Dorian THOMAS-TORRES, Eric MAISEL,
Pierre DE LOOR, Benjamin BERNARD

Rapport de stage Master Systèmes Intelligents, Interactifs et Autonomes

Janvier – Juin 2020

Lab-STICC

ÉCOLE NATIONALE D'INGÉNIEURS DE BREST

Crédit Mutuel ARKEA

Remerciements

Je tiens à remercier toute l'équipe du LAB-STICC de m'avoir accueilli au CERV pour la durée de mon stage et plus particulièrement Eric MAISEL et Pierre DE LOOR d'avoir été mes professeurs référents.

Je remercie de même le crédit mutuel ARKEA m'avoir proposé un sujet de stage en recherche et plus particulièrement Benjamin BERNARD et son équipe d'avoir été mes interlocuteurs privilégiés.

Sommaire

Résumé

Avec le récent développement des assistants vocaux multimédias, ou plus communément appelés chatbots, il devient possible de faire des réservations comme si l'on s'adressait verbalement à une personne réelle. Ces commerçants électroniques n'ont pas les mêmes capacités d'interprétation que celle des humains. Des erreurs peuvent donc apparaitre au cours de la conversation. Dans ce document nous nous intéresserons à ces erreurs, à comment nous avons recodé un système de reconnaissance d'entités nommées axé sur la prononciation pour corriger certaines de ces erreurs dans un contexte particulier et à l'étude d'un réseau de neurones pour faire de la classification de texte avec peu de données.

Mots-clés

Assistant vocal, Traitement du Langage Naturel, Reconnaissance d'entités nommées, Few-Shots Learning, classification de texte, analyse de sentiments.

1. Introduction

Un assistant personnel intelligent est un système électronique permettant de rendre service à un utilisateur humain. L'utilisateur doit formuler une requête, généralement sous forme textuelle ou vocale, qui sera traitée et réalisée par l'assistant. De nos jours, la plupart de ces systèmes sont capables de tenir une conversation avec leur interlocuteur quand les requêtes qui lui sont adressées sont compréhensibles et correctement interprétées. Ces systèmes sont plus communément appelés chatbots. En lui transmettant les bonnes requêtes, l'assistant vocal pourra envoyer des messages qui lui seront dictés au destinataire voulu, informera de la météo dans les prochaines heures, réservera son taxi à l'heure souhaitée par l'utilisateur, jouera les titres de musique préférés de celui-ci ou lui tiendra tout simplement compagnie.

Le premier système de reconnaissance vocale a été créé par IBM bien avant leur premier ordinateur portable [1]. Le système pouvait reconnaître 16 mots et les chiffres de 0 à 9. SIRI, l'assistant vocal conçu par Apple, fait sa première apparition sur l'iPhone 4s en 2010. C'est ce système qui aurait lancé la mode des assistants vocaux dans le domaine grand public [2]. Après que SIRI ait fait forte impression, les concurrents d'Apple ont décidé de développer des systèmes similaires à SIRI tels qu'Alexa (Amazon) ou Cortana (Microsoft).

Il arrive parfois que les chatbots n'interprètent pas correctement les demandes de l'utilisateur malgré les avancées technologiques réalisées dans ce domaine. L'incompréhension chez un assistant vocal peut être due à la mauvaise prononciation d'une requête (« Je voudrais voir Skywalker », en prononçant Skywalker à la française) ou l'emploi d'homophones (« Je voudrais aller voir Glass. », pourrait être traduit sous forme symbolique par le système, par « Je voudrais aller voir glace. »). Elle peut aussi provenir d'une méconnaissance d'un terme lié à un contexte particulier comme « Passe-moi la dernière de Cloclo » alors que l'on s'adresse à un assistant vocal qui permet de réserver des places de cinéma.

Dans ce document, nous commencerons par énumérer les erreurs qui peuvent survenir lors d'une interaction humain-chatbot, puis nous décrirons le fonctionnement du système de reconnaissance d'entités nommées (Named Entity Recognition ou NER) que nous avons recodé pour palier à ces erreurs et enfin nous parlerons du travail effectué sur un réseau de neurones utilisé pour faire de la classification de texte applicable à des jeux de données contenants peu d'exemples : le Few-Shots Learning (FSL). *Les codes et les ressources sont disponibles en annexes.*

2. Problématique

La principale problématique proposée par le crédit mutuel ARKEA est de trouver un système capable de corriger les erreurs de compréhension d'un assistant vocal dédié à la réservation de places de cinéma pendant une interaction avec un client. Ce chatbot est utilisé notamment par le cinéma Liberté à Brest. Il est accessible à partir de l'assistant vocal google.

3. Les erreurs en reconnaissance vocale

3.1. Les sources possibles

Dans leur article, Niculescu et Banchs [3] (§2) passe en revue plusieurs sources d'erreurs possibles dans l'interaction entre un système de reconnaissance vocale et un utilisateur humain. Une des premières erreurs qui peut être réalisée par la machine est l'erreur de mise en contexte. Il faut que l'utilisateur contextualise correctement sa demande. Par exemple, il souhaite réserver une place de cinéma, il devra utiliser des termes fortement liés au domaine du cinéma et faire preuve de bon sens en s'adressant au chatbot.

D'après Niculescu et Banchs cela reste difficilement réalisable avec les technologies domestiques actuelles car traiter ces requêtes demande une puissance de calcul et de nombreuses ressources externes. C'est pour cela que les assistants vocaux multimédias ont besoin d'une connexion internet pour dialoguer avec des serveurs dédiés.

Une autre source d'erreur provient de la capacité du système à séparer l'énoncé en unités linguistiques (phonèmes, syllabes, mots, phrases, paragraphes, …), ce que l'on appelle la segmentation ou tokenisation en anglais. Cela peut être lié au fait que les systèmes de reconnaissance vocale sont sensibles au changement de technologie d'acquisition et à leur environnement (bruyant, raisonnant, …).

Ils peuvent aussi avoir des difficultés à traiter les homophones, par exemple « le pain » et « le pin », car actuellement il reste compliqué de modéliser la sémantique exacte d'une phrase (ici parlant de la boulangerie ou de la forêt) et donc de trouver le mot qui fait sens.

Les erreurs peuvent également être provoquées par le style de diction de l'utilisateur qui n'est pas adapté à celui du système de reconnaissance vocale. Il varie, par exemple, en fonction de l'expérience passée que l'utilisateur a pu avoir les chabots (formulation par mots-clés et non par phrases complètes) ou en fonction de sa prononciation, ou de sa vitesse de diction, qui peut engendrer des erreurs de segmentation.

L'utilisateur peut formuler des requêtes hors du domaine de connaissances du système (« Que dois-je choisir entre un pantalon et un short ? » alors que l'on s'adresse à un chatbot spécialisé dans le cinéma) ou hors du contexte de la discussion courante. S'il oublie ou n'arrive pas à formuler une requête dans le temps impartie par le système, on parle alors d'erreur d'entrée vide.

3.2. Prévention des erreurs grâce à des méthodes générales

Niculescu et Banchs proposent §3.1 plusieurs solutions d'ordre général pour réduire l'apparition d'erreurs lors d'un dialogue verbal homme-machine. D'après eux, il est impossible d'éviter les erreurs (qui d'ailleurs peuvent survenir dans les dialogues entre humains). Les erreurs produites non intentionnellement par le chatbot permettraient de rendre la discussion plus réaliste, thèse que partagent Bradesko et

Mladenic §5.2 [4]. Pour prévenir les erreurs de compréhension on peut utiliser différents canaux d'acquisition de l'information (systèmes multimodaux). Par exemple, le système complète le discours verbal par des expressions faciales, labiales et gestuelles de l'utilisateur pour connaître son état émotionnel actuel. En cas d'ambiguïté sur des noms propres (par exemple entre la ville de Foix et l'organe foie), le chabot devrait permettre à l'utilisateur de les définir lui-même. Plus tôt cette demande sera réalisée dans le dialogue, moins le système aura de chance de commettre d'erreur à cause de ces ambiguïtés. De même, il faut signaler à l'utilisateur les interactions qu'il peut avoir avec le système si celles-ci changent pendant la conversation. Cela peut permettre d'éviter des situations comme la suivante : après avoir réservé sa place de cinéma, le chatbot demande à l'utilisateur « Avez-vous encore besoin de mon aide ? », l'utilisateur lui demande « combien de temps durera le trajet ? », or le système n'est pas capable de répondre à cette requête car il n'effectue que des réservations et ne fait pas de prédiction.

Il faut aussi faire attention à la quantité d'informations transmise lorsque l'assistant vocal s'exprime. Il faut qu'elle soit correctement répartie tout le long de la conversation et segmentée en courts énoncés. Les premières paroles du chatbot sont très importantes car elles permettent de décrire le ou les services qu'il peut rendre et comment s'adresser à lui. Quand le système a besoin de plusieurs réponses ou que l'utilisateur reformule plusieurs fois son énoncé, il est conseillé que le chatbot rappelle les requêtes ou les réponses formulées précédemment par l'utilisateur pour que celui-ci reste conscient de l'état de la conversation. Le chatbot doit éviter d'utiliser des formulations ouvertes ou qui ne sont pas contingentées à un domaine (sauf si cela est nécessaire) pour ne pas faire hésiter l'utilisateur. Une demande bien formulée de la part de l'assistant vocal permet à l'utilisateur de bien construire sa réponse.

Dans certains cas le système peut prendre du temps pour traiter l'information. Il est alors recommandé de le faire savoir. Il faut aussi éviter que le système coupe la parole à son interlocuteur car les longs moments de silence en traitement du langage naturel représentent généralement la fin de l'énoncé utilisateur. On peut faire en sorte que le chatbot puisse être interrompu lorsque l'utilisateur prend la parole. Enfin, l'assistant vocal peut mettre en valeur, explicitement ou implicitement, son style conversationnel pour que l'utilisateur s'adapte à celui-ci et éviter une mauvaise compréhension. Par exemple, on ne s'adresse pas de la même manière à un enfant ou à un scientifique.

3.3. Correction des erreurs grâce à des méthodes générales

Pour Niculescu et Banchs §3.2, il existe plusieurs méthodes de correction des erreurs qui évitent de faire appel à des techniques avancées. On peut mettre en place dans le système un mécanisme de réponse qui contient les données clés de la requête utilisateur. Par exemple l'utilisateur demande « J'ai besoin d'une place pour la Reine des neiges 2 », le système répond « Bien sûr, je vous réserve une place pour Maléfique », l'utilisateur peut alors corriger immédiatement « Non, Je veux acheter une place pour la Reine des neiges 2 ! ». Demander de clarifier une requête peut s'avérer

difficile et pousser l'utilisateur à utiliser des termes inconnus. Dans ce cas de figure il faut présenter les alternatives possibles. Si l'utilisateur ne répond pas après une seconde et dernière sollicitation, l'assistant vocal doit mettre fin à la conversation poliment et guider l'utilisateur vers une autre source d'information comme un moteur de recherche classique. Il peut être utile de conserver un historique de la conversation, qui peut permettre à l'assistant ainsi qu'à l'utilisateur de se tenir au courant du contexte et de détecter les moments d'incompréhension. L'agent conversationnel peut faire usage de l'humour pour camoufler l'incompréhension et le temps de correction. Cela permet de conserver la qualité d'expérience utilisateur et de donner au système un comportement plus humain.

3.4. Correction des erreurs grâce au machine learning

Il existe des techniques plus avancées de correction d'erreur qui utilisent le Machine Learning comme celle de Kou et al. [5]. Auparavant, pour mieux apprendre la prononciation des mots et améliorer la compréhension du chatbot, ils ont utilisé la méthode du *crowdsourcing* qui consiste à demander à plusieurs personnes de prononcer le même énoncé pour que l'assistant vocal apprenne à maîtriser ces différentes prononciations. Mais cette méthode présente quelques défauts comme l'apprentissage de la prononciation de mots peu utilisés, par exemple pour certain nom propre, qui peut varier en fonction de la localisation du locuteur. Ils proposent donc une nouvelle solution qui consiste à d'apprendre la prononciation d'un mot à l'aide de la correction fournie par l'utilisateur (système multimodal). Les entrées choisies pour corriger les erreurs sont la saisie textuelle au clavier et une technique appelée la sélection alternative de données.

Généralement quand une requête vocale n'est pas correctement transcrite textuellement, l'utilisateur a tendance à la corriger au clavier dans les 30 secondes qui suivent. Lors d'une expérimentation menée dans 11 langues et sur plusieurs paires de requêtes vocales et de correction textuelle, seulement 30 à 40% de celles-ci ont une correction correcte. Pour détecter les paires correctes, Kou et al. les classifient de façon binaire à l'aide d'une régression linéaire à qui ils fournissent en entrée un vecteur contenant des caractéristiques sur les mots, les caractères, les phonèmes et des données acoustiques. Le modèle de classification est entraîné sur 8000 paires pour chaque langage et dans lesquelles les saisies au clavier sont labellisées manuellement. Après avoir effectué la classification, il s'avère que la plupart des erreurs commises en américain proviennent d'homophones ou de prononciations qui ne figurent pas dans le dictionnaire d'entrées.

La sélection de données alternative est un mécanisme proposé par le système de recherche vocale Google. Ce mécanisme propose à l'utilisateur plusieurs formulations alternatives proches sémantiquement du mot sélectionné et dont la paire de données vocale et textuelle est déjà correctement corrélée grâce à un algorithme de traitement automatisé du langage.

L'expérimentation proposée par Kou et al. consiste à trouver parmi plusieurs prononciations celle qui se rapproche le plus d'un échantillon comprenant l'énoncé textuel et audio. La reconnaissance vocale se fait à l'aide d'un réseau de neurones profond, d'un transducteur à état fini (« *FTS* ») et un modèle de langage 5-gram (le mot courant dépend des 4 derniers mots) [6]. Les performances sont mesurées selon le taux d'erreur sur les mots (« *Word Error Rate* ») pour s'assurer que le système de reconnaissance vocale n'apprenne pas de « *prononciation assassine* ». C'est une prononciation pour un mot rare qui est liée par le système à celle d'un mot plus fréquent qui deviendra alors non reconnaissable. Et selon le test de « *side-by-side (SxS)* » [7] qui permet de mieux visualiser l'effet du changement de prononciation sur la reconnaissance vocale. Deux systèmes de reconnaissance vocale (ASR) sont créés, un à partir du dictionnaire où les prononciations corrigées apprises ont été ajoutées et l'autre avec un dictionnaire classique. Les requêtes utilisées en entrée des systèmes d'ASR ont été triées en 4 catégories représentées par différents poids : exacte, utilisable, non utilisable et sans aucun sens. Le but est d'avoir un score plus élevé au test SxS avec le système d'ASR au dictionnaire modifié qu'avec le celui au dictionnaire basique. Les résultats obtenus grâce à la correction par saisie textuelle montrent une légère baisse du taux d'erreur sur les mots et une augmentation du score au test SxS.

Data set	# Utterances	Baseline WER	Experiment WER
Search	22K	10.9	10.8
Actions	12K	21.9	21.8
Maps	18K	11.4	11.3

Figure 1 : comparaison du WER entre l'ASR utilisant le dictionnaire américain de base et le dictionnaire modifié extraite de [5].

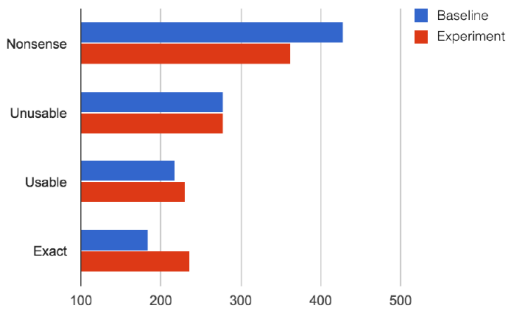


Figure 2 : résultats du test SxS pour une correction de requêtes en anglais au clavier extraits de [5].

Avec la technique de correction des données par choix alternatif il n'y a pas de modification du « *WER* », cela est dû à une faible quantité de données recueillies. Par contre le score au test SxS est encourageant et passe de 0,418 pour un dictionnaire classique à 0,457 pour un dictionnaire amélioré à l'aide des techniques utilisées.

4. Cine mind : système de Named Entities Recognition (NER) des titres de films

Cine mind (annexe i) est un programme adapté de Cimind [8] qui permet de faire de la reconnaissance d'entités nommées ou Named Entity Recognition (NER) « phonétique ». Cela consiste à rechercher un mot, ou un groupe de mots appartenant à une classe (noms de personnes, noms d'organisations ou d'entreprises, noms de lieux, quantités, distances, valeurs, dates, ...) dans un document textuel [9] grâce à la prononciation. Dans cette partie nous allons tenter de répondre à l'énoncé suivant : étant donné un ensemble de classes *Cf* (chaque classe correspondant à un film *f*), étant donné *t*, la transcription textuelle d'une requête vocale exprimée par un utilisateur, il s'agit de trouver la classe à laquelle appartient *t*. Sachant que *t* appartient à *Cf* si et seulement si *t* contient une référence au titre de *f*.

En entrées, Cine_mind a besoin d'une représentation textuelle de la requête vocale *t* (conversion Speech-To-Text). Cette représentation est produite par un agent DialogFlow [10] en amont du programme dont nous ne détaillerons pas ici le fonctionnement et d'une base de données qui sera d'écrite plus tard. Le programme retourne une représentation du degré d'appartenance à une classe de *Cf*. Le calcul se fait en deux étapes, d'abord un prétraitement est réalisé pour rejeter les titres de films qui ne concordent phonétiquement avec aucune partie de la requête utilisateur, puis une comparaison entre la transcription *t* et les titres retenus précédemment.

Le schéma synoptique de la *figure 3* présente le déroulement global de l'algorithme. Les étapes qui y sont présentées sont développées dans les rubriques qui suivent.

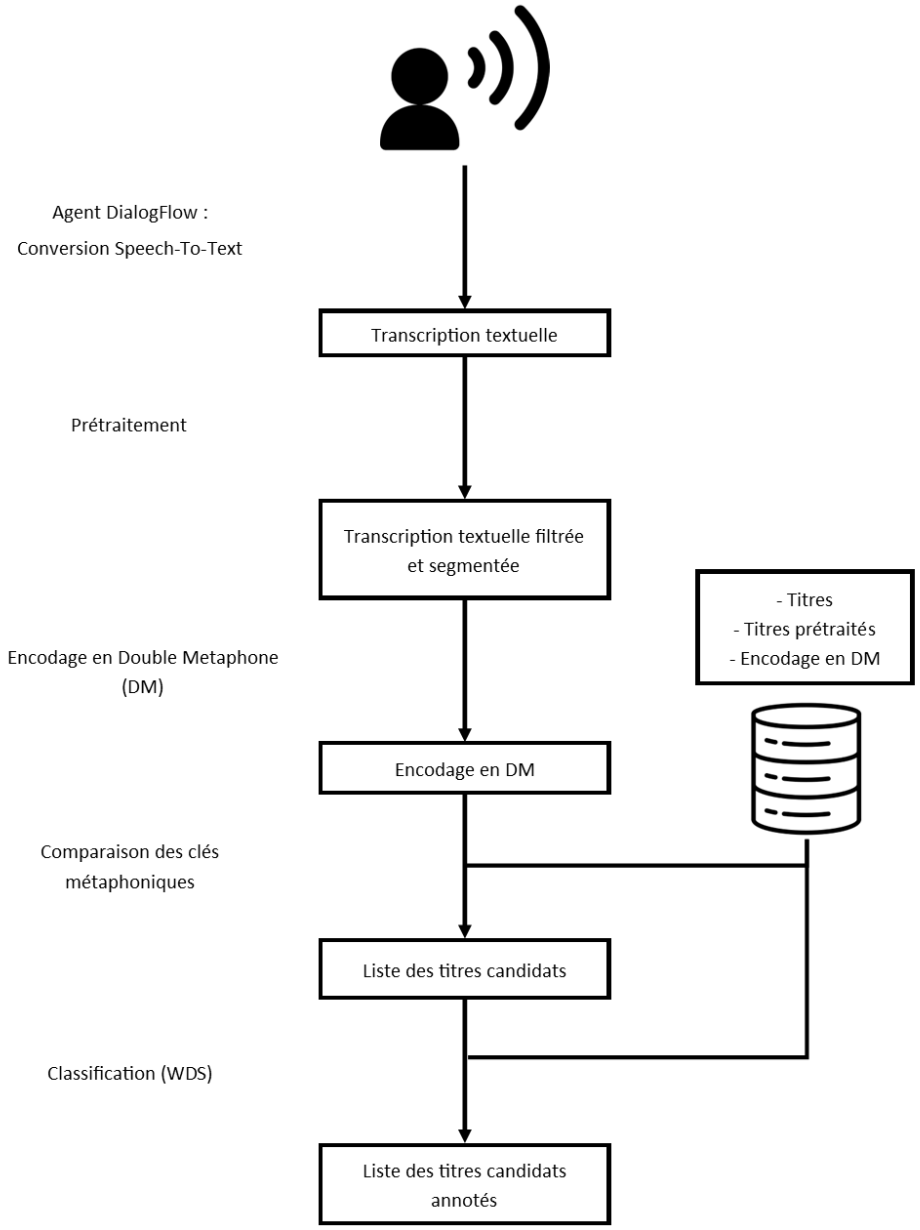


Figure 3 : Synoptique du déroulement de l'algorithme Cine_mind.

4.1. Pré-traitement de la requête utilisateur *t* [11]

La première étape consiste en une segmentation du texte (tokenisation) à l'aide d'une API de traitement du langage naturel. Il s'agit de découper les documents en mots, phrases, symboles ou tout autre élément pouvant avoir du sens. Ces éléments sont appelés tokens. Le programme divise les requêtes textuelles formulées par les utilisateurs en mots.

Exemple : « Je voudrais aller voir Glace. » devient [« Je », « voudrais », « aller », « voir », « Glace », « . »].

Tous les tokens sont ensuite capitalisés. Contrairement à son nom, on transforme les lettres majuscules en minuscules.

Exemple : « Je voudrais aller voir glace » devient [« je », « voudrais », « aller », « voir », « glace », « . »].

Puis les stop words et le bruit sont supprimés. Les stops words sont des mots qui n'ont pas de signification utile pour le traitement à venir et la suppression de bruit concerne toute la ponctuation qui est inutile pour la suite du traitement.

Exemple : « Je voudrais aller voir glace » devient [« voudrais », « voir », « glace »].

Enfin, nous intégrons un algorithme de correction d'orthographe pour rendre le traitement à suivre plus efficace.

4.2. Encodage en Double Métaphone (DM) [12]

Le Metaphone est un algorithme inventé par Lawrence Philips en 1990 pour indexer les mots par leur prononciation. Plus performant que l'algorithme Soundex [13], il utilise les variations et les incohérences dans l'orthographe et la prononciation de l'anglais pour produire un encodage plus précis des mots. Cela permet réaliser un meilleur travail de mise en correspondance des mots et des noms de même consonnance. Les mots de même consonance partagent généralement les mêmes clés.

Contrairement à l'algorithme d'origine dont l'application est limitée à l'anglais uniquement, le Double Metaphone (2000) prend en compte les particularités orthographiques d'un certain nombre d'autres langues. En 2009, Lawrence Philips a publié une troisième version (Metaphone 3) qui permet d'indexer dans 99% des cas les mots anglais, les mots non anglais familiers aux Américains et les prénoms et noms de famille les plus fréquents aux États-Unis.

Nous nous intéresserons plus particulièrement au Double Metaphone qui apporte un certain nombre d'améliorations fondamentales au Metaphone. Il est appelé "Double" car il peut renvoyer un code primaire et secondaire pour une chaîne de caractères. Cela permet d'expliquer certains cas ambigus. Par exemple, le codage du nom "Smith" donne un code primaire de SM0 et un code secondaire de XMT, tandis que le nom "Schmidt" donne un code primaire de XMT et un code secondaire de SMT - tous deux ont XMT en commun. L'algorithme Double Metaphone essaie de tenir compte des irrégularités anglaises d'origine slave, germanique, celtique, grecque, française, italienne, espagnole, chinoise et autres. Par exemple, il teste environ 100 contextes différents de l'utilisation de la lettre C seule. Il existe certaines idées fausses sur les algorithmes du Metaphone. Les affirmations suivantes sont vraies :

- Tous sont conçus pour travailler sur des dictionnaires de mots courants (noms communs, verbes, adjectifs, ...), pas seulement sur des noms.
- Les algorithmes « métaphoniques » ne produisent pas de représentations phonétiques des mots. La sortie est une représentation phonétique approché qui suit les règles suivantes : les mots qui commencent par un son de voyelle auront un « A » comme premier caractère de l'encodage (dans Double Metaphone et Metaphone 3, le métaphone d'origine préserve juste la voyelle réelle). Les voyelles après un son de voyelle initial ne seront pas codées.

Les paires de consonnes voisées et non voisées seront conçues sur le même codage (exemples : D / T, B / P, Z / S, G / K, ...). Cet encodage approché est nécessaire pour tenir compte des variations orthographiques et de prononciation. Les Anglais se plaignent souvent que les Américains prononcent « T » comme « D ». Les Anglais prononcent souvent la lettre « S » en « Z », généralement lorsqu'un nom se terminant par une consonne voisée ou un liquide pluralisé, par exemple « seasons », « examples », ... Ne pas encoder les voyelles après un son de voyelle initial aidera à regrouper les mots où une voyelle et une consonne peuvent être transposées dans la prononciation erronée ou alternative.

Exemple : [« voudrais », « voir », « glace »] devient [(FTR, FTRS), (FR,), (KLS,)].

4.3. Comparaison de prononciation

Une base de données est constituée au préalable avec les titres de films hebdomadaires et leur encodage en DM. Les clés de prononciation (primaire et secondaire) des mots de *t* sont comparées aux clés des titres de films contenues dans cette base de données pour dresser une liste de candidats potentiels. Voici la liste des titres de films (candidats) ayant approximativement la même prononciation que le titre du film contenu dans la requête de notre utilisateur :

Exemple : [« glass suis de glace », « la glace et moi », « glass »].

4.4. Classification des candidats

Un algorithme appelé Weighted Distance Scale (WDS), basé sur la distance de Levenshtein (DL) [14] entre *t* et chaque classe de *Cf* contenu dans la base de données permet de trier les titres de films dans un ordre de pertinence : plus le score en sortie de la WDS est élevé, plus *f* à de chance d'être celui contenu dans *t*.

Exemple : [(« glass », 75.75), (« suis de glace », 62.96), (« la glace et moi », 62.96)]

La distance de Levenshtein (Levenshtein distance ou edit metric en Anglais) est un algorithme de mesure de différence entre deux chaînes de caractères créé par Vladimir Levenshtein en 1965. Le résultat obtenu lors de cette mesure est le nombre minimal de caractères à supprimer, insérer ou remplacer pour passer d'une chaîne à l'autre. Pour chaque opération effectuée on incrémente la distance de 1. La complexité algorithmique est représentée par la fonction $C(n,m) = (n+1) \times (m+1)$. n et m étants les longueurs des deux chaînes de caractères.

Exemple : la DL entre glass et glace est de 2.

4.5. Méthode d'interprétation des résultats

Le résultat obtenu en sortie de Cine_mind doit être interprété de la manière suivante :

- Si le titre du film recherché est le seul à figurer dans la liste de résultats, alors il y a reconnaissance parfaite de *f* (match).

- Si le titre du film recherché figure dans la liste de résultats et a le score le plus élevé, alors il y a reconnaissance parfaite de *f* (match). Ce qui est notre cas de figure.

- Si le titre du film recherché figure dans la liste d'au plus 4 résultats et n'a pas le score le plus élevé, alors il y a reconnaissance partielle de *f* (partial match). Nous choisissons une marge d'erreur de 4 titres de films, car si ces titres doivent être énoncés verbalement par le chabot, il sera plus simple pour l'utilisateur de retenir cette liste.

- Si le titre du film recherché figure dans la liste de plus de 4 résultats et n'a pas le score le plus élevé, alors il faut considérer que *f* n'est pas reconnu (miss match). Il faut proposer à l'utilisateur une autre méthode de choix.

- Si le titre du film recherché ne figure pas dans la liste, alors c'est un échec de reconnaissance de *f* (miss match). Il faut soit redemander à l'utilisateur de prononcer de nouveau sa requête, soit lui proposer une autre alternative.

4.6. Méthode d'évaluation de Cine_mind

Par manque de données labellisées, nous avons dû évaluer notre programme avec un jeu de données construit par nos soins à partir des requêtes extraites de DialogFlow. 110 requêtes sont labellisées avec le titre du film qu'elles contiennent ou Null si elles n'en contiennent pas. La base de données contient une vingtaine de titre de films qui figurent ou non dans les labels pour tester la robustesse à la multiplication d'homophones et voir si la quantité de données dans la base de données influence l'efficacité de l'algorithme. Nous obtenons une précision de 97% (proportion d'identifications positives effectivement correcte correspondante aux trois premiers points de la section 2.5, « j'ai bien visé ») et un rappel de 95% (proportion des résultats positif correctement identifiée dans les trois premiers points de la section 2.5, « je n'ai rien oublié »). Ces résultats sont assez satisfaisants dans notre cas d'utilisation.

4.7. Remarque

A travers l'exemple choisi, nous voulons faire comprendre au lecteur que la transcription STT n'a pas était correctement réalisée par DialogFlow à cause d'une homophonie. L'utilisateur a prononcé *« Je voudrais aller voir Glass »* et non *« Je voudrais aller voir glace »*. Cette phrase « complexe » est un cas que le précédent chatbot n'était pas capable de traiter. Avec Cine_mind il est maintenant capable de le faire.

4.8. Conclusion

Avec Cine_mind, l'algorithme de reconnaissance « phonétique » d'entités nommées nous avons cherché à corriger les erreurs de transcriptions faites par l'agent DialogFlow. Les résultats obtenus sont très positifs malgré le fait que pour réaliser une meilleure évaluation de Cine_mind, il faudrait un ensemble de données labéllisées plus volumineux.

5. Un Siamese networks (SNN) pour la détection de sentiments en few shot learning

Le fait d'avoir peu de données sur lesquelles travailler nous a fait envisager d'essayer une nouvelle approche appelée Few Shot Learning (FSL). L'enjeu est d'entraîner un réseau de neurones profond avec peu d'exemples. En général cette approche est utilisée en classification d'image. Nous avons choisi de mettre en application les réseaux de neurones siamois ou Siamese networks (SNN) de Leiming et al. [15] pour faire de l'analyse de sentiment en FSL, c'est-à-dire de classer des documents en fonction du sentiment (texte globalement positif, négatif ou neutre) qu'ils dégagent. Le FSL comprend deux sous-catégories qui sont le Zero-Shot Learning (ZSL) et le One-Shot Learning (OSL). L'approche ZSL est l'apprentissage d'une classe sans qu'elle soit représentée dans le jeu de données d'entraînement. L'approche OSL consiste quant à elle, à l'apprentissage d'une classe à partir d'un unique exemple.

5.1. Le plongement de mots [11]

Un réseau de neurones ne peut pas travailler avec des mots représentés par de simple chaînes de caractères. Il faut transformer chaque mot en vecteur. Pour cela il est possible d'utiliser la représentation vectorielle statique ou le plongement de mots (word embedding).

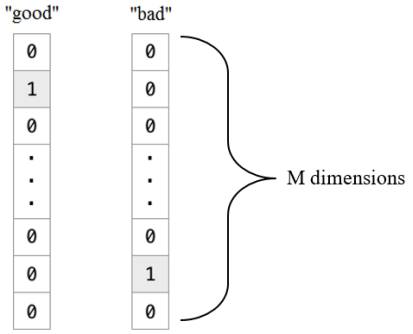


Figure 4 : représentation vectorielle statique de Khuong A. Vo.*

Plusieurs problèmes sont engendrés par la représentation vectorielle statique :

- Un problème de taille. Chaque mot est représenté par un seul bit dans un vecteur de *M* bits.
- Un problème sémantique. La sémantique des mots n'est pas respectée
- Un problème d'ordre. L'ordre des mots dans une phrase n'est pas conservé.

Pour pallier à ces problèmes il existe plusieurs méthodes de plongement de mots. Leiming et al. [15] ont décidé d'utiliser la méthode nommée Word2Vec. Le modèle utilise un réseau de neurones peu profond (shallow neural network) à deux couches cachées pouvant avoir deux architectures : le sac de mots continu (continuous bag-of-words, CBOW) qui permet de prédire la représentation vectorielle *w(t)* du mot courant grâce à son contexte (c'est-à-dire grâce aux vecteurs *w(t-i)* des mots qui l'entoure) et le Skip-gram qui permet de prédire les mots qui entourent le mot courant. Ici nous utiliserons le résultat d'un modèle CBOW déjà pré-entraîné [16] pour sa capacité à déterminer la forme vectorielle d'un mot grâce à son contexte.

** https://www.researchgate.net/figure/Representing-words-by-one-hot-vector_fig1_328161921*

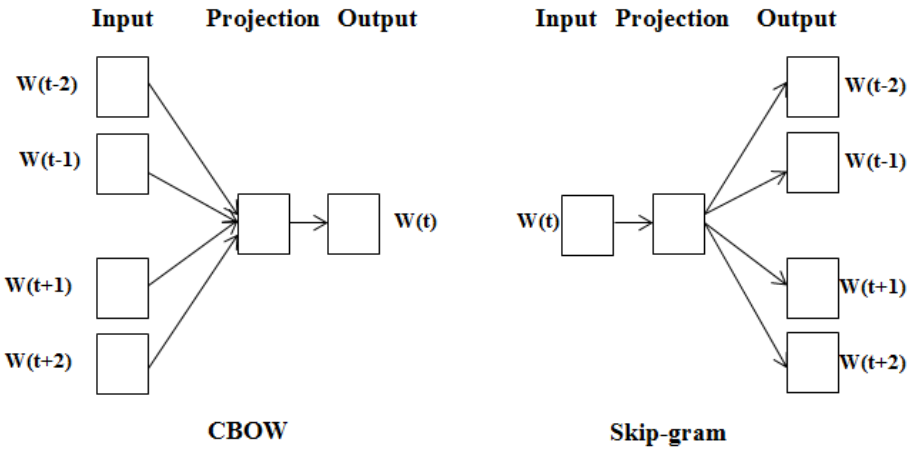


Figure 5 : Fonctionnement des architectures CBOW et Skip-gram extraite de [11].

C'est un dictionnaire encodé au format binaire qui permet de transformer chaque mot d'une phrase en un vecteur de 300 flottants *w*. A partir de ces vecteurs, nous formons une matrice *P* de forme *(64 x w(t))* représentant la phrase globale. Si cette matrice ne contient pas au moins 64 lignes, alors sera complétée avec des vecteurs *w* nuls. Chaque matrice *P* est labellisée en fonction de son sentiment général (positive, neutre ou négative). La forme matricielle peut être aisément traitée par le sous-réseau de convolution présenté dans la prochaine section.

5.2. Le sous réseau de convolution (CNN)

Le CNN permet d'obtenir un vecteur représentatif de *P* réencodé dans une dimension réduite et de trouver quel est le sentiment global de *P*. Il est composé, en entrée de trois couches de convolution parallèles. Ces trois couches possèdent chacune une taille différente de filtre de convolution : $(3,4,5) \times 300$. Leiming et al. choisissent la fonction Relu ($f(x) = \max(0, x)$) comme fonction d'activation pour les trois couches. Une opération de max-pooling est ensuite appliquée à la matrice obtenue par convolution pour extraire la valeur maximum. Quand les valeurs maximales ont été extraites, une couche compacte permet de les concaténer en un seul vecteur. Un classifieur softmax (fonction exponentielle normalisée) est ajouté pour réaliser la propagation arrière (modification des poids des arcs en partant de la sortie du réseau de neurones) lors de la phase de préentraînement. Les hyperparamètres conseillés pour le pré entraînement sont un taux d'apprentissage de 0.95, un lot de 50 exemples, un abandon de 50% sur 25 itérations.

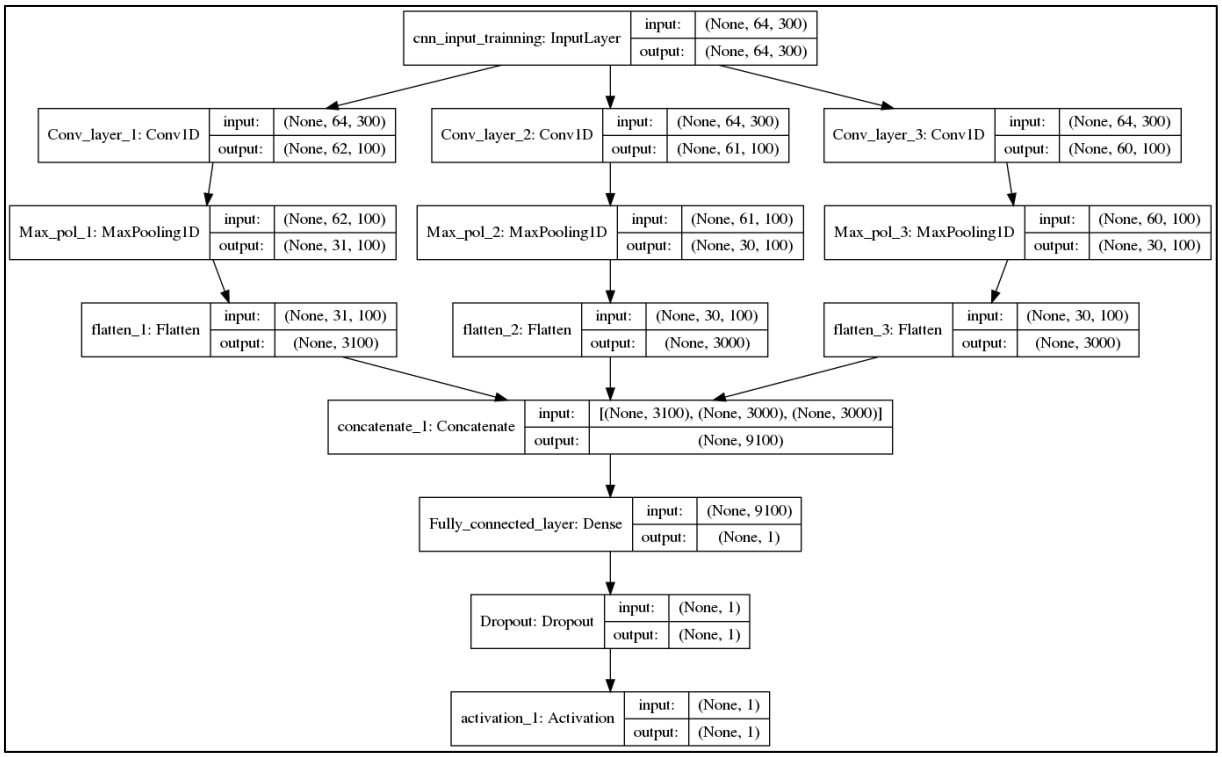


Figure 6 : architecture du CNN.

5.3. Le Siamese Neural Network (SNN)

On appelle cette architecture Siamese neural network (SNN) ou réseaux siamois car ce sont deux réseaux de neurones identiques qui sont mis en parallèle (annexe ii et iii). Dans notre SNN, c'est le CNN présenté précédemment qui est parallélisé. Le SNN a donc deux entrées auxquelles nous soumettons un lot de deux matrices distinctes (*P1* et *P2*) sélectionnées dans le même jeu de données d'entraînement que le CNN. Le label du lot indique si le sentiment de P_1 et P_2 est identique ou non. C'est ici qu'intervient la notion de FSL car la taille du jeu de données d'entraînement est alors divisée par 2. Le SNN doit apprendre à déterminer si les classes des deux phrases d'entrées ont un sentiment similaire ou non. L'architecture du CNN subit une légère modification lors de son intégration au SNN, le classifieur softmax est remplacé par un séparateur à vaste marge (SVM) pour une question d'équilibre entre classes. Nous ajoutons une loss fonction (fonction économique) hinge pour apprendre la représentation discriminative :

$$h(x_i - x_j) = \begin{cases} \|x_i - x_j\| & if\ l_i = l_j \\ \max\left(0, m - \|x_i - x_j\|^2\right) & if\ l_i \neq l_j \end{cases} \qquad \text{(eq1)}$$

ou *l* est le label du vecteur texte *x* réencodé par le CNN. Elle permet de faire le rapprochement entre deux vecteurs ayant la même classe et de séparer les vecteurs d'une valeur d'au moins *m* quand ils sont de classe différente. La fonction sigmoïde est ensuite ajoutée pour d'obtenir un résultat compris entre 0 est 1. Quand P_1 et P_2 ont le même label (Les deux phrases d'entrées ont le même sentiment) alors le lot est annoté 1, sinon 0. Les hyperparamètres sont les mêmes que lors de la phase de pré entraînement.

5.4. Méthode d'évaluation du SNN

Après avoir implémenté le SNN à l'aide de l'API Keras [17], il est entraîné et évalué à l'aide de deux jeux de données composés de tweets en langue anglaise labélisés avec leur sentiment : le Health Care Reform (HCR) [18] et le Twitter US Airline Sentiment (TUAS) [19]. Chaque tweet est segmenté en mots et filtré avec l'API de traitement automatique du langage Spacy [20] puis transformé en matrice *P*. Pour rappel, on calcule la similitude entre les sentiments de P_1 et P_2. Le tableau 1 en fin de section résume les résultats obtenus à chaque modification apportée à l'algorithme.

En travaillant avec le jeu de données HCR (1280 exemples de train et 480 de test) et les hyperparamètres proposés par défaut par Leiming et al., le SNN obtient des résultats médiocres (1).

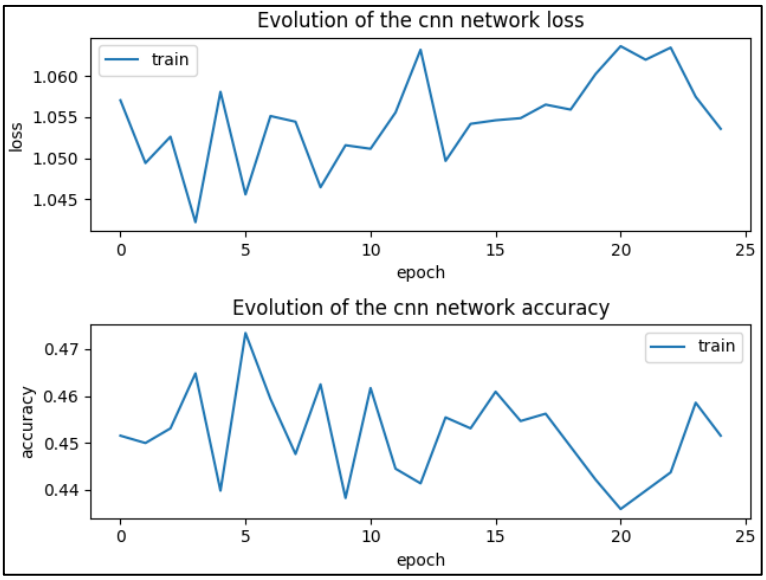


Figure 7 : résultats du CNN obtenus avec le jeu de données HCR et les paramètres par défaut (1).

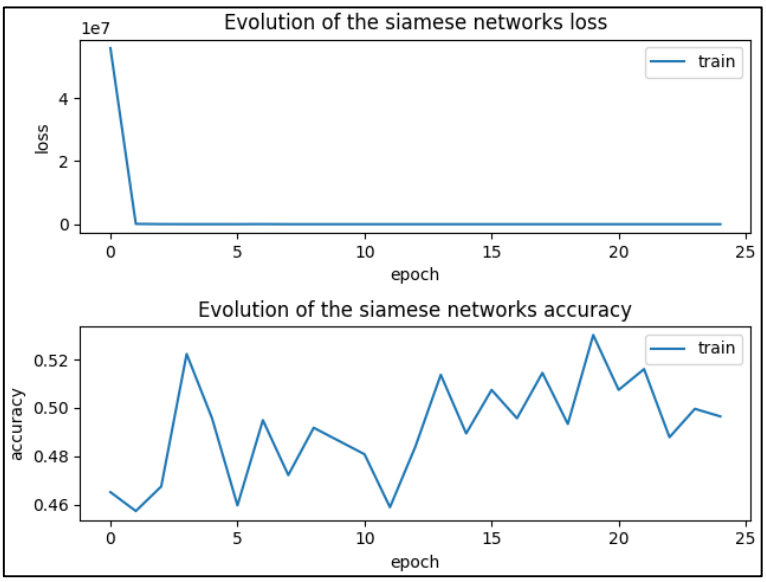


Figure 8 : résultats du SNN obtenus avec le jeu de données HCR et les paramètres par défaut (1).

Nous avons un problème de sous apprentissage. Nous diminuons alors le taux d'apprentissage à 0,001, valeur proposée par défaut dans la documentation Keras, pour résoudre ce problème (2).

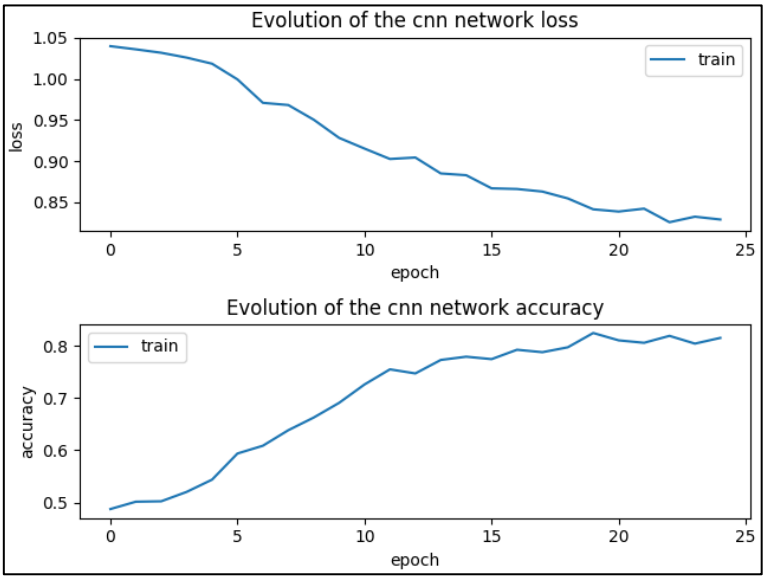


Figure 9 : résultats du CNN obtenus avec le jeu de données HCR et un taux d'apprentissage plus faible (2).

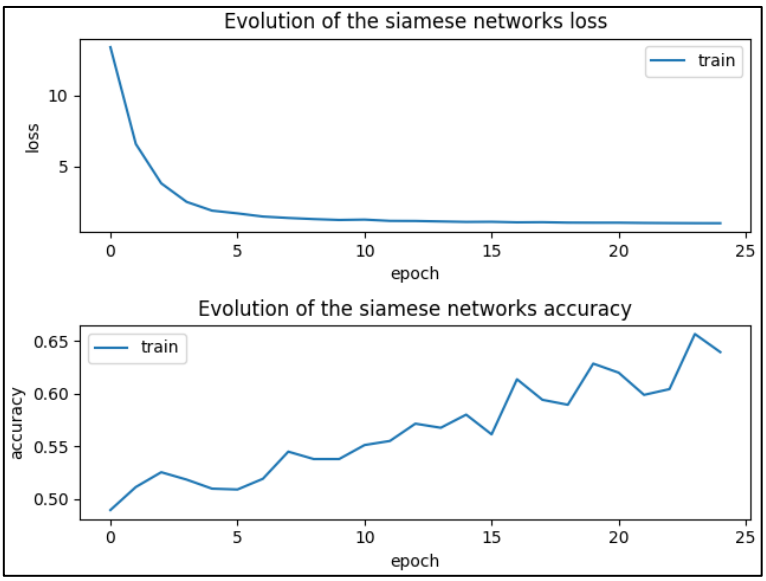


Figure 10 : résultats du SNN obtenus avec le jeu de données HCR et un taux d'apprentissage plus faible (2).

Nous constatons maintenant un surapprentissage. Pour le corriger nous implémentons une fonction permettant de faire de l'arrêt précoce (early stopping), c'est-à-dire d'arrêter l'apprentissage lorsque le paramètre surveillé (ici la loss de validation) atteint une certaine valeur (3).

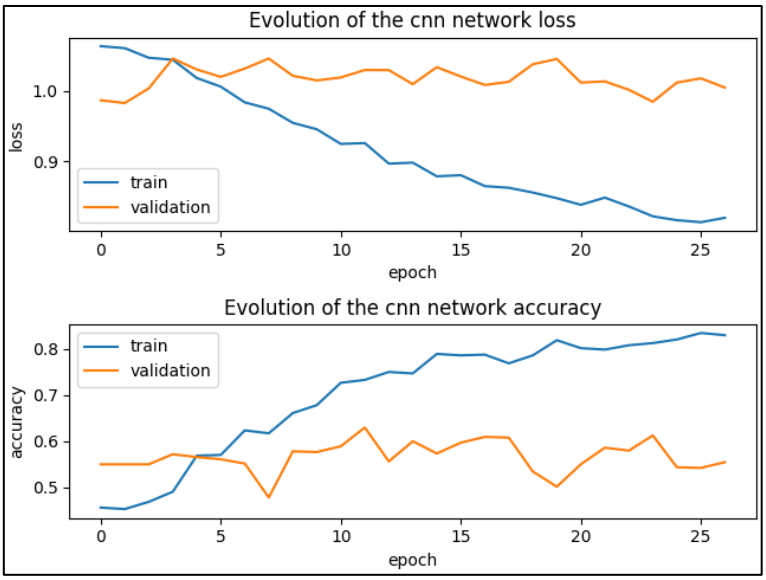


Figure 11 : résultats du CNN obtenus avec le jeu de données HCR et early stopping (3).

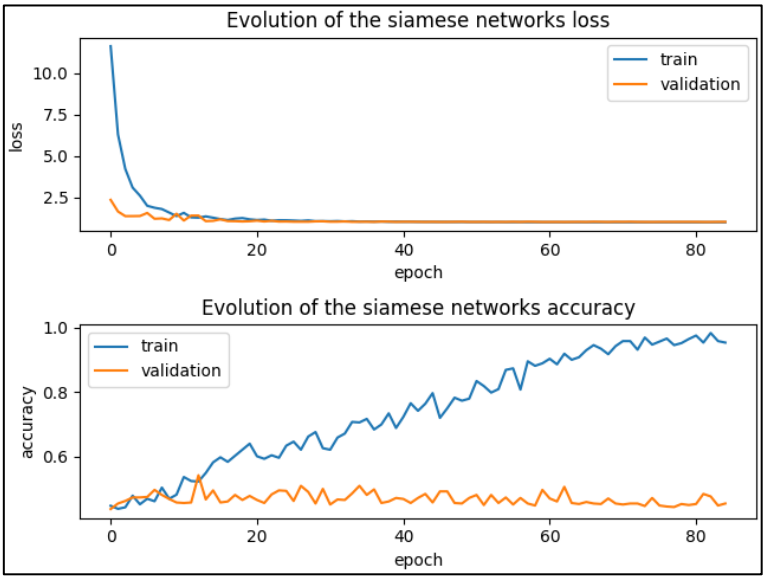


Figure 12 : résultats du SNN obtenus avec le jeu de données HCR et early stopping (3).

Nous pouvons constater une diminution du surapprentissage, notamment pour notre CNN. La courbe de la fonction objectif (loss function) débute plus bas que celle de la phase d'entraînement car lors de la phase de validation, l'abandon (dropout) n'est pas pris en compte. Le nombre d'exemples d'évaluation influence aussi la position du point de départ de cette courbe. Nous utilisons ensuite le jeu de données TUAS qui contient plus de données (2438 exemples d'entraînement et 2439 de test) pour savoir si l'augmentation du nombre d'exemples d'entraînement influence la qualité de l'apprentissage et les résultats de prédiction (4).

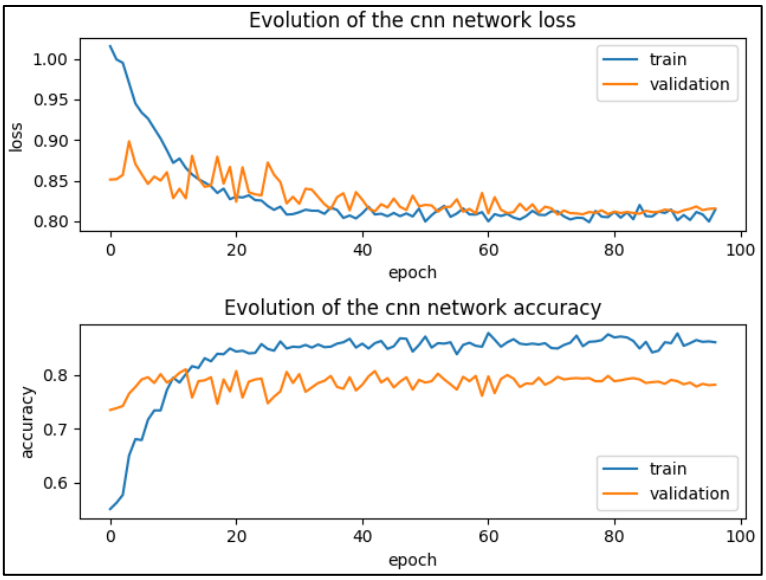


Figure 13 : résultats du CNN obtenus avec le jeu de données TUAS (4).

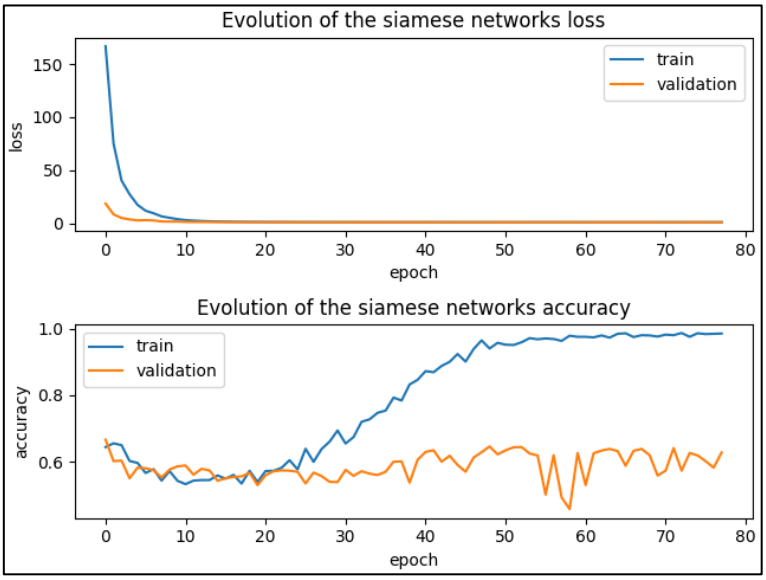


Figure 14 : résultats SNN obtenus avec le jeu de données TUAS (4).

Une amélioration est visible dans les prédictions réalisées par notre SNN. Or, en augmentant le nombre d'exemples dans notre jeu de données nous nous éloignons du but principal qui est de faire apprendre notre réseau de neurones des classe *sentiments* à l'aide de peu d'exemples.

			loss	mae	accuracy
(1)	CNN	Entraînement	0.998958	0.332292	0.501562
		Test	0.994444	0.327778	0.508333
	SNN	Entraînement	1.036040	0.492799	0.569195
		Test	2.310810	0.500740	0.498956
(2)	CNN	Entraînement	0.707305	0.040638	0.977344
		Test	0.988859	0.322192	0.579167
	SNN	Entraînement	0.996022	0.484973	0.826427
		Test	1.008365	0.500482	0.486430
(3)	CNN	Entraînement	0.848996	0.182330	0.771875
		Test	0.988772	0.322106	0.620833
	SNN	Entraînement	0.997591	0.486057	0.724785
		Test	1.006639	0.499432	0.511482
(4)	CNN	Entraînement	0.742793	0.076126	0.888433
		Test	0.837326	0.170659	0.749897
	SNN	Entraînement	0.991695	0.483627	0.813705
		Test	0.999317	0.497354	0.601970

Tableau 1 : résultats des expériences.

5.5. Discussion

Malgré de nombreux essais, nous n'avons pas pu reproduire correctement l'expérience et obtenir des résultats similaires à ceux de Leiming et al.. Cela peut provenir de la façon dont l'expérience a été reconstituée :

- Nous n'avons pas travaillé à partir des mêmes jeux de données, car ils sont difficiles à retrouver.
- Nous n'avons pas d'information sur le langage, ni sur les APIs utilisées dans le l'article de Leiming et al..
- Nous avons établi des modifications pour réduire certains biais comme le surapprentissage.
- Nous utilisons un SVM non sensible au coût et peut être mal placé dans l'architecture.
- Nous avons mal interprété le critère d'évaluation de [15] en fin d'introduction du §5.

5.6. Conclusion

Nous n'avons pas obtenu de résultats convenables en mettant en application le SNN de Leiming et al. pour faire de la comparaison de sentiment avec peu de données. Nous pouvons nuancer cet échec de manière positive car les scores réalisés par le CNN sont plutôt acceptables pour du FSL.

6. Conclusion générale

Lors de ce stage au Lab-STICC en partenariat avec le crédit mutuel ARKEA, nous avons pu mettre en place un système prometteur de reconnaissance d'entités nommées pour détecter et corriger des titres de films à l'intérieur d'une requête vocale transcrite sous forme textuelle.

A cause d'une quantité limitée de ces données, nous avons pensé à étudier un système pour résoudre ce genre de problématique. Le Siamese Networks (SNN) mis en place pour faire de la comparaison de sentiment en FSL n'a pas porté ses fruits. Nous pouvons tout de même tempérer cet échec car le CNN employé dans le SNN retourne des résultats plutôt positifs pour de l'apprentissage avec peu de données.

Grâce à ce stage, nous avons pu travailler sur des problématiques actuelles qui demandent encore d'être étudiées et améliorer notre savoir et nos compétences en apprentissage automatique, plus particulièrement en apprentissage profond (deep learning).

Annexes

i. Code de Cine_mind

https://nuxeo.enib.fr/nuxeo/nxdoc/default/b7307731-9c94-40ad-9510-48507f814104/view_documents

Les fichiers sont stockés dans un fichier zip et chiffré à l'aide de GNU Privacy Guard (gpg). Le mot de passe de déchiffrage est : *Cine-Siamese,20*.

Pour assurer le bon fonctionnement du programme, penser à modifier les chemins en dur. Ne pas hésiter à jouer les variables lors des essais.

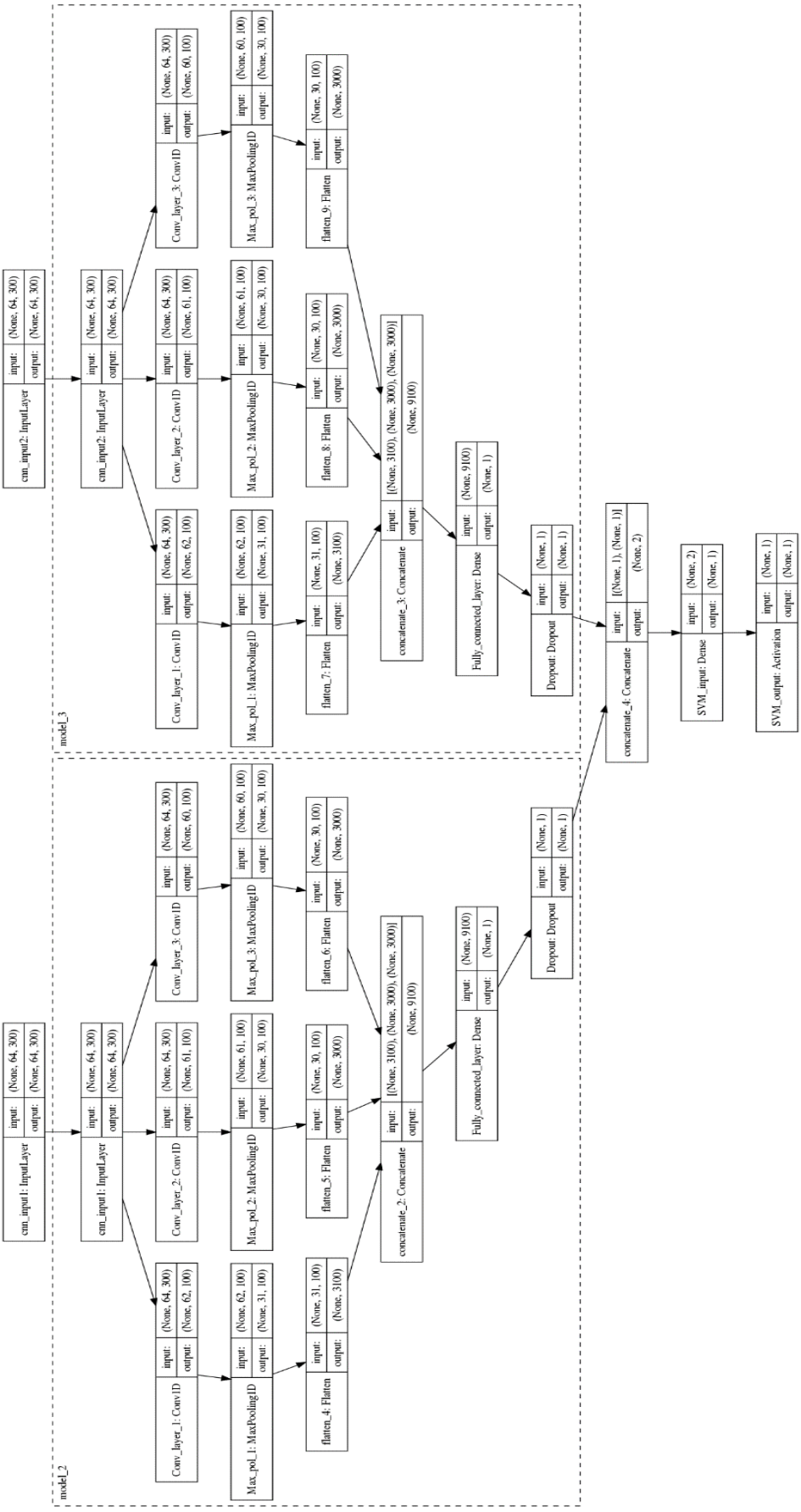
ii. Code du SNN

https://nuxeo.enib.fr/nuxeo/nxdoc/default/85bb8ca2-6afb-4d59-ba96-0f120d720a76/view_documents

Les fichiers sont stockés dans un fichier zip et chiffré à l'aide de GNU Privacy Guard (gpg). Le mot de passe de déchiffrage est : *Cine-Siamese,20*.

Pour assurer le bon fonctionnement du programme il faut télécharger le dictionnaire word2Vvec [16] et modifier les chemin en dur. Ne pas hésiter à jouer les variables lors des essais.

iii. Architechture du SNN



Références

[1] Dekimasu, « Assistant personnel intelligent », *Wikipédia*, déc. 15, 2019. https://fr.wikipedia.org/w/index.php?title=Assistant_personnel_intelligent&oldid=165436420 (consulté le janv. 10, 2020).
[2] L. Julia et O. Khayat, *L'intelligence artificielle n'existe pas*, First. Paris, France, 2019.
[3] A. Niculescu et R. Banchs, « Strategies to cope with errors in human-machine spoken interactions: using chatbots as back-off mechanism for task-oriented dialogues », in *ERRARE 2015*, Sinaia, Roumania: Romanian Academy, 2016.
[4] L. Bradesko et D. Mladenic, « A Survey of Chabot Systems through a Loebner Prize Competition », présenté à Eighth Conference of Language Technologies, Jožef Stefan Institute, Ljubljana, Slovenia, oct. 2012, p. 34-37.
[5] Z. Kou, D. Stanton, F. Peng, F. Beaufays, et T. Strohman, « Fix it where it fails: Pronunciation learning by mining error corrections from speech logs », in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, South Brisbane, Queensland, Australia, avr. 2015, p. 4619-4623, doi: 10.1109/ICASSP.2015.7178846.
[6] J. Hui, « Speech Recognition — Acoustic, Lexicon & Language Model », *Medium*, sept. 22, 2019. https://medium.com/@jonathan_hui/speech-recognition-acoustic-lexicon-language-model-aacac0462639 (consulté le janv. 05, 2020).
[7] Google, « Google Code Archive - Long-term storage for Google Code Project Hosting. » https://code.google.com/archive/p/sxse/ (consulté le déc. 31, 2019).
[8] C. Cabot, S. Darmoni, et L. F. Soualmia, « Cimind: A phonetic-based tool for multilingual named entity recognition in biomedical texts », *J. Biomed. Inform.*, vol. 94, p. 103176, juin 2019, doi: 10.1016/j.jbi.2019.103176.
[9] « Reconnaissance d'entités nommées », *Wikipédia*, janv. 10, 2020. https://fr.wikipedia.org/w/index.php?title=Reconnaissance_d%27entit%C3%A9s_nomm%C3%A9es&oldid=166226287 (consulté le avr. 14, 2020).
[10] Google, « Dialogflow ». https://dialogflow.com/ (consulté le avr. 14, 2020).
[11] Kowsari, Jafari Meimandi, Heidarysafa, Mendu, Barnes, et Brown, « Text Classification Algorithms: A Survey », *Information*, vol. 10, nº 4, p. 150, avr. 2019, doi: 10.3390/info10040150.
[12] « Metaphone », *Wikipédia*, févr. 25, 2002. https://en.wikipedia.org/wiki/Metaphone#Double_Metaphone.
[13] O. Margaret King et R. Robert, « Soundex », *Wikipédia*. https://en.wikipedia.org/wiki/Soundex.
[14] « Distance de Levenshtein », *Wikipédia*, févr. 27, 2020. http://fr.wikipedia.org/w/index.php?title=Distance_de_Levenshtein&oldid=167861974 (consulté le mars 06, 2020).
[15] L. Yan, Y. Zheng, et J. Cao, « Few-shot learning for short text classification », *Multimed. Tools Appl.*, vol. 77, nº 22, p. 29799-29810, nov. 2018, doi: 10.1007/s11042-018-5772-4.
[16] eyaler, « eyaler/word2vec-slim », *GitHub*, avr. 29, 2020. https://github.com/eyaler/word2vec-slim (consulté le mai 06, 2020).
[17] F. Chollet, « Keras », 2015. https://keras.io/ (consulté le avr. 30, 2020).
[18] M. Speriosu, « speriosu / updown / data / hcr », *Bitbucket*. https://bitbucket.org/speriosu/updown/src/default/data/hcr/ (consulté le avr. 30, 2020).

[19] « Twitter US Airline Sentiment », *Kaggle*. https://kaggle.com/crowdflower/twitter-airline-sentiment (consulté le avr. 30, 2020).
[20] « spaCy · Industrial-strength Natural Language Processing in Python », *spaCy*. https://spacy.io/ (consulté le mai 06, 2020).