçu est alors manuellement validé par l'équipe de Mozilla pour l'inclure dans la banque de données d'exemples principale. À ce jour, pour la langue anglaise, la plateforme a récolté plus de 22Go de données, soit 803 heures d'enregistrements correspondant à plus de 30 000 voix différentes dont 582 heures ont été validées. Cependant, ce volume de données est relativement petit comparé à celui déjà utilisé pour l'apprentissage initial. En effet, plusieurs sources ont été combinées pour construire cet ensemble de données. Dans (Hannun et al., 2014), il a été mentionné que trois ensembles d'apprentissage existants ont été choisis dont WSJ (Wall Street Journal)[4], Switchboard[5] et Fisher[6], qui à eux trois cumulent 2380 heures d'enregistrements audios en anglais et plus de 27 000 voix différentes. Vient s'ajouter à cela, l'ensemble Baidu[7] avec 5000 heures d'enregistrements et 9600 locuteurs.

3.3.3 Modèle de la langue

Type du modèle

C'est un modèle basé sur les N-grammes, 3-grammes pour être plus précis, qui est utilisé pour comme modèle de langue. Il permet de façon assez simple et intuitive de capturer l'enchaînement des mots dans une langue donnée, rendant ainsi la transcription finale assez proche de la façon dont les mots sont distribués dans le corpus d'apprentissage.

Données d'apprentissage

À l'origine, DeepSpeech utilise un modèle de langue dont la source n'est pas dévoilée par les chercheurs dans (Hannun et al., 2014). Mais, son volume est approximativement de 220 millions de phrases avec 495 000 mots différents. Cependant, puisque ce corpus nous reste inconnu et qu'il a probablement été construit pour reconnaître des séquence de mots en anglais assez générales, nous avons décidé de construire notre propre modèle de langue en récoltant des données depuis des dépôts sur le site Github, plus précisément les fichiers README.md des dépôts qui font office de manuels d'utilisation d'un projet hébergé sur le site. Ce type de fichiers renferme généralement des instructions de manipulation de fichiers, de lancement de commandes, etc ; ce qui offre un bon corpus pour le modèle de langue. En effet, notre système se concentre plus sur l'aspect de manipulation d'un ordinateur, donc la probabilité de trouver certaines séquences de mots qui appartiennent au domaine technique est en théorie plus élevée. La procédure suivie, qui est illustrée dans la figure 3.4, est la suivante :

- L'acquisition des données dans leur format brut **.md** (markdown) se fait de deux manières :
    - Depuis le site officiel de GitHub en faisant des requêtes http au serveur en suivant le patron suivant pour les urls :

4. `http://www.cstr.ed.ac.uk/corpora/MC-WSJ-AV/`
5. `https://catalog.ldc.upenn.edu/LDC97S62`
6. `https://catalog.ldc.upenn.edu/LDC2004S13`
7. `https://ai.baidu.com/broad/introduction`

- **id** : un entier qui sert d'identificateur pour l'instance.
- **intent** : l'intention (non encore codifiée) attribuée à l'instance.
- **tags** : les étiquettes de chaque mot de la requête. Une étiquette peut être soit NUL (ce n'est pas un argument) ou bien le nom de l'entité (argument) que représente le mot à la position étiquetée. La liste complète des intentions avec leurs arguments se trouve dans le tableau 4.2 du chapitre "Réalisation et résultats".
- **postags** : la liste des étiquettes morphosyntaxiques de chaque mots de la requête. L'ensemble des étiquettes utilisées est celui du Penn Treebank (Marcus et al., 1994). Dans l'exemple ci-dessus, NN signifie "Nom au singulier". NNS signifie "Nom au pluriel". VB et VBD signifient respectivement "Verbe à l'infinitif" et "Verbe au passé simple". Enfin, DT signifie "Déterminant".
- **text** : le texte de la requête nettoyée et dont les mots sont séparés uniquement par un espace.

3.5 Module de gestion du dialogue

Le but de ce module est de décider quelle action prendre à chaque instant du dialogue. De prime abord, nous allons présenter l'architecture globale de ce module notamment la représentation des informations reçues et la politique d'actions. Ensuite, nous allons détailler la conception de chaque partie.

3.5.1 Architecture du module

Comme nous l'avons déjà vu, l'architecture typique des gestionnaires de dialogue se compose de deux parties principales :

- Un module qui suit l'état du dialogue : Pour gérer le dialogue avec l'utilisateur, le gestionnaire doit représenter l'état du dialogue de façon à pouvoir répondre aux actions de l'utilisateur. Ce module sert à suivre cet état après chaque étape du dialogue.
- Une politique d'actions : Celle-ci détermine l'action à prendre à partir d'un état donné.

3.5.1.1 État du dialogue

Avant de détailler les deux modules du gestionnaire, il est nécessaire d'introduire une représentation de l'état du dialogue. Classiquement, les trames sémantiques ont été utilisées dans ce but (voir 2.6.2). Le suivi d'état se fait, dans ce cas, en gardant trace des emplacements remplis durant le dialogue. Nous avons opté pour une représentation plus riche par l'utilisation des graphes de connaissances, qui sont une forme de représentation où les connaissances sont décrites sous forme d'un graphe orienté étiqueté. Des travaux ont déjà utilisé ce type de connaissances (Stoyanchev and Johnston, 2018) et des ontologies (Wessel et al., 2019) pour représenter l'état du système de dialogue. À partir de ce dernier, une base de règles décide quelle action prendre directement du graphe. L'avantage

le nombre de fichiers qu'on peut supprimer ou le nombre de répertoires auxquels on peut accéder ne sont pas fixes.

Le simulateur commence d'abord par générer une arborescence aléatoire qui représente la situation initiale du système. Ensuite, il duplique cette arborescence en y introduisant des modifications pour générer une arborescence but. Enfin, le simulateur essaye de guider l'agent pour arriver au but en utilisant les actions utilisateurs possibles.

En plus des actions de création et suppression de fichiers ainsi que les changements de répertoires qui peuvent guider l'agent au but, d'autres sous-buts peuvent être créés suivant une distribution de probabilité comme copier ou changer l'emplacement d'un fichier, renommer un fichier, ouvrir un fichier, etc. Dans ce cas, le simulateur donne la priorité aux sous-buts avant de reprendre les actions menant au but final.

L'algorithme suivi par le simulateur est le suivant :

Algorithm 1 Algorithme simulateur

```
1 : procedure Step(entrées : action_agent, max_tour, tour ; sorties : recompense, fin, succes, tour)
2 :     tour ← tour + 1
3 :     if tour > max_tour then
4 :         fin ← true
5 :         succes ← echec
6 :         reponse_utilisateur ← réponse_fin()
7 :     else
8 :         succes ← maj_état(action_agent)
9 :         if succes then
10 :            fin ← true
11 :            reponse_utilisateur ← réponse_fin()
12 :        else
13 :            reponse_utilisateur ← décider_action(action_agent)
14 :    recompense ←calculer_récompense(action_agent, succes)
15 : return reponse_utilisateur
```

- **ligne 1 :** en entrée de la procédure :
  - $action\_agent$: l'action pour laquelle l'agent attend une réponse du simulateur.
  - max_tour : le nombre maximum d'échanges agent-simulateur.
  - $tour$: le numéro de l'échange actuel.

  en sortie de la procédure :
  - $recompense$: la récompense que donne le simulateur suite à la dernière action de l'agent.
  - fin : un booléen qui détermine si la discussion est terminée.
  - $succes$: un booléen qui détermine si l'agent est arrivé à un succès.
  - $tour$: la variable $tour$ est modifiée à l'intérieur de la procédure ; elle doit être donc mise en sortie aussi.

Le tableau 3.1 associe les changements avec leurs récompenses :

Événements	**Récompenses**
Similarité améliorée	2
Succès	2
Sous-but réalisé	2
Similarité diminuée	-3
Confirmation d'une question	0
Autre	-1

Table 3.1 – *Tableau des récompenses*

- Nous avons choisi de donner une récompense unitaire négative pour chaque échange qui n'affecte pas l'arborescence de fichiers. La récompense négative permet d'éviter que le dialogue dure longtemps. L'agent essaye donc de minimiser ce type d'actions afin de ne pas cumuler les récompense négatives.
- **Confirmation d'une question :** la seule exception à cela c'est quand l'utilisateur confirme une action à l'agent. La récompense est nulle (= 0) pour que l'agent puisse demander une confirmation quand il n'est pas sûr de ce qu'il doit faire sans diminuer le cumul des récompenses reçues.
- **similarité améliorée :** On calcule la similarité entre l'arborescence courante et l'arborescence but. La valeur de la similarité est égale à n_sim/n_diff avec :
  - n_sim : nombre de fichiers qui existent dans les deux arborescences.
  - n_diff : nombre de fichiers qui n'existent que dans une des deux arborescences.

  Cette valeur change soit en ajoutant ou en supprimant un fichier. Dans le cas où un fichier est ajouté, si celui-ci existe dans l'arborescence but, n_sim est incrémentée et n_diff reste fixe ; la valeur de la similarité augmente. Sinon, s'il n'existe pas dans l'arborescence but, n_diff est incrémentée et n_sim reste fixe ; la valeur de la similarité diminue. Alternativement, si un fichier est supprimé, et celui-ci n'existe pas dans l'arborescence but, n_diff est décrémentée et n_sim reste fixe ; la valeur de la similarité augmente. Sinon s'il existe dans l'arborescence but, n_sim est décrémentée et n_diff reste fixe ; dans ce cas la valeur de la similarité diminue. En conséquence, si cette valeur s'améliore, on donne à l'agent une récompense positive supérieure en valeur absolue à celle donnée dans les échanges d'informations.
- **Succès :** c'est-à-dire, le but final du simulateur est réalisé. Pour arriver à cet événement, soit l'agent ajoute un fichier à l'arborescence ou bien il le supprime. Les autres actions, comme renommer un fichier ou lui changer d'emplacement, font partie des sous-buts. L'agent ne fait donc qu'améliorer la similarité en ajoutant ou en supprimant un fichier ; c'est pourquoi cet évènement et celui qui l'a précédé ont la même valeur de récompense.
- **Sous-but réalisé :** c'est la récompense donnée quand l'agent arrive au sous-but du simulateur. Par exemple, en ouvrant un fichier, en le renommant ou en le copiant dans un autre répertoire, etc. Elle est égale à celle donnée dans le cas où la similarité est améliorée car l'exécution de l'action menant à ce sous-but ne nécessite pas plus d'efforts ou d'échanges que quand le simulateur ajoute ou supprime un fichier.

- **Similarité diminuée :** la récompense est dans ce cas négative et supérieure en valeur absolue à celle que l'agent obtient lorsqu'il améliore la similarité. Ce choix a pour but d'éviter que l'agent boucle sur des actions dont la somme des récompenses est supérieure ou égale à zero. Par exemple, il peut créer ensuite supprimer le même fichier. Si la somme de ces deux actions est supérieure ou égale à zéro, l'agent peut boucler sur ces actions indéfiniment sans pour autant recevoir des récompenses négatives.

Pour résumer le fonctionnement du simulateur, à l'arrivée d'une nouvelle action agent, celle-ci met à jour l'état du simulateur. L'état du simulateur se compose de deux parties : un simulateur d'arborescence de fichiers qui simule l'état de l'arborescence courante, et des variables d'état qui contiennent d'autres informations comme l'état des sous-buts, le fichier en cours de traitement, les informations reçues de l'agent, le répertoire courant, etc. Après la mise à jour de l'état, si l'action de l'agent nécessite une réponse immédiate, comme la demande d'une information ou la permission d'exécuter une action, celle-ci est traitée directement, sinon le simulateur initie le traitement d'une nouvelle sous-tâche. C'est-à-dire, si un but intermédiaire existe, une action qui le traite est générée ; sinon, une action qui traite le but final est générée.

3.5.4 Modèles d'apprentissage

Comme on l'a déjà cité dans le chapitre précédent, il existe plusieurs algorithmes d'apprentissage par renforcement comme Q-Learning ou State-Action- Reward-State-Action (SARSA) (Rummery and Niranjan, 1994). Cependant, ces algorithmes, en essayant d'estimer la fonction Q de récompense, traitent le problème comme un tableau état/action et essayent d'estimer pour chaque état et action la récompense résultante. Ceci implique que ces algorithmes ne peuvent pas estimer la fonction de récompense pour des états qu'ils n'ont pas vus pendant l'apprentissage. Pour pallier à ce problème, Deep Q Learning (DQL)(Mnih et al., 2015) utilise un réseau de neurones comme estimateur de la fonction Q, ce qui lui permet d'avoir une notion de similarité entre les états. Ainsi, il peut estimer la récompense pour des états jamais vus auparavant.

Encodeur de graphe

La flexibilité des graphes les rend difficiles à introduire dans un réseau de neurones vu que ce dernier n'accepte que des entrées de tailles fixes. Des méthodes ont été utilisées pour introduire les graphes dans des réseaux de neurones notamment les convolutions sur les graphes avec Graph Convolution Networks (GCN)(Kipf and Welling, 2017) qui s'avèrent être des variantes des Gated Graph Neural Networks (GGNN)(Li et al., 2016). Ces derniers utilisent des réseaux de neurones récurrents (RNNs) entre chaque deux nœuds reliés par un arc pour transférer l'information d'un nœud à un autre. C'est-à-dire que le RNN prend en entrée les vecteurs encodant un nœud et un de ses arc pour essayer de propager l'information contenue dans ces vecteurs au nœud destination. Ce dernier met à jour le vecteur l'encodant en sommant les résultats du RNN provenant des différents nœuds voisins. Ce qui résulte en des vecteurs ayant un encodage qui tient en compte le voisinage du noeud. Cette

3.7 Conclusion

Dans ce chapitre, et en nous inspirant des travaux existants, nous avons pu modéliser et conceptualiser tout les aspects du système que nous jugeons assez conforme au standard. De plus, des améliorations ont été proposées pour enrichir le système de base retrouvé dans la littérature ainsi que pour faciliter l'extensibilité des fonctionnalités de bases avec un moindre effort d'intégration. Ceci est le résultat d'une conception modulaire et facilement maintenable.

Dans le chapitre suivant, nous allons passer à l'implémentation des différents modules, au développement d'une application dédiée et à une partie expérimentation pour tester les approches proposées. Nous discuterons leurs améliorations et les comparerons avec des standards existants.

Chapitre 4

Réalisation et résultats

4.1 Introduction

Dans la partie conception, nous avons proposé certaines modifications à apporter sur des approches existantes ainsi que des méthodes que nous avons jugées intéressantes pour notre système. Nous allons, dans la suite de ce chapitre, montrer la faisabilité de ces méthodes ainsi que leurs avantages et leurs limites.

Nous commençons par décrire l'environnement de travail. Nous détaillerons par la suite les aspects de l'implémentation des différents modules de notre application qui seront évalués et analysés. Pour clôturer avec une application d'un assistant personnel pour la manipulation de fichiers.

4.2 Environnement de développement

Dans cette section, nous allons présenter les différents outils (logiciels et matériels) qui ont été utilisés pour l'implémentation de Speact.

4.2.1 Machines utilisées

Principalement, le développement se divise en deux parties :

- Apprentissage : les données sont récoltées ou construites puis nettoyées et préparées. Les modèles sont développés, entraînés puis testés.
- Les modules sont implémentés puis connectés et intégrés dans l'application.

Pour ce, faire nous avons utilisé des machines dont les spécificités sont mentionnées dans la figure 4.1 ainsi qu'un ensemble de bibliothèques et librairies dont une partie est présentée dans la figure 4.2.

PySpark

PySpark[6] est un package Python utilisé comme interface pour intéragir avec un serveur Spark. Il permet entre autres de lire et écrire des données dans le nouveau format Hadoop.

Scikit-Learn

Scikit-Learn[7] est une bibliothèque Open source conçue pour rapidement développer des modèles pour l'apprentissage automatique, principalement utilisée pour ses nombreux outils de pré-traitement des données (codification, normalisation, filtrage, etc.).

Numpy

Numpy[8] est une bibliothèque spécialisée dans la manipulation de grands volumes de données numériques, notamment les vecteurs (tableaux) multi-dimensionnels. Les opérations sur ces derniers sont implémentées en C pour optimiser au maximum leur coût en temps de calcul ou ressources mémoires utilisées. Elle offre des structures de données compatibles avec beaucoup d'autre librairies comme Tensorflow ou Keras.

Tensorflow & Keras

Tensorflow[9] est une bibliothèque dédiée à l'apprentissage automatique, et plus particulièrement aux réseaux de neurones et l'apprentissage profond. Optimisée pour exécuter des opérations à grande échelle et massivement distribuées sur un réseau, Tensorflow offre la possibilité d'implémenter une grande variété d'architectures de modèles avec un maximum d'efficacité. Elle dispose d'un package Python permettant d'interagir avec le cœur de la bibliothèque mais reste néanmoins assez bas-niveau. Keras[10] quant-à elle propose de rajouter une couche d'abstraction à Tensorflow. C'est un package python destiné à faciliter le développement de modèles pour l'apprentissage profond tout en offrant la possibilité de rajouter et modifier un grand nombre de fonctionnalités par défaut. Sa force réside dans le fait qu'il peut utiliser au plus bas niveau plusieurs librairies autres que Tensorflow comme Theano[11] et CNTK[12].

Flask

Falsk[13] est une micro-librairie Open source dédiée au développement d'applications basées web. De base, cette librairie est très légère, mais elle offre la possibilité d'ajouter des

6. https://spark.apache.org/
7. https://scikit-learn.org/stable/
8. https://www.numpy.org/
9. https://www.tensorflow.org/
10. https://keras.io/
11. http://deeplearning.net/software/theano/
12. https://github.com/microsoft/CNTK
13. http://flask.pocoo.org/

extensions qui s'intègrent très facilement au système de base.

Vuetify

Vuetify est une librairie Open source basée sur VueJs dédié au développement d'interfaces web ou mobiles. Elle implémente le paradigme Material Design de Google et offre la possibilité d'étendre les composants de base et de créer des interfaces belles et adaptatives.

4.2.4 Outils et logiciels de développement

PyCharm

PyCharm est un environnement de développement intégré spécialisé et optimisé pour programmer dans le langage Python. Il permet l'analyse de code en continu et offre un débogueur intégré pour exécuter un code instruction par instruction. Il offre également l'intégration de logiciel de gestion de versions comme Git, et supporte le développement web avec Flask.

Git

Système décentralisé de gestion de versions. Il permet entre autres de gérer les différentes versions d'un projet durant son développement, mais aussi de garder l'historique des modifications effectuées ainsi que la possibilité de régler des conflits lors de l'intégration finale des contributions des développeurs.

Google Colaboratory

Colaboratory [14] est un outil de recherche et développement pour la formation et la recherche associées à l'apprentissage profond. C'est un environnement Python qui ne nécessite aucune configuration et offre la possibilité d'utiliser de puissantes machines rendues accessibles par Google pour accélérer la phase d'apprentissage.

Protégé

Protégé est un système dédié à la création et la modification d'ontologies. Il est développé en Java et est Open source distribué sous une licence libre (la Mozilla Public License). Il se démarque par le fait qu'il permet de travailler sur des ontologies de très grandes dimensions.

14. https://colab.research.google.com/

4.3 Reconnaissance automatique de la parole

Pour ce premier module, il a été très difficile d'effectuer les tests idéaux. En effet, nous n'avons pas pu trouver un ensemble de données qui proposait du contenu en rapport avec Speact. Néanmoins, nous avons pu construire un mini-ensemble pour tester l'apport de notre modèle de langue. Par contre, Les résultats ne doivent pas être pris comme une référence absolue, mais plutôt comme une indication, ou un point de départ, pour de possibles futurs tests.

4.3.1 Ensemble de test

Pour tester le modèle acoustique, les données récoltées à travers le projet CommonVoice (voir la section 3.3.2) constituent un assez bon échantillon, de par la nature des enregistrements (sur téléphone portable, par plusieurs genres et accents de locuteurs ...), mais aussi de par le volume (environs 500 heures d'enregistrements audios). Nous avons toutefois décidé de construire un mini-ensemble de test d'environ 204 enregistrements audios d'une longueur moyenne de 5 secondes chacun. Ces échantillons ont été prélevés sur trois locuteurs masculins. 20% de ces échantillons ont été prélevés dans un environnement fermé mais bruité (il s'agit d'un espace de travail pour étudiants) et sur téléphone. Le reste a été prélevé dans un environnement fermé avec peu de bruit à partir du micro d'un ordinateur portable. Chaque enregistrement est soit :

- une requête prélevée de l'ensemble de test du module de compréhension du langage naturel (voir les sections 4.4.2 et 3.4.3), ou
- une question prélevée de l'ensemble de données AskUbuntu [15] qui regroupe des questions relatives à la manipulation d'un ordinateur sous le système d'exploitation GNU/-Linux.

Pour le modèle de langue, il s'agit de celui mentionné dans la section 2.4.3. Quelques modifications ont été rajoutées comme le filtrage des mots qui n'appartiennent pas à la langue anglaise, mais au prix du sacrifice de quelques noms propres non reconnus ou bien de séquences de mots/lettres sans réel sens.

4.3.2 Méthodologie d'évaluation

Les tests ont été effectués dans un serveur Colab pour libérer les machines locales. Les principales étapes sont les suivantes :

1. **Préparation des données** : Les données sont prélevées d'une base de données sqlite qui comprend une seule table Transcriptions. Les colonnes de la table sont :
   - **id** : identifiant de l'enregistrement
   - **path** : chemin vers le fichier audio de la requête.
   - **text** : transcription textuelle de la requête.

15. https://github.com/taolei87/askubuntu

Un script de conversion est ensuite lancé pour s'assurer que chaque enregistrement est au format .wav avec une fréquence de rafraîchissement égale à 16KHz, le modèle acoustique de DeepSpeech attend cette valeur exacte pour lancer l'inférence, sinon une erreur se produira.

2. **Métriques retenues** : À chaque instance testée, le **WER** (Word Error Rate) est calculé. Pour rappel la formule du WER est la suivante :

$$WER(y, \hat{y}) = \frac{S + D + I}{S + D + C} = \frac{S + D + I}{N}$$

où :

- $\hat{y}$ est la séquence de mots prédite appelée Hypothèse.
- y est la séquence de mots réelle appelée Référence.
- S est le nombre de substitutions (compté en mots) réalisées entre l'hypothèse et la référence.
- D est le nombre de suppressions qu'a effectué le système, donc le nombre de mots supprimés dans l'hypothèse par rapport à la référence.
- I est le nombre d'insertions effectuées par le système (c.à.d. le nombre de mots rajoutés à l'hypothèse par rapport à la référence).
- C est le nombre de mots bien placés.
- N est la longueur totale de la séquence en nombre de mots

3. **Boucle d'évaluation** : L'opération précédente est réitérée en incrémentant à chaque fois le taux d'utilisation de notre modèle de langue (de 20% à 100% avec un pas de 10%). Le WER associé est ensuite comparé à celui obtenu en utilisant le modèle de langue par défaut que propose DeepSpeech, le modèle acoustique sans modèle de langue, le résultat de l'API de Google [16] et enfin le modèle par défaut de CMU Sphinx [17]

4.3.3 Résultats

Les résultats sont décrits dans le tableau 4.1 et la figure 4.3. Nous remarquons que sur notre mini-ensemble de test, les modèles par défaut obtiennent un score très proche de 1, ce qui démontre qu'ils ont été entraînés sur des cas assez généraux, ce qui n'est pas très bon. Cependant, en changeant juste le modèle de langue par défaut en injectant des échantillons que nous avons récolté, une nette amélioration est visible (environ -20%). Ce taux d'erreur diminue en augmentant la taille du corpus utilisé pour le modèle de langue. Toutefois, après avoir pris plus de 75% du corpus, l'erreur a légèrement augmenté. Cela peut s'expliquer par la nature assez bruitée du corpus, les fichiers README.MD qui sont rédigés par des personnes, l'erreur humaine, et l'absence de processus de vérification de l'orthographe, de la grammaire ou de la syntaxe du contenu. Nous avions envisagé de choisir comme corpus des extraits de livres dédiés à la manipulation des ordinateurs sous Linux, mais nous n'avons pas trouvé de ressources gratuites ou Open source exploitables.

16. https://cloud.google.com/speech-to-text/
17. https://cmusphinx.github.io/

Intention	Description de l'intention	Groupe	Argument(s) (Entité(s))
create_file_desire	Création d'un fichier	ALTER	-file_name -parent_directory
create_directory_desire	Création d'un répertoire	ALTER	-directory_name -parent_directory
delete_file_desire	Suppression d'un fichier	ALTER	-file_name -parent_directory
delete_directory_desire	Suppression d'un répertoire	ALTER	-directory_name -parent_directory
open_file_desire	Ouverture d'un fichier	INFO	-file_name -parent_directory
close_file_desire	Fermeture d'un fichier	INFO	-file_name -parent_directory
copy_file_desire	Copie d'un fichier	ALTER	-file_name -origin -destination
move_file_desire	Déplacement d'un fichier	ALTER	-file_name -origin -destination
rename_file_desire	Renommage d'un fichier	-	-old_name -new_name
change_directory_desire	Changement du répertoire de travail courant	-	-new_directory
inform	Informer d'une intention	EXCH	-file_name -parent_directory
request	Demander une information	EXCH	-file_name -directory
deny	Réponse négative	-	-
confirm	Réponse positive	-	-
unknown	Intention inconnue	-	-

Table 4.2 – *Tableau récapitulatif de toutes les intentions avec leurs descriptions et leurs arguments.*

Les intentions sont regroupées en groupes ; chaque groupe comporte des intentions qui influent sur les documents de la même manière. Il existe bien évidemment des intentions sans groupe, on peut considérer qu'ils forment un seul groupe, ou bien que chacun constitue son propre groupe :

- **ALTER** : Groupe d'intentions qui altère l'état d'un document.
- **INFO** : Groupe d'intentions pour effectuer une opération puis informer l'utilisateur.
- **EXCH** : Groupe d'intentions dont le but est d'échanger des informations avec l'utilisateur.

4.4.2 Méthodologie d'évaluation

Après avoir construit l'ensemble de test, un parcours exhaustif des différentes combinaisons des paramètres suivants est effectué :

- **Architecture d'encodage** : C'est à dire l'utilisation ou pas d'un réseau récurrent LSTM de base ou bien BiLSTM. Le but étant de montrer que le modèle pourra mieux interpréter les données en entrée s'il capture le contexte de chaque mot.
- **Nombre de neurones pour la couche de classification d'intention** : Lorsque l'encodeur retourne le dernier vecteur d'état caché, ce dernier passera par un réseau de neurones complètement connecté et multi-couches dont le nombre de couches est fixé à 3 par souci de performance, une couche d'entrée, une couche intermédiaire et une couche de sortie. Le nombre de neurones sur la couche cachée dépend grandement de la complexité de la tâche à effectuer. La classification d'intentions pour l'exploration de fichier était relativement simple ; nous avons commencé avec 32 neurones puis nous avons doublé ce nombre jusqu'à 512 pour, en théorie, donner plus de puissance au classificateur tout en évitant un sur-apprentissage par surplus de neurones.
- **Nombre d'unités d'une cellule LSTM (respectivement BiLSTM)** : Les portes d'une cellule LSTM (resp. BiLSTM) sont en vérité des réseaux de neurones denses (complètement connectés) et donc un ensemble de matrices de poids à optimiser. La capacité à "apprendre" la représentation des séquences dépend aussi du nombre de neurones dans ces mini-réseaux. Par le même raisonnement employé pour le classificateur d'intentions, nous avons commencé avec un petit nombre de neurones pour examiner d'une part où se trouverait le seuil minimal qui permettra au modèle de généraliser, et d'autre part où le seuil critique se situerait pour permettre au modèle de ne pas tomber dans un cas de sur-apprentissage. Nous partons de 128 jusqu'à 512 unités avec pas de 32 unités.
- **Fonctions d'activation** : C'est un élément essentiel qui permet d'introduire la non-linéarité dans les relations entre chaque neurones de couches voisines. Ces fonctions permettent de mieux représenter les seuils d'activation des neurones. La fonction la plus utilisée dans la littérature est actuellement ReLu (Rectified Linear Unit) car elle a expérimentalement donné de meilleurs résultats dans une grande variété de tâches et problèmes liés à l'apprentissage automatique et à la classification et étiquetage de textes. Par souci d'exhaustivité, nous avons quand même décidé de tester deux autres fonctions $tanh$ (Tangente Hyperbolique) et $sigmoid$ (Sigmoïde). Pour chaque couche de sortie, la fonction $Softmax$ a été appliquée car chaque couche traite d'un problème de classification multi-classes. Voici les équations pour chacune de ces fonctions :

$$\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

$$sigmoid(x) = \frac{1}{1 + e^{-x}}$$

$$softmax_i(x) = \frac{e^{x_i}}{\sum_{j=1}^{K} e_j^x} \text{ pour } i = 1, ..., K \text{ et } x = (x_1, ..., x_K) \in \mathbb{R}^K$$

- **Fonction erreur** : C'est l'élément clé pour la phase d'apprentissage. Cette fonction détermine le degré d'exactitude du modèle, c'est à dire à quel point il est proche de la bonne réponse. Nous avons décidé d'utiliser comme fonction erreur la fonction $Categorical\_Crossentropy$ ou Erreur Logistique ; de par la nature de l'ensemble d'apprentissage et de test. Une version pondérée de cette fonction a été préférée, pour palier au problème du non équilibrage des classes, que ce soit pour la classification d'intentions, ou pour la reconnaissance d'entités du domaine.

  Les poids des classes sont calculés selon la formule suivante :

  $$Poids_i = \max(1, \log \frac{T}{T_i})$$

  où :

  - T est le nombre total d'instances
  - T_i est le nombre d'instances dont la classe est C_i

  La formule de la fonction erreur devient donc :

  $$Erreur(y, \hat{y}) = -\sum_{i}^{C} y_i * \log(\hat{y}_i) * Poids_i$$

  où :

  - $\hat{y}$ est le vecteur en sortie produit par le modèle à la suite d'une fonction $Softmax$.
  - y est le vecteur de classe réelle présent dans l'ensemble d'apprentissage
  - C est le nombre de classes au total.
- **Fonction d'apprentissage** : Le choix de la fonction d'apprentissage est généralement affecté par un désir de précision et de rapidité. Une fonction qui converge rapidement en un minimum local peut être parfois préférée à une autre qui prendrait un temps considérable pour soit se retrouver dans le même minimum ou un autre minimum local (donc sans garantie de minimum optimal de la fonction erreur). Les deux fonctions utilisées sont RmsProp et Adam qui sont connues pour leur rapidité de convergence.
- **Encodage des entrée-sorties** : Là encore le choix de l'encodage des données influe grandement sur la capacité du modèle à distinguer et à représenter les différentes informations qui lui sont présentées. Comme initiative de notre part, nous avons mentionné dans la section 3.4.3 l'ajout de l'étiquette morphosyntaxique de chaque mot à l'encodage. Nous avons donc lancé les tests sur un encodage avec et sans l'ajout des étiquettes pour mieux constater son impact.
- **Découpage des données** : La stratégie adoptée était de prendre aléatoirement les même proportions pour chaque sous-ensemble de chaque classe (intentions ou entités de domaine). Nous avons aussi décidé de faire varier les proportions de test et d'apprentissage en fixant celui de validation car nous avons remarqué que notre modèle n'arrivait pas à bien généraliser la relation entre les entrées et les sorties. Notre intuition portait sur le fait que le manque de données pouvait en être la cause (voir la section 3.4.3 pour plus de détails). Ainsi, nous avons varié le taux de découpage pour

les données de tests entre 25% et 75% avec un pas de 25%. Le taux de découpage pour les données de validations est fixé à 10%.

Pour éviter que le modèle ne sur-apprenne, nous avons volontairement interchangé quelques mots dans la séquence d'entrée et celle de sortie pour introduire un certain taux d'erreur et de variété. Cet échange se fait suivant une probabilité q fixée à 20%. Bien entendu, les étiquettes morphosyntaxiques ne sont pas échangées, et ce pour garder la structure de la phrase correcte.

Pour le reste des hyper-paramètres, la plupart ont été fixés par manque de temps et de ressources. Ainsi, le nombre d'epochs (itérations) a été limité au maximum à 15 avec une politique d'arrêt anticipé si la fonction erreur d'évaluation ne diminue pas plus d'un taux $\Delta E = 2 * 10^{-3}$ pendant au moins 4 itérations successives. Les métriques employées pour évaluer les deux classificateurs sont les suivantes :

- **Précision** : il s'agit d'une métrique classique qui évalue à quel point le modèle est bon pour prédire les classes.

$$P = \frac{VP}{VP + FP}$$

où :

  - VP (Vrais Positifs) : nombre de cas où le modèle prédit correctement la classe comme étant positive.
  - FP (Faux Positifs) : nombre de cas où le modèle ne prédit pas correctement la classe comme étant positive.

- **Rappel** : Cette métrique évalue la capacité du modèle à effectuer des classifications correctes par rapport à tout l'ensemble de test. Plus formellement :

$$R = \frac{VP}{VP + FN}$$

où :

  - FN (Faux Négatifs) : nombre de cas où le modèle ne prédit pas correctement la classe comme étant négative.

- **F-Mesure** : Mesure qui combine (d'une point de vue ensembliste) la précision et le rappel. Elle ne privilégie aucune des deux et essaye de donner un aperçu plus global de l'efficacité de l'algorithme en prenant compte des résultats de ces deux mesures.

$$F - Mesure = \frac{2 * R * P}{R + P}$$

4.4.3 Résultats

Pour les résultats qui vont suivre, chaque tableau sera accompagné d'un paragraphe qui servira de commentaire aux résultats obtenus. Des remarques peuvent y être insérées pour attirer l'attention sur des détails non évidents.

- **L'utilisation de l'étiquetage morphosyntaxique** : L'ajout de l'information sur la syntaxe de la requête semble aider le modèle à construire une meilleure représentation interne de la requête. Il ne voit pas un mot seulement à un instant t, mais plutôt la paire (mot,rôle syntaxique du mot dans la phrase). Cet ajout est conforme à notre explication théorique dans la section 3.4.1

4.5 Ontologie et manipulation du graphe d'état

Les ontologies déjà présentées dans la section 3.5.2.1 ont été créées en utilisant Protégé. Elles ont été ensuite exportées en format Turtle [18] afin de les exploiter dans la suite de ce travail en utilisant la bibliothèque RDFLib de Python.

Les nœuds et les relations du graphe ont des identifiants entiers pour faciliter leur utilisation avec les réseaux de neurones. Une phase de transformation des URIs [19] en identifiants entiers s'avère nécessaire. L'ontologie comprend 61 nœuds et 13 relations. Contrairement au nombre de relations, le nombre de concepts augmente au cours du dialogue ; de nouveaux nœuds sont introduits après chaque échange. Ceci nécessite de garder un ensemble d'identifiants non utilisés pour les associer pendant le dialogue. Le tableau 4.13 résume l'association des identifiants aux nœuds et relations des graphes.

Ressource	**Valeurs des identifiants**
Nœuds de l'ontologie	[1-61]
Relations de l'ontologie	[1-13]
Nœuds créés pendant un dialogue	[62-256]

Table 4.13 – *Tableau des identifiants*

4.6 L'agent de dialogue

Nous avons proposé dans la partie conception 3.5.4 deux architectures pour entraîner l'agent de dialogue. La première se compose de deux parties, un module pour encoder le graphe d'état et un autre pour décider l'action à prendre. Chacun est entraîné séparément. Les deux modules étant des réseaux de neurones, nous avons pensé à une deuxième architecture en les connectant pendant la phase d'apprentissage. Cette connexion permet à l'encodeur de graphe de choisir les parties du dialogue, représentées par le graphe d'état, à mémoriser afin de mieux estimer la fonction de récompense.

18. Turtle est une syntaxe pour l'écriture des triplets d'un graphe de connaissance avec RDF
19. Les URIs identifient de manière unique les ressources dans le web sémantique et sont également utilisés pour identifier les concepts et les relations dans les ontologies

4.6.1 Encodeur de graphe

Dans la première méthode, l'entraînement de l'encodeur se fait avec une architecture encodeur-décodeur en passant les triplets du graphe comme entrées et sorties de cette architecture.

4.6.1.1 Implémentation

Le réseau de neurones a été implémenté en utilisant la bibliothèque Keras de Python. Nous avons utilisé des cellules GRUs (Gated Reccurent Units (Cho et al., 2014)) comme unités récurrentes pour l'encodeur et le décodeur vu leur efficacité comparable aux LSTMs tout en utilisant un seul vecteur d'état, ce qui les rend moins exigeants en mémoire. Comme notre tâche consiste à encoder un graphe dans un vecteur, la taille de ce dernier est très importante. Intuitivement, plus cette taille est grande plus le nombre de triplets qu'on peut y encoder est grand. D'où l'intérêt des cellules GRUs qui permettent d'utiliser de plus grands vecteurs d'état en moins d'espace mémoire que les LSTMs.

La génération aléatoire des graphes de taille t triplets se fait en suivant les étapes suivantes :

- choisir un nombre de nœuds nn aléatoire entre 2 et $t+1$.
- choisir un nombre d'arcs na aléatoire entre 1 et $nn-1$.
- choisir nn identifiants de nœuds et na identifiants d'arcs aléatoirement de l'ensemble des identifiants possibles.
- créer les triplets en choisissant pour chaque triplet deux nœuds et un arc aléatoirement des ensembles résultats de l'étape précédente.

Pendant l'apprentissage, le générateur choisit une taille pour le graphe inférieure à une taille maximale et crée un graphe en suivant les étapes sus-citées. Ce dernier est passé à l'encodeur-décodeur comme entrée et sortie désirée.

4.6.1.2 Résultats et discussion

Pour estimer la capacité de l'encodeur du graphe, nous avons varié la taille maximale du graphe ainsi que le vecteur d'état du GRU. Les résultats sont présentés dans le figure 4.6.

Nous remarquons évidemment que la précision diminue avec l'augmentation de la taille maximale du graphe. Cependant, l'augmentation de la taille du vecteur encodant le graphe n'améliore pas beaucoup les résultats. Ceci peut être dû à la nature combinatoire du problème. En effet, la taille du graphe augmente exponentiellement en fonction du nombre de triplets. nb et t représentent respectivement le nombre de triplets possibles et la fonction correspondant à la taille du graphe le nombre de graphes possible. t est définie par la fonction de récurrence suivante :

$$t(1) = nb$$

$$t(n+1) = t(n) \times nb$$

Nous utilisons un simulateur d'utilisateur pour communiquer avec l'agent de dialogue. Afin de simuler aussi les erreurs des modules précédents (reconnaissance automatique de la parole et compréhension du langage naturel), nous ajoutons un module qui insère du bruit dans les actions du simulateur. Sachant qu'une action utilisateur se compose de l'intention et des emplacements, ce dernier module prend donc deux valeurs de probabilités en paramètre ; la probabilité de bruiter l'intention et celle de bruiter les emplacements. Dans la suite de ce travail, nous allons varier ces deux valeurs ainsi que la taille du vecteur d'état afin d'arriver à un compromis entre la robustesse face aux erreurs, la réussite des tâches utilisateur et l'utilisation de la mémoire.

Nous allons évaluer les différents résultats en comparant leur taux de succès qui est donné par le rapport entre le nombre de fois où l'agent arrive au but sur le nombre total d'essais. Un succès nécessite d'arriver au but dans un nombre d'échanges limite que nous avons estimé comme suit :

$$limite = nb_{diff} \times 4 \times (1 + (pb_i + pb_e) \times 2)$$

- limite : le nombre d'échanges maximal.
- nb_{diff} : le nombre de fichiers dans l'arborescence de départ en plus ou en moins par rapport à l'arborescence but.
- pb_i : la probabilité d'erreurs dans l'intention de l'utilisateur.
- pb_e : la probabilité d'erreurs dans les emplacements de l'action.

La première partie de l'équation, $nb_{diff} \times 4$, détermine une approximation du nombre d'actions nécessaire afin d'arriver au but ; nous avons estimé qu'il faut un maximum de 4 échanges pour ajouter ou supprimer un fichier. La deuxième partie permet de prendre en compte les erreurs du simulateur. Sachant que le nombre moyen d'actions erronées en n échanges est égal à $n \times (pb_i + pb_e)$, nous avons donc ajouté deux fois ce nombre. Ceci permet en premier lieu de ne pas compter les actions erronées dans la limite des échanges possibles, ainsi que d'ajouter des actions afin que l'agent puisse se retrouver dans la conversation après une erreur du simulateur.

4.6.2.1 Apprentissage avec DQN déconnecté

Dans cette partie, nous utilisons un des encodeurs déjà entraînés dans la partie précédente. Nous avons donc choisi d'utiliser l'encodeur de taille 300 qui a été entraîné avec des graphes de taille maximale de 20. Nous avons varié le nombre de vecteurs utilisés ainsi que les probabilités d'erreurs du simulateur.

Le tableau 4.14 nous permet de conclure ce qui suit :

- Évidemment, avec moins de probabilités d'erreurs, le réseau arrive à mieux reconnaître les motifs dans le vecteur d'état et leurs relations avec les récompenses du simulateur. Cependant, l'augmentation de la taille du vecteur n'améliore que légèrement les résultats ; en calculant la moyenne des taux de réussite pour chaque nombre de vecteurs, les résultats sont comme suit : 53.5% de taux de réussite pour quatre vecteurs, 53.75% pour cinq vecteurs et 57.5% pour six vecteurs.

formation que certains nœuds sont plus importants que d'autres, les nœuds d'action par exemple, et qu'ils peuvent changer complètement l'état du vecteur.

- Les erreurs provenant de l'encodeur : Bien que la précision de l'encodeur obtenue était de 99%, en empilant 6 vecteurs encodés avec le même taux d'erreurs pour chacun et sachant que la probabilité d'erreur dans un vecteur est indépendante des autres, cette précision diminue à 96%.

4.6.2.5 Vitesse d'apprentissage

Nous avons comparé les vitesses d'apprentissage des deux approches. En utilisant le réseau DQN séparé, ce dernier apprend avec une moyenne de **1013.15 instances/seconde**. De l'autre côté, le réseau connecté ne fait que **197.61 instances/seconde**. La raison derrière la lenteur de la deuxième approche revient à la nécéssité de refaire l'encodage de tout le graphe d'état de l'instance sauvegardée pendant les échanges avec le simulateur. Tandis que dans la deuxième approche, on ne sauvegarde que le vecteur déjà encodé.

Les temps des échanges avec le simulateur des deux approches sont proches puisqu'ils se comportent de la même manière dans ce cas. Les deux méthodes n'encodent que les nouveaux triplets arrivant dans le vecteur d'état précédent.

4.6.2.6 Courbe d'apprentissage

Enfin, nous comparons à présent les courbes d'apprentissage des deux méthodes. Celles-ci montrent le taux de réussite par rapport au nombre d'épisodes [20]. La figure 4.7 contient quatre courbes : deux courbes pour chaque méthode, avec et sans erreurs.

Nous remarquons que la deuxième méthode arrive beaucoup plus rapidement à comprendre la fonction de récompense. Il s'avère donc que déconnecter le DQN de l'encodeur rend effectivement la tâche de trouver la relation entre les motifs des vecteurs d'états et les récompenses plus difficile.

L'apprentissage peut être amélioré dans les deux cas, et surtout en ce qui concerne le premier, en utilisant un meilleur encodage des nœuds et des arcs du graphe. En effet, les nœuds sont actuellement encodés avec des identifiants entiers seulement, perdant ainsi les riches connaissances sémantiques qu'on peut extraire des relations du graphe. Un meilleur encodage serait d'utiliser des méthodes d'apprentissage semi-supervisé qui permettraient de donner aux nœuds proches un encodage similaire. Cette méthode permet d'éviter quelques problèmes que nous avons cités dans 4.6.2.4, entre autres le problème d'encoder deux états lointains dans des vecteurs proches.

20. Un épisode est un ensemble d'échanges agent-simulateur qui aboutit à un succès ou un échec

Conclusion générale

Tout au long de la réalisation de ce mémoire, nous avons étudié l'état actuel des assistants personnels intelligents. Nous avons dû passer une majeure partie de cette étape à comprendre les fondements théoriques et conceptuels de chaque composant de ces systèmes, principalement à cause de la grande densité de techniques, concepts et théories qui sont nouvelles pour nous. Cela ne nous a pas empêchés de réaliser un travail dont nous sommes particulièrement fiers. Mais, nous gardons toutefois un esprit critique ainsi qu'une objectivité envers le travail fourni.

Après avoir assimilé la totalité des concepts, et qui font office d'état de l'art du domaine, nous sommes arrivés à certaines conclusions. Tout d'abord, développer un système en partant de rien était un travail assez massif, dépassant de loin le cadre d'un projet de fin d'études de master. Quatre modules ont été développés : le module de reconnaissance automatique de la parole, le module de compréhension automatique du langage naturel, le module de gestion du dialogue et le module de génération du langage naturel. L'accent à été mis sur certains d'entre eux comme le module de compréhension du langage naturel et le module de gestion du dialogue. Ces derniers dépendaient principalement de notre problématique. Le module de reconnaissance automatique de la parole a été sujet à une amélioration spécifique à nos besoins tout en exploitant un système de base déjà existant, à savoir DeepSpeech.

Avec une idée claire du travail à réaliser, nous avons pu entamer la conception de chaque module en y incorporant nos ajouts et modifications. Beaucoup de ces modifications sont le fruit de longues séances de débats et de discussions.

En ce qui concerne le module de reconnaissance automatique de la parole, l'ajout de notre modèle de langue a amélioré les résultats. Cela s'accordait avec nos prédictions théoriques. La partie de construction du corpus pour ce modèle a été soumise à beaucoup d'optimisations incrémentales, en tombant à chaque fois sur un nouveau problème, ou bien un obstacle matériel (manque de puissance de calcul). Par conséquent, les résultats n'étaient pas assez encourageants, surtout si le but est de concurrencer les systèmes propriétaires comme celui de Google qui réalise un score quasi-parfait sans apprentissage supplémentaire.

Pour le module de compréhension automatique du langage naturel, les ajouts faits au modèle d'apprentissage ont été expérimentalement validés dans le chapitre "Réalisations et résultats". L'ajout de l'information morphosyntaxique aux vecteurs d'encodage de chaque mot de la requête a enrichi la codification de cette dernière. L'introduction d'erreurs aléatoires dans le corpus d'apprentissage a quand à elle permis la gestion d'éventuelles erreurs que pourrait engendrer le module de reconnaissance automatique de la parole. La construction de l'ensemble d'apprentissage à partir de zéro était l'étape la plus longue de la réalisation de ce module. Plus l'ensemble grandissait, plus il était difficile de maintenir sa validité sans l'intervention d'un soutien externe. De plus, vu que la tâche à accomplir était relativement simple et limitée, le réel impact de cet ajout ne peut pas être certifié et validé dans un cadre plus général. Une autre problématique est celle du manque de données d'apprentissage consacrées au domaine de la manipulation d'ordinateurs. Ces données sont généralement construites manuellement par les développeurs du système. Un autre point à soulever est celui du manque de diversité dans les tâches réalisables par l'assistant. Cet ensemble de tâches reste facilement extensible. Il suffit d'ajouter des exemples assez exhaustifs à l'ensemble de données. Cela reste néanmoins une tâche lourde et manuelle à plus grande échelle.

En ce qui concerne le gestionnaire de dialogue, nous l'avons conçu pour qu'il soit facilement ajustable et mis à l'échelle. L'utilisation d'une architecture hiérarchisée d'agents de dialogue permet de réduire grandement la complexité de développement et d'ajout d'un nouveau gestionnaire de tâches dans le système. Pour représenter l'état interne d'une telle architecture, nous avons utilisé des graphes de connaissances au lieu des trames sémantiques dont la flexibilité permet de représenter tout l'état du dialogue des différents agents de l'architecture simultanément. Pour ce qui est des agents de dialogue, ils sont entraînés en utilisant des techniques d'apprentissage par renforcement. Ils interagissent avec un simulateur d'utilisateur afin d'atteindre un but à travers l'optimisation d'une politique d'actions basée sur un système de récompenses.

Cependant, l'utilisation du graphe de connaissances et de l'apprentissage profond a un prix. En effet, la codification de la totalité du graphe en un seul vecteur de taille fixe était une tâche assez difficile. La taille nécessaire de ce dernier devrait augmenter exponentiellement avec l'ajout de nouvelles connaissances. De plus, la codification choisie pour les nœuds du graphe a eu pour effet de faire perdre de l'information sémantique. Le décodage du graphe s'en est trouvé grandement affecté.

Pour ce qui est de l'application, nous avons fait le choix d'utiliser une architecture trois tiers basée web. Ce choix fût motivé par le fait que l'utilisation d'un serveur offre une puissance de calcul considérablement plus élevée que celle d'une machine personnelle en local. L'interface reste assez simple et épurée. Le but est de prioriser la parole comme moyen de communication avec le système. Cependant, cette interface a aussi pour but de montrer les fonctionnalités du système. Elle est donc plus orientée vers le développement.

En ce qui concerne les perspectives envisageables pour ce travail, nous avons longuement réfléchis à des alternatives possibles pour certains modules.

Premièrement, l'avenir de systèmes Open Source pour la reconnaissance automatique de la parole est très prometteur. Ces derniers offrent un moyen libre de mener des études et contribuer au développement à grande échelle de cette discipline. Une perspective future pour ce module serait de lancer notre propre plateforme de collecte de données pour le modèle acoustique et le modèle de langue. Cette idée a déjà été discutée dans la partie de l'état de l'art. Malheureusement, le temps a cruellement manqué. L'investissement de la communauté dans de telles initiatives n'en reste pas moins indéniable, comme l'a démontré le projet CommonVoice de Mozilla.

Deuxièmement, pour le module de compréhension automatique du langage naturel, faire appel à une collecte massive de données est une solution explorable. Le développement d'un outil d'aide à l'annotation d'un corpus était aussi le sujet d'un long débat. Le manque de temps nous a poussé à retarder son développement. La mise à l'échelle d'une telle plateforme pourrait grandement faire avancer la tâche fastidieuse qu'est la collecte de données.

Ensuite, en ce qui concerne le module de gestion du dialogue, il est envisageable d'utiliser une méthode d'apprentissage semi-supervisée pour l'encodage des nœuds du graphe. Cet ajout pourrait permettre d'enrichir la valeur sémantique de ces nœuds, et facilitera la tâche au réseau de neurones de l'agent apprenant pour le décodage du graphe, principalement grâce grâce au fait que les nœuds dont les sens sont arbitrairement proches auront des codifications similaires.

En ce qui concerne le module de génération du langage naturel, une méthode plus sophistiquée comme l'utilisation de modèles d'apprentissage automatique basés encodeur-décodeur, ou bien un convertisseur de graphes de connaissances en texte pourraient être utilisés. Cependant, ces architectures requièrent un très grand volume de données d'apprentissage annotées et spécifiques à notre problématique.

Enfin, pour ce qui est de l'application principale, une amélioration possible serait le déploiement du serveur dans un service de Cloud Hosting. Cela permettrait de minimiser les temps d'inférence et de post-traitement. Le développement d'une interface plus légère et plus orientée vers les cas pratiques est une amélioration possible. Garder les deux cas de figures, c.à.d. utilisation et développement, est aussi possible.

Pour conclure, nous estimons que la totalité du projet était une énorme occasion d'approfondir nos connaissances, que ce soit celles qui nous ont été enseignées durant notre cursus, comme l'apprentissage automatique, le traitement automatique du langage naturel, le web-sémantique et la représentation de connaissances, ou bien celles que nous avons apprises au cours de notre étude de la littérature, comme l'apprentissage par renforcement, le traitement automatique de la parole, le nettoyage des données, etc. Ce projet nous a aussi initiés au travail en équipe pour la réalisation d'un projet assez conséquent, tout en étant encadrés par nos supérieurs. Finalement, nous pensons que la plus grande satisfaction vient du fait que nous avons réalisé, dans un certain délai restreint, un travail qui traite d'un sujet récent et ambitieux, et ainsi, de poser notre propre pierre à l'édifice pour, idéalement, encourager les chercheurs en Algérie à s'investir dans ce domaine.

Bibliographie

Abowd, G. D., Dey, A. K., Brown, P. J., Davies, N., Smith, M., and Steggles, P. (1999). Towards a better understanding of context and context-awareness. In *Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing*, HUC '99, pages 304–307. Springer-Verlag. London, UK, UK.

Al-Anzi, F. S. and AbuZeina, D. (2018). Literature survey of arabic speech recognition. pages 1–6.

Angeli, G., Manning, C. D., and Jurafsky, D. (2012). Parsing time : Learning to interpret time expressions. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics : Human Language Technologies*, North American Chapter of ACL (NAACL) HLT '12, pages 446–455. Association for Computational Linguistics. Stroudsburg, PA, USA.

Åström, K. J. (1965). Optimal control of Markov Processes with incomplete state information. *Journal of Mathematical Analysis and Applications*, October:174–205.

Barlow, H. B. (1999). Unsupervised learning. chapter Unsupervised Learning, pages 1–17. Bradford Company. Scituate, MA, USA.

Bateman, J. A. (1997). Enabling technology for multilingual natural language generation : The kpml development environment. *Nat. Lang. Eng.*, 3(1):15–55. New York, NY, USA.

Bellman, R. (1957). A markovian decision process. *Indiana Univ. Math. J.*, 6:679–684.

Belz, A. (2008). Automatic generation of weather forecast texts using comprehensive probabilistic generation-space models. *Nat. Lang. Eng.*, 14(4):431–455. New York, NY, USA.

Bertsekas, D. P. (2007). *Dynamic Programming and Optimal Control, Vol. II*. Athena Scientific, 3e edition.

Bloit, J. and Rodet, X. (2008). Short-time viterbi for online hmm decoding : Evaluation on a real-time phone recognition task. pages 2121 – 2124. Las Vegas, NV, USA.

Bocklisch, T., Faulkner, J., Pawlowski, N., and Nichol, A. (2017). Rasa : Open source language understanding and dialogue management. *Computing Research Repository (CoRR)*, abs/1712.05181.

Chandramohan, S., Geist, M., Lefèvre, F., and Pietquin, O. (2011). User simulation in dialogue systems using inverse reinforcement learning. In *INTERSPEECH*, pages 1025–1028. Florence, Italy.

Chauhan, P. M. and Desai, N. P. (2014). Mel frequency cepstral coefficients (mfcc) based speaker identification in noisy environment using wiener filter. *Electrical Communications and Computers (CONIELECOMP)*. Coimbatore, India.

Chen, H., Liu, X., Yin, D., and Tang, J. (2017). A survey on dialogue systems : Recent advances and new frontiers. *SIGKDD Explor. Newsl.*, 19(2):25–35. New York, NY, USA.

Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734. Association for Computational Linguistics. Doha, Qatar.

Chorowski, J., Bahdanau, D., Serdyuk, D., Cho, K., and Bengio, Y. (2015). Attention-based models for speech recognition. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, pages 577–585. MIT Press. Cambridge, MA, USA.

Cowan, B. R., Pantidi, N., Coyle, D., Morrissey, K., Clarke, P., Al-Shehri, S., Earley, D., and Bandeira, N. (2017). "what can i help you with ?" : Infrequent users' experiences of intelligent personal assistants. In *Proceedings of the 19th International Conference on Human-Computer Interaction with Mobile Devices and Services*, MobileHCI '17, pages 43:1–43:12. Association for Computing Machinery (ACM). Vienna, Austria.

Cuayáhuitl, H., Renals, S., Lemon, O., and Shimodaira, H. (2005). Human-computer dialogue simulation using hidden markov models. pages 290–295. San Juan, Puerto Rico.

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. Vancouver, BC, Canada.

Dethlefs, N. (2014). Context-sensitive natural language generation : From knowledge-driven to data-driven techniques. *Language and Linguistics Compass*, 8(3):99–115.

Dingler, T. (2016). Cognition-aware systems as mobile personal assistants. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing : Adjunct*, UbiComp '16, pages 1035–1040. Association for Computing Machinery (ACM) Press. Heidelberg, Germany.

Elhadad, M. and Robin, J. (1996). An overview of surge : a reusable comprehensive syntactic realization component. In *International Natural Language Generation Workshop*.

Espinosa, D., White, M., and Mehay, D. (2008). Hypertagging : Supertagging for surface realization with ccg. In *Proceedings of ACL-08 : HLT*, pages 183–191. Association for Computational Linguistics (ACL). Columbus, Ohio.

Evans, R., Piwek, P., and Cahill, L. (2002). What is nlg ? In *Proceedings of the Second International Conference on Natural Language Generation*, pages 144–151. Association for Computational Linguistics (ACL). Tilburg University, The Netherlands.

Ferreira, T. C., Calixto, I., Wubben, S., and Krahmer, E. (2017). Linguistic realisation as machine translation : Comparing different mt models for amr-to-text generation. In *Pro-*

ceedings of the 10th International Conference on Natural Language Generation, pages 1–10. Association for Computational Linguistics (ACL). Santiago de Compostela, Spain.

Forney, G. D. (1973). The viterbi algorithm. *Proceedings of the Institute of Electrical and Electronics Engineers (IEEE)*, 61(3):268–278.

Gatt, A. and Krahmer, E. (2018). Survey of the state of the art in natural language generation : Core tasks, applications and evaluation. *Journal of Artificial Intelligence Research*, 61:65–170. USA.

Georgila, K., Henderson, J., and Lemon, O. (2005). Learning user simulations for information state update dialogue systems. In *Proceedings of the Ninth European Conference on Speech Communication and Technology*, pages 893–896. Lisbon, Portugal.

Ghahramani, Z. (2002). Hidden markov models. chapter An Introduction to Hidden Markov Models and Bayesian Networks, pages 9–42. World Scientific Publishing Co., Inc. River Edge, NJ, USA.

Ghai, W. and Singh, N. (2012). Literature review on automatic speech recognition. *International Journal of Computer Applications*, 41(8):42–50. Lisboa, Portugal.

Goddeau, D., Meng, H., Polifroni, J., Seneff, S., and Busayapongchai, S. (1996). A form-based dialogue manager for spoken language applications. volume 2, pages 701 – 704.

Goo, C.-W., Gao, G., Hsu, Y.-K., Huo, C.-L., Chen, T.-C., Hsu, K.-W., and Chen, Y.-N. (2018). Slot-gated modeling for joint slot filling and intent prediction. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics : Human Language Technologies, Volume 2 (Short Papers)*, page 753757. Association for Computational Linguistics (ACL). New Orleans, Louisiana.

Goyal, R., Dymetman, M., and Gaussier, E. (2016). Natural language generation through character-based rnns with finite-state prior knowledge. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics : Technical Papers*, pages 1083–1092. The COLING 2016 Organizing Committee. Osaka, Japan.

Graves, A., Mohamed, A., and Hinton, G. (2013). Speech recognition with deep recurrent neural networks. In *2013 Institute of Electrical and Electronics Engineers (IEEE) International Conference on Acoustics, Speech and Signal Processing*, pages 6645–6649. Vancouver, BC, Canada.

Halliday, M. A. K. and Matthiessen, C. M. I. M. (2004). *Introduction to Functional Grammar*. Hodder Arnold, 3 edition. London.

Hannun, A. Y., Case, C., Casper, J., Catanzaro, B., Diamos, G., Elsen, E., Prenger, R., Satheesh, S., Sengupta, S., Coates, A., and Ng, A. Y. (2014). Deep speech : Scaling up end-to-end speech recognition. *Computing Research Repository (CoRR)*, abs/1412.5567.

Henderson, J., Lemon, O., and Georgila, K. (2008). Hybrid reinforcement/supervised learning of dialogue policies from fixed data sets. *Comput. Linguist.*, 34(4):487–511. Cambridge, MA, USA.

Henderson, M., Thomson, B., and Young, S. J. (2013). Deep neural network approach for the dialog state tracking challenge. In *Special Interest Group on Discourse and Dialogue*

(SIGDIAL) Conference, pages 467–471. Association for Computational Linguistics (ACL). Metz, France.

Hochreiter, S. (1998). The vanishing gradient problem during learning recurrent neural nets and problem solutions. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 6(2):107–116. River Edge, NJ, USA.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Comput.*, 9(8):1735–1780. Cambridge, MA, USA.

Imtiaz, J., Koch, N., Flatt, H., Jasperneite, J., Voit, M., and van de Camp, F. (2014). A flexible context-aware assistance system for industrial applications using camera based localization. In *Proceedings of the 2014 Emerging Technology and Factory Automation (ETFA)*, pages 1–4. Institute of Electrical and Electronics Engineers (IEEE). Barcelona, Spain.

Janson, A. and de Gafenco, M. T. (2015). Engaging the appropriation of technology-mediated learning services - a theory-driven design approach. In *European Conference on Information Systems (ECIS)*. Münster, Germany.

Jurafsky, D. and Martin, J. H. (2008). *Speech and Language Processing, 2nd Edition*. Prentice Hall. Upper Saddle River, NJ, USA.

Kemeny, J. G. and Laurie Snell, J. (1957). Markov processes in learning theory. *Psychometrika*, 22(3):221–230.

Kipf, T. N. and Welling, M. (2017). Semi-supervised classification with graph convolutional networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. Toulon, France.

Knote, R., Janson, A., Eigenbrod, L., and Söllner, M. (2018). The what and how of smart personal assistants : Principles and application domains for is research. In *Multikonferenz Wirtschaftsinformatik (MKWI)*, pages 1083–1094. Lüneburg, Germany.

Kotsiantis, S., Zaharakis, I., and Pintelas, P. (2006). Machine learning : A review of classification and combining techniques. *Artificial Intelligence Review*, 26:159–190.

Labbé, C. and Portet, F. (2012). Towards an abstractive opinion summarisation of multiple reviews in the tourism domain. pages 87–94.

Langkilde-Geary, I. (2000). Forest-based statistical sentence generation. In *1st Meeting of the North American Chapter of the Association for Computational Linguistics*. California, USA.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521:436–44.

Lee, C., Jung, S., Kim, K., Lee, D., and Lee, G. G. (2010). Recent approaches to dialog management for spoken dialog systems. *Journal of Computing Science and Engineering (JCSE)*, 4:1–22. Pohang, Republic of Korea.

Li, Y., Tarlow, D., Brockschmidt, M., and Zemel, R. S. (2016). Gated graph sequence neural networks. *Computing Research Repository (CoRR)*, abs/1511.05493.

Lipton, Z. C. (2015). A critical review of recurrent neural networks for sequence learning. *Computing Research Repository (CoRR)*, abs/1506.00019.

Liu, B. and Lane, I. (2016). Attention-based recurrent neural network models for joint intent detection and slot filling. *Computing Research Repository (CoRR)*, abs/1609.01454. Montreal, Canada.

Liu, P., Qiu, X., and Huang, X. (2016). Recurrent neural network for text classification with multi-task learning. *Computing Research Repository (CoRR)*, abs/1605.05101.

López, G., Quesada, L., and Guerrero, L. A. (2017). Alexa vs. siri vs. cortana vs. google assistant : a comparison of speech-based natural user interfaces. In *International Conference on Applied Human Factors and Ergonomics*, pages 241–250. Springer, Cham.

Luger, E. and Sellen, A. (2016). "like having a really bad pa" : The gulf between user expectation and experience of conversational agents. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, CHI '16, pages 5286–5297. Association for Computing Machinery (ACM). San Jose, California, USA.

Mann, W. C. and Matthiessen, C. M. I. M. (1983). Nigel : A systemic grammar for text generation.

Marcus, M., Kim, G., Marcinkiewicz, M. A., MacIntyre, R., Bies, A., Ferguson, M., Katz, K., and Schasberger, B. (1994). The penn treebank : Annotating predicate argument structure. In *Proceedings of the Workshop on Human Language Technology*, pages 114–119. Association for Computational Linguistics. Stroudsburg, PA, USA.

McCandless, M. K. (1994). Automatic acquisition of language models for speech recognition. In *International Conference on Spoken Language Processing (ICSLP)*, pages 835–838. Massachusetts Institute of Technology (MIT). MA, USA.

Milhorat, P., Schlögl, S., Chollet, G., , B., Esposito, A., and Pelosi, G. (2014). Building the next generation of personal digital assistants. pages 458–463. Sousse, Tunisia.

Mitchell, T. (2006). The discipline of machine learning.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M. A., Fidjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518.

Murtagh, F. (1991). Multilayer perceptrons for classification and regression. *Neurocomputing*, 2(5-6):183–197. Garching, Germany.

Narang, S. and Gupta, M. D. (2015). Speech feature extraction techniques : A review. In *International Journal of Computer Science and Mobile Computing*, volume 4, pages 107–114.

Norman, D. A. (2002). *The design of everyday things*. Basic Books. New York.

Olah, C. (2015). Understanding lstm networks – colah's blog. `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`. (Accessed on 02/19/2019).

O'Shaughnessy, D. D. (1988). Linear predictive coding. *Institute of Electrical and Electronics Engineers (IEEE) Potentials*, 7:29–32.

Purington, A., Taft, J. G., Sannon, S., Bazarova, N. N., and Taylor, S. H. (2017). "alexa is my new BFF". In *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems - CHI EA '17*, pages 2853–2859. Association for Computing Machinery (ACM) Press. Denver, Colorado, USA.

Rabiner, R. and Juang, B. H. (1986). An introduction to hidden markov models. *Institute of Electrical and Electronics Engineers (IEEE) ASSP Magazine*, 3:4–16.

Rahali, H., Hajaiej, Z., and Ellouze, N. (2014). Robust features for speech recognition using temporal filtering technique in the presence of impulsive noise. *International Journal of Image, Graphics and Signal Processing*, 6:17–24.

Reiter, E. and Dale, R. (1997). Building applied natural language generation systems. *Natural Language Engineering*, 3(1). New York, NY, USA.

Roark, B., Saraclar, M., and Collins, M. (2007). Discriminative n-gram language modeling. *Comput. Speech Lang.*, 21(2). London, UK, UK.

Rummery, G. A. and Niranjan, M. (1994). On-line Q-learning using connectionist systems. Technical Report TR 166, Cambridge University Engineering Department. Cambridge, England.

Russell, S. J. and Norvig, P. (2003). *Artificial Intelligence : A Modern Approach*. Pearson Education, 2 edition.

Schatzmann, J., Thomson, B., Weilhammer, K., Ye, H., and Young, S. J. (2007). Agenda-based user simulation for bootstrapping a pomdp dialogue system. In *North American Chapter of ACL (NAACL)*, pages 149–152. Association for Computational Linguistics (ACL). Rochester, New York.

Schuster, M. and Paliwal, K. (1997). Bidirectional recurrent neural networks. *Trans. Sig. Proc.*, 45(11). Piscataway, NJ, USA.

Serban, I. V., Sordoni, A., Bengio, Y., Courville, A., and Pineau, J. (2016). Building end-to-end dialogue systems using generative hierarchical neural network models. In *Proceedings of the Thirtieth Conference on Artificial Intelligence*, Association for the Advancement of Artificial Intelligence (AAAI), pages 3776–3783. Association for the Advancement of Artificial Intelligence (AAAI) Press.

Snell, J. (2016). macos sierra review : Hey siri, where did my files go ? - six colors. `https://sixcolors.com/post/2016/09/sierra-review/`. (Consulté le 29/10/2018).

Sonali, B. (2014). Research paper on basic of artificial neural network. page 1.

Sordoni, A., Galley, M., Auli, M., Brockett, C., Ji, Y., Mitchell, M., Nie, J.-Y., Gao, J., and Dolan, W. B. (2015). A neural network approach to context-sensitive generation of conversational responses. In *North American Chapter of ACL (NAACL)*, pages 196–205.

Stoyanchev, S. and Johnston, M. (2018). Knowledge-graph driven information state ap-

proach to dialog. In *Association for the Advancement of Artificial Intelligence (AAAI) Workshops*.

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. (2015). Rethinking the inception architecture for computer vision. *Computing Research Repository (CoRR)*, abs/1512.00567.

Theune, M., Klabbers, E., De Pijper, J. R., Krahmer, E., and Odijk, J. (2001). From data to speech : A general approach. *Natural Language Engineering*, 7(1). New York, NY, USA.

Trappl, R. (2013). *Your Virtual Butler*. Springer Berlin Heidelberg.

Tulshan, A. and Namdeorao Dhage, S. (2019). *Survey on Virtual Assistant : Google Assistant, Siri, Cortana, Alexa : 4th International Symposium SIRS 2018, Bangalore, India, September 1922, 2018, Revised Selected Papers*, pages 190–201.

VanDijck, J. (2005). From shoebox to performative agent : the computer as personal memory machine. *New Media & Society*, 7:311–332.

Velay, M. and Daniel, F. (2018). Seq2seq and multi-task learning for joint intent and content extraction for domain specific interpreters. *Computing Research Repository (CoRR)*, abs/1808.00423:3–4.

Wang, Y., Shen, Y., and Jin, H. (2018). A bi-model based rnn semantic frame parsing model for intent detection and slot filling. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics : Human Language Technologies, Volume 2 (Short Papers)*. Association for Computational Linguistics. New Orleans, Louisiana.

Weisz, G., Budzianowski, P., hao Su, P., and Gax0161ix0107, M. (2018). Sample efficient deep reinforcement learning for dialogue systems with large action spaces. *Institute of Electrical and Electronics Engineers (IEEE)/ACM Transactions on Audio, Speech, and Language Processing*, 26:2083–2097.

Wen, T.-H., Gasic, M., Mrksic, N., hao Su, P., Vandyke, D., and Young, S. J. (2015). Semantically conditioned lstm-based natural language generation for spoken dialogue systems. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Lisbon, Portugal.

Wen, T.-H., Gasic, M., Mrksic, N., Rojas-Barahona, L. M., hao Su, P., Ultes, S., Vandyke, D., and Young, S. J. (2017). A network-based end-to-end trainable task-oriented dialogue system. In *uropean Chapter of the Association for Computational Linguistics (EACL)*, pages 438–449. Valencia, Spain.

Wessel, M., Acharya, G., Carpenter, J., and Yin, M. (2019). *OntoVPAAn Ontology-Based Dialogue Management System for Virtual Personal Assistants : 8th International Workshop on Spoken Dialog Systems*. Miyazaki, Japan.

Williams, J. D. and Young, S. (2007). Scaling pomdps for spoken dialog management. *Trans. Audio, Speech and Lang. Proc.*, 15(7):2116–2129. Piscataway, NJ, USA.

Young, S., Gašić, M., Keizer, S., Mairesse, F., Schatzmann, J., Thomson, B., and Yu, K.

(2010). The hidden information state model : A practical framework for pomdp-based spoken dialogue management. *Comput. Speech Lang.*, 24(2):150–174. London, UK, UK.

Yu, D. and Deng, L. (2015). *Automatic Speech Recognition*. Springer London.

Yu, J., Reiter, E., Hunter, J., and Mellish, C. (2007). Choosing the content of textual summaries of large time-series data sets. *Natural Language Engineering*, 13(1). New York, NY, USA.

Zibreg, C. (2016). Apple shares examples of siri's third-party app integration on ios 10. `https://www.idownloadblog.com/2016/09/01/apple-siri-ios-10-app-integration/`. (Consulté le 29/10/2018).