



HAL
open science

Topic clustering within chatbots

Riade Benbaki, Haolin Pan, Pierre Clarou, Alexandre Thomas

► **To cite this version:**

Riade Benbaki, Haolin Pan, Pierre Clarou, Alexandre Thomas. Topic clustering within chatbots. [Rapport de recherche] École polytechnique. 2020. hal-03090625

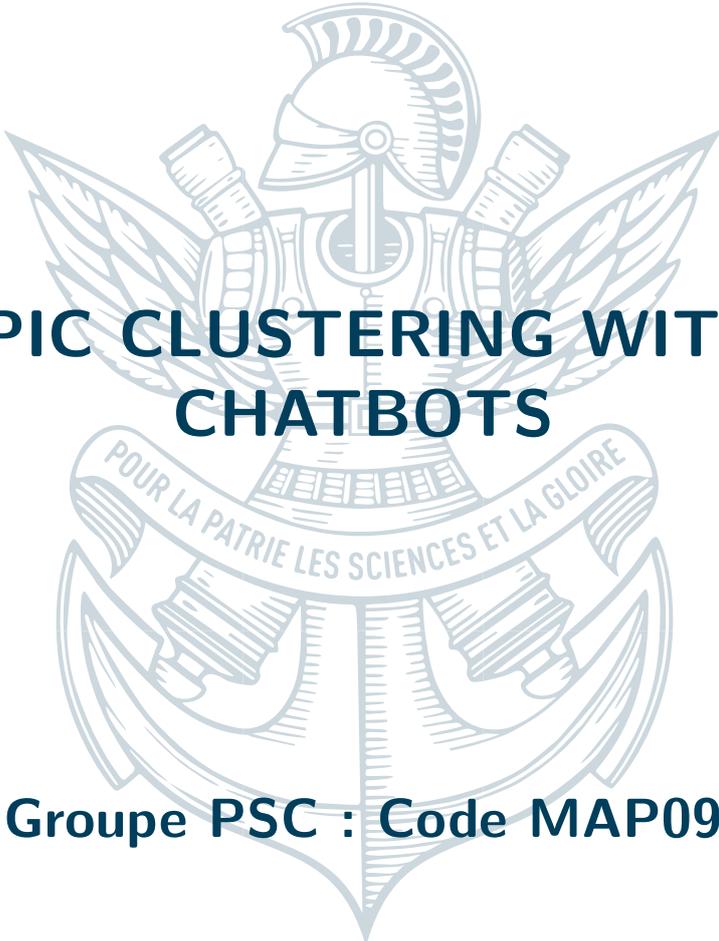
HAL Id: hal-03090625

<https://hal.science/hal-03090625>

Submitted on 29 Dec 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



TOPIC CLUSTERING WITHIN CHATBOTS

Groupe PSC : Code MAP09

13 avril 2020

Riade Benbaki, Haolin Pan, Pierre Clarou, Alexandre Thomas



Résumé

Nous commençons le projet en explorant un grand nombre de méthodes NLP qui s'apparentent au *Topic Clustering*. Ensuite, nous choisissons parmi ces techniques une qui nous semble la plus prometteuse, à savoir les *Contextualised Word Embeddings*. Dans notre projet, nous nous intéressons plus particulièrement à *ELMo*, que l'on compare à d'autres modèles similaires en terme de performance. En optimisant les hyper-paramètres des modèles construits, nous obtenons un *f1-score* supérieur de 0.9.

Nous poussons l'étude un peu plus loin, en examinant non seulement la détection de sujet, mais aussi la génération d'une réponse adéquate, avec un mécanisme d'attention particulièrement efficace dans le cas d'un utilisateur non-coopératif. Enfin, nous étudions comment les 3 modèles décrits peuvent être combinés pour répondre à la problématique du projet.

Mots-Clés : Topic Clustering, Intents Detection, ELMo, REDP, Neural Turing Machine.

TABLE DES MATIÈRES

1	Introduction	4
2	Étude théorique	6
2.1	Algorithme naïf	6
2.2	Algorithmes basiques de clustering	6
2.3	Allocation Latente de Dirichlet	7
2.4	Allocation Sémantique Latente	7
2.5	GPT2	8
2.6	Neural Network Language Model, NNLM	9
2.7	Doc2Vec	10
2.8	Embedding Contextualisé	10
2.8.1	Universal Sentence Encoder	10
2.8.2	ELMo	11
3	Modèles construits	12
3.1	Dataset	12
3.2	Détection de sujet	13
3.2.1	Modèle	13
3.2.2	Expérience	13
3.2.3	Résultats	14
3.3	Détection d'intention	15
3.3.1	Modèle	15
3.3.2	Expériences	16
3.3.3	Analyse de résultats	17
3.4	Génération des réponses	19
3.4.1	Mécanisme d'attention : Recurrent Embedding Dialogue Policy	19
3.4.2	Neural Turing Machine	20
3.4.3	Score de Bahdanau	22
3.4.4	REDP	23
3.4.5	Comparaison avec un LSTM	24
3.5	Intégration des différents modèles	26
4	Conclusion	28

1

INTRODUCTION

Aujourd'hui, de plus en plus d'entreprises cherchent à automatiser les discussions avec leurs clients, afin d'être capables de traiter avec énormément de gens à la fois sans qu'il y ait besoin d'un gros service autour. C'est pour cela que les chatbot ont été créés : Ce sont des algorithmes permettant de « discuter » avec une ou plusieurs personnes pour répondre à leurs questions. Il est possible d'en dégager deux types :

Le premier type, qui nous intéressera peu durant ce PSC, consiste en le Hard-codage de « mots clef » au sein du code du chat bot, puis de générer des réponses en fonction des mots clef détectés. Il n'y a donc aucune évolution dans le comportement du Bot tant que son code est le même, de plus ses réponses sont déterministes et calculables, le code s'apparentant à un grand arbre de décision. Le second type commence à se développer grâce aux récents progrès techniques et notamment l'augmentation de la mémoire et de la vitesse de calcul, ainsi qu'à la quantité de données rassemblées de plus en plus grande. Les chatbot commencent à fonctionner grâce à des intelligences artificielles plus poussées, et notamment de l'apprentissage profond. Cela permet au Bot de progresser pour s'exprimer et répondre de plus en plus correctement.

Cependant un problème majeur subsiste et différencie clairement la machine de l'Homme. L'Homme est capable de détecter facilement un changement de sujet alors que les chatbot non. Dans certains cas ce n'est pas un problème, notamment lorsque le sujet traité est très restreint (ex : Description d'un objet vendu unique) et en cas d'incompréhension il est possible de demander à l'utilisateur de répéter sa question. Cependant, dans certains cas le Bot peut se retrouver incapable de trouver le sujet d'une phrase. Par exemple :

User : *Bonjour je souhaiterais savoir quelles sont les avantages de cet abonnement téléphonique*
Bot : Décrit les avantages demandés.
User : *Y a-t-il des suppléments ?*
Bot : Décrit les suppléments disponibles.
User : *Merci, combien cela va-t-il coûter ?*

TABLE 1 – Exemple de discussion

Dans ce cas précis, le Bot peut se trouver coincé, il ne sait pas sur quoi porte le mot « cela ».

Est-ce, l'abonnement, ou le prix des suppléments. Un Homme peut ici prendre la décision en fonction du contexte (par exemple « Est-ce que l'utilisateur a déjà demandé le prix de l'abonnement auparavant ou non? »), alors qu'un chatbot aura énormément de mal et risque de répondre une mauvaise réponse. Dans ce cas particulier il est possible que le bot réponde en donnant la liste de tous les prix pour être sûr de ne pas se tromper. Cependant cette possibilité n'est pas toujours viable surtout lors de conversations longues dans lesquelles beaucoup de

sujets sont abordés.

Notre PSC consiste à étudier ce qu'on appelle le Topic Clustering. Notre objectif est de rechercher un algorithme capable de délimiter les différents sujets dans une conversation Machine-Machine ou Homme-Machine. Dans cette partie, l'étiquetage de ces sujets n'est pas nécessaire (Topic Labelling). Ainsi un algorithme capable de délimiter un changement de sujet sans forcément préciser de quel sujet il s'agit est tout à fait acceptable. Dans une deuxième partie, si le ou les algorithmes trouvés ne sont pas capables d'étiqueter correctement les sujets, nous pourrions dû nous atteler à la recherche d'un étiquetage pour lesdits sujets.

Pour terminer, notre dernière possibilité s'il nous reste encore du temps, est de mettre en place un petit site sur lequel on pourrait tester un chatbot utilisant nos algorithmes de Topic Clustering.

2

ÉTUDE THÉORIQUE

Notre première étape a été d'étudier plusieurs techniques de NLP qui peuvent servir à réaliser les tâches de *Text Clustering* ou de *Text Labelling*. Nous présentons dans cette section toutes les techniques envisagées, pour ensuite se concentrer sur une qui nous a semblé la plus prometteuse, qu'on implémente en pratique.

2.1 ALGORITHME NAÏF

Nous avons étudié l'algorithme le plus naïf. Grâce aux propriétés du Word Embedding il est possible d'étudier la proximité d'un mot avec un autre en regardant le produit scalaire entre les deux mots. Plus ce produit est élevé, plus les mots sont proches. Pour cet algorithme, il faut définir à l'avance une liste de sujets. Le principe est ici, pour chaque phrase, de chercher le sujet prédéfini, le plus proche en moyenne des mots de la phrase. Cependant cela pose plusieurs problèmes. L'algorithme a beaucoup de difficultés à séparer des sujets assez proches et détaillés. De plus le choix initial des sujets force quelqu'un à étudier le document à l'avance pour pré dégager les sujets principaux à détecter. Enfin le dernier problème est que l'algorithme ne prend jamais en compte du contexte. Cela fait perdre à l'algorithme énormément d'efficacité.

2.2 ALGORITHMES BASIQUES DE CLUSTERING

La deuxième idée est de se servir de l'espace des mots créé par le Word Embedding et en particulier de travailler sur l'ensemble des mots du document en même temps. Il y a pour cela deux algorithmes potentiellement efficaces : Le K-means[2] et le DBscan. Le principe de K-means est de placer K centroïdes aléatoirement, d'associer chaque point à son centroïde le plus proche, et enfin de déplacer les centroïdes au centre des clusters formés. En répétant les deux dernières étapes, les centroïdes finissent par se stabiliser et donnent ainsi des thèmes et des mots proches de ce thème. Le mot le plus proche du centroïde définit ainsi le thème et tous les mots proches de ce sujet sont étiquetés par ce sujet.

Le DBscan cherche lui à détecter les zones de forte densité pour créer les clusters autour de celles-ci. Cet algorithme crée donc des clusters dans lesquels deux mots ne sont jamais très éloignés. Les problèmes du K-means et du DBscan sont, cependant, nombreux. Il faut, pour le K-means, donner le nombre de sujets à l'avance alors que cela dépend à priori du document alors que le DBscan a besoin d'une distance limite entre deux mots du même cluster. De plus, les algorithmes ne tiennent pas compte des phrases ni de l'ordre des mots ce qui est assez problématique et donne des résultats faux souvent.

2.3 ALLOCATION LATENTE DE DIRICHLET

Le premier algorithme plus poussé de clustering non supervisé que nous avons trouvé est le LDA (Latent Dirichlet Allocation, ou allocation latente de Dirichlet)[3][4]. Cet algorithme associe à chaque mot d'un document un des K thèmes proposés à l'avance selon une distribution de Dirichlet, puis de modifier le thème de certains mots en fonction des thèmes de tous les autres mots.

A chaque étape, l'algorithme traite une fois chaque mot. Pour un mot donné, le principe est de supposer le sujet de tous les autres mots juste, calculer les probabilités $\mathbb{P}(t|d) * \mathbb{P}(w|t)$ c'est-à-dire la probabilité $\mathbb{P}(t|d)$ que le document d soit assigné au thème t multiplié par $\mathbb{P}(w|t)$ la probabilité que le thème t dans le corpus soit assigné au mot w . Ce produit donne la probabilité que le thème t soit assigné au mot w dans le document d . L'algorithme calcule cela pour tous les thèmes t et assigne au mot w un thème t aléatoirement suivant les pondérations calculées précédemment.

Après plusieurs étapes, les thèmes se stabilisent et donnent le sujet associé à chaque mot. Cependant les résultats obtenus risquent d'être assez insatisfaisant d'une part car tous les mots sont classés et le temps de calcul peut être assez long, et d'autre part car nous avons besoin du document complet afin de travailler alors qu'un chat bot devrait être capable de classer les sujets à chaque phrase. Or dans ce cas, une phrase de plus pourrait potentiellement ajouter un sujet ou remettre en question les sujets calculés précédemment.

2.4 ALLOCATION SÉMANTIQUE LATENTE

Nous avons ensuite étudié l'allocation sémantique latente (LSA)[3][4]. L'algorithme se sert de l'importance des mots dans les phrases pour dégager des thèmes.

Il commence par construire une matrice $X = \begin{pmatrix} x_{1,1} & \dots & x_{1,n} \\ \vdots & \ddots & \vdots \\ x_{m,1} & \dots & x_{m,n} \end{pmatrix}$ appelée matrice des occurrences dans laquelle $X_{i,j}$ contient le nombre d'occurrences du terme i dans la phrase (ou document) j . La colonne j est appelée $C_j = \begin{pmatrix} x_{1,j} \\ \vdots \\ x_{m,j} \end{pmatrix}$ et la ligne i , $L_i^T = (x_{i,1} \dots x_{i,n})$. Cette matrice

est ensuite décomposée en valeurs singulières (décomposition SVD).

$$X = U\Sigma V$$

$$U = \left(\begin{pmatrix} \mathbf{u}_1 \end{pmatrix} \dots \begin{pmatrix} \mathbf{u}_l \end{pmatrix} \right) \quad \Sigma = \begin{pmatrix} \sigma_1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \sigma_l \end{pmatrix} \quad V = \begin{pmatrix} \mathbf{v}_1 \\ \vdots \\ \mathbf{v}_l \end{pmatrix}$$

$$\hat{l}_i^T = ((\mathbf{u}_1)_i \dots (\mathbf{u}_l)_i) \quad \hat{c}_i = \begin{pmatrix} (\mathbf{v}_1)_i \\ \vdots \\ (\mathbf{v}_l)_i \end{pmatrix}$$

L'algorithme sélectionne ensuite les k plus grandes valeurs singulières ainsi que les vecteurs singuliers associés. Grâce à cela, les colonnes \hat{c}_i des matrices ainsi obtenues nous permettent de voir l'importance du mot i dans chacun des k sujets obtenus. Les vecteurs lignes et colonnes précédents, c'est-à-dire les vecteurs « sujet » et « phrase » sont traduits dans un nouvel espace de « concepts ».

Cela nous permet globalement d'étiqueter les différents sujets obtenus aux parties du document, et ainsi de séparer et d'étiqueter les sujets dans toutes les phrases du document et même dans des parties de phrases. Cependant l'algorithme présente plusieurs problèmes. Le LSA se sert du principe du sac de mot et en récupère donc tous les inconvénients. Le plus problématique est la perte de l'ordre[1]. Les mots sont rangés dans une matrice d'occurrences et l'ordre dans les phrases ou dans le document n'est pas très bien retenu alors qu'il semble fortement influencer sur le sens des phrases. De plus le sac de mots n'est pas capable de gérer les problèmes de polysémie. Pour lui, un mot n'a qu'un seul sens. Cela peut donc totalement fausser les résultats obtenus dans le cas d'une apparition d'un mot avec plusieurs sens à plusieurs endroits du document ou de la même phrase.

2.5 GPT2

Une autre piste étudiée est GPT2, un algorithme développé par openAI[5]. Cet algorithme est capable de détecter les sujets dans une ou plusieurs phrases pour en écrire la suite. Cependant, si l'algorithme se limite à cela, il ne nous sera pas très utile car il n'est capable de travailler que sur une ou un petit nombre de phrases alors que séparer des sujets dans des conversations peut nécessiter de travailler sur beaucoup plus de phrases.

L'exemple ci-dessous nous permet de montrer la faiblesse majeure de GPT2 : l'algorithme a des difficultés à traiter la totalité du texte. Nous pouvons voir ici que seule la dernière phrase est prise en compte. L'algorithme se concentre sur la dernière phrase pour imaginer la suite de la conversation. Dans des conversations Humain/Chatbot pouvant durer plus longtemps, cet

User : *Hello, I want to get some informations about the new mobile phone you released*
 Bot : *We release our new product with a new bundle*
 User : *Interesting! How much will it cost ?*
 Bot : *Enough to hire a security guy to keep me safe.*
 User : *Wait a minute. I'm done, give me aall the money now!*
 Bot : *Listen to me well. Even if you have the money, there is no way I would take any help from you*
 User : *Are you saying that you don't want help ?*
 Bot : *I will never leave your side. Please give me what I want, for nothing less*
 User : *I'll help you to the best of my ability*
 Bot : *I will not leave your side. Please give.*

TABLE 2 – Exemple de discussion

algorithme "oublie" une partie de la conversation et par conséquent peut s'avérer inefficace pour détecter des changements de sujet.

2.6 NEURAL NETWORK LANGUAGE MODEL, NNLM

NNLM [9] est un algorithme de *Word Embedding* basé sur un réseau de neurones à trois couches, une couche **Input**, une couche cachée et une couche **Output**. La couche **Input** convertit les mots en entrées en $n - 1$ vecteurs de mots compact de basse dimension. La couche cachée est une couche avec **tanh** pour fonction d'activation. Enfin, la couche **Output** utilise une fonction **softmax** pour transformer les vecteurs en probabilités.

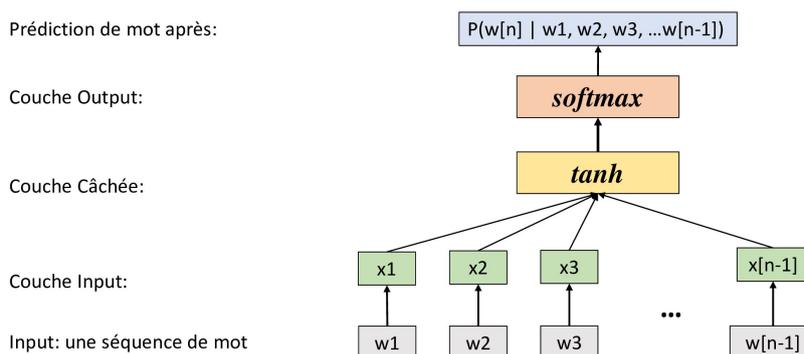


FIGURE 1 – Structure du NNLM

Pour une séquence de mots de longueur n dans le corpus, $\omega_{i-(n-1)}, \dots, \omega_i$, la fonction objectif du réseau de neurones NNLM est $P(\omega_i | \omega_{i-(n-1)}, \dots, \omega_{i-1})$

Finalement, après avoir entraîné le modèle, on prend les représentations apprises dans la première couche comme représentations des mots.

NNLM est un modèle classique de *Word Embedding* qui est à la base de beaucoup de modèles plus avancés comme *CBOW* et *Skip-gram*. Dans la construction de notre modèle, nous l'utilisons comme modèle de référence en terme de performances.

2.7 Doc2Vec

Doc2Vec est un modèle qui est basé sur le *NNLM*, mais dont le but est de produire des représentations vectorielles de phrases et non pas de mots uniquement.

Semblable au modèle *NNLM*, le modèle *Doc2Vec* cherche à prédire la probabilité d'occurrence d'un mot dans un contexte donné. Mais dans le modèle *Doc2Vec*, l'information du contexte que le réseau prend comme entrée contient non seulement les mots du contexte, mais aussi un vecteur représentant la phrase courante. Autrement dit, le modèle *Doc2Vec* prédit la probabilité d'un mot sachant les mots autour, et la phrase ou le paragraphe considéré. Ainsi, lors de l'apprentissage, on apprend des représentations des phrases, et non uniquement des mots.

Nous avons essayé le modèle *Doc2Vec* pour notre projet. Mais, appliqué à notre ensemble de données, on a un f1-score de moins de 0.1. Ce résultat est donc pire qu'une sélection aléatoire. Nous pensons que la raison est la quantité de données insuffisante pour construire nous même un modèle de *Word Embedding* performant, ce qui nous a conduit à l'idée d'utiliser un modèle pré-entraîné.

2.8 EMBEDDING CONTEXTUALISÉ

Les limitations rencontrées dans les approches citées dessus nous ont poussé à envisager des *Contextualized Word Embeddings*, c'est à dire à des représentations vectorielles qui soient sensibles au contexte. Par exemple, considérons les deux phrases : *I want to book a table in the center city* et *I am reading a book*. Le mot *book* dans les deux phrases a un sens complètement différent. Néanmoins, dans un modèle comme *NNLM* ou *Word2Vec*, les deux mots auront la même représentation. Mais pour les modèles contextualisés, cette représentation est différente selon le contexte.

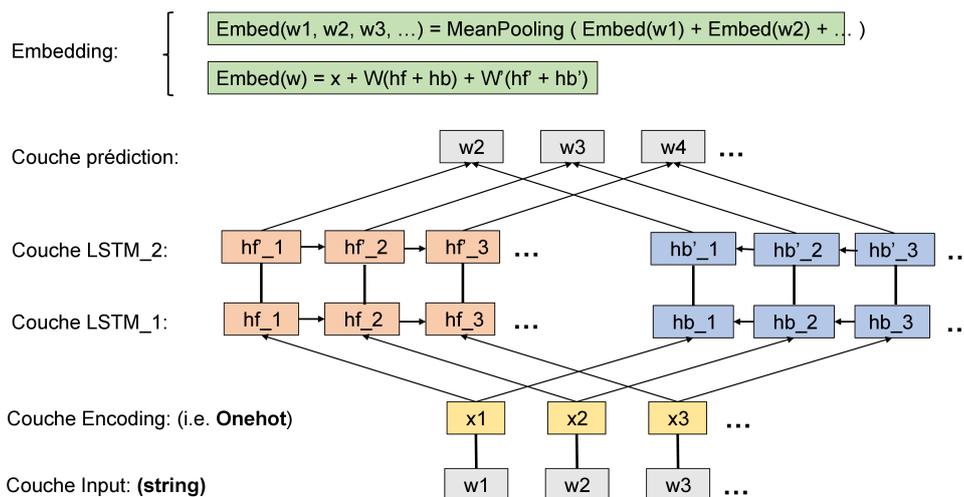
2.8.1 • UNIVERSAL SENTENCE ENCODER

Universal Sentence Encoder [11] est un modèle de *Word Embedding* contextualisé basé sur la structure *Transformer* [15]. En une phrase, le *Transformer* prend compte la représentation d'autres mots dans la phrase pour implémenter un mécanisme "*Attention*", qui permet de stocker l'information du contexte dans la représentation du mot.

2.8.2 • ELMo

ELMo est un réseau bi-directionnel profond qui produit une représentation vectorielle d'un mot qui respecte à la fois les caractéristiques sémantiques et syntaxiques du mot, et le contexte où il figure.

FIGURE 2 – Fonctionnement d'ELMo avec deux couches bi-LSTM



Le modèle prend en entrée une phrase. Ensuite, pour chaque mot, un premier *embedding* est calculé, par Word2Vec ou Onehot encoding par exemple. Dans le cas du modèle que nous utilisons, ce premier *embedding* est calculé en partant d'un Onehot encoding au niveau des caractères, puis en appliquant plusieurs filtres convolutionnels de différentes tailles. Pour une taille de filtre donnée, un vecteur est calculé en appliquant un *Max-Pooling*. Ensuite ces vecteurs sont concaténés pour former la représentation du mot.

Ensuite, on utilise un nombre L de couches LSTM[14] bi-directionnelles sur la séquence des vecteurs pour prédire le mot prochain. On a donc L couches LSTM qui traitent la séquence dans son ordre naturel, et L couches LSTM qui la traitent dans le sens inverse. On combine ensuite les représentations faites par chacune de ses couches pour obtenir la représentation finale du mot. Les poids de cette combinaison peuvent être appris en fonction de la tâche où on utilise le modèle ensuite. Dans notre cas d'utilisation, on prend une moyenne de ces représentations.

Nous utilisons dans notre partie pratique un modèle *ELMo* pré-entraîné, disponible sur **Tensorflow Hub**[13], qui utilise 2 couches LSTM.

3

MODÈLES CONSTRUITS

Code source : https://github.com/riadeb/Topic_Clustering

3.1 DATASET

Nous travaillons, pour les deux modèles qui suivent, sur le dataset Taskmaster-1 [16]. Ce dataset contient 13.215 conversations entre un humain et un agent dans 6 domaines différents : Commander une pizza, prendre un rendez-vous pour réparer un voiture, commander un taxi, acheter des tickets de cinéma, acheter du café et réserver une table dans un restaurant. Chaque conversation est annotée avec son sujet, et quelques phrases de chaque conversation sont annotées selon l'intention.

FIGURE 3 – Extrait d'une conversation dans Taskmaster-1

```
[{'conversation_id': 'dlg-00055f4e-4a46-48bf-8d99-4e477663eb23',
  'instruction_id': 'restaurant-table-2',
  'utterances': [{'index': 0,
    'speaker': 'USER',
    'text': "Hi, I'm looking to book a table for Korean fod."},
  {'index': 1,
    'speaker': 'ASSISTANT',
    'text': 'Ok, what area are you thinking about?'},
  {'index': 2,
    'speaker': 'USER',
    'text': 'Somewhere in Southern NYC, maybe the East Village?',
    'segments': [{'start_index': 13,
      'end_index': 49,
      'text': 'Southern NYC, maybe the East Village',
      'annotations': [{'name': 'restaurant_reservation.location.restaurant.accept'}]},
    {'start_index': 13,
      'end_index': 25,
      'text': 'Southern NYC',
      'annotations': [{'name': 'restaurant_reservation.location.restaurant.accept'}]}]}
```

Nous utilisons plus spécifiquement les 7708 conversations dans le fichier *self-dialogs.json*. Celles ci ont en moyenne 22 répliques par conversation, avec un minimum de 15 et un maximum de 30.

3.2 DÉTECTION DE SUJET

3.2.1 • MODÈLE

Introduction La première étape du projet consiste à détecter le sujet d'une conversation à partir de la première phrase. Dans le cas où une phrase n'introduit aucun nouveau sujet, le but est de détecter cela aussi et ainsi de pouvoir distinguer entre les phrases qui introduisent un nouveau sujet (et donc de détecter son changement) et les phrases qui n'en introduisent pas.

Un exemple de ceci est la conversation suivante, où l'utilisateur aborde un premier sujet, *movie*, puis change pour un nouveau sujet, *uber*, avec une phrase au milieu qui n'annonce pas de nouveau sujet.

User : <i>I want to see a movie</i>	"movie"
Bot : <i>what movie ?</i>	"non – opening"
User : <i>I don't know, what's available ?</i>	"non – opening"
User : <i>I need a ride home</i>	"uber" → Changement

Algorithme L'entrée de l'algorithme est une chaîne de caractères qui représente une réplique de l'utilisateur. Cette chaîne est d'abord transformée en vecteur en utilisant un modèle de *Word Embedding* pré-entraîné puis passe par une seule couche qui transforme la représentation vectorielle de l'entrée en un vecteur à 7 dimensions, chacun d'elle correspond à une catégorie parmi les 7 catégories : '*auto*', '*coffee*', '*movie*', '*pizza*', '*restaurant*', '*uber*', '*non-opening*'.

Représentations vectorielle des phrases - *Word Embedding* Pour notre modèle final, nous utilisons comme *Embedding* ELMo [12]. Pour justifier ce choix, nous comparons ses performances avec 3 autres modèles d' *Embedding* NNLM-en-dim128 (Neural-Net Language Models) [9], Gnews-swivel-20dim [10] et Universal Sentence Encoder (dim512) [11].

3.2.2 • EXPÉRIENCE

Données Nous constituons un jeu de données spécifique à ce modèle à partir de *TaskMaster 1*, en ajoutant la première phrase de chaque conversation à la catégorie de la conversation (*auto*, *coffee*...) et en prenant aussi une phrase aléatoire de chaque conversation, à l'exception de la première, que nous ajoutons à la catégorie *non-opening*. Ensuite nous divisons les données en trois parties, une partie d'entraînement (72%), une partie de validation (18%) pour trouver les meilleurs hyperparamètres, et une partie de test (10%) pour évaluer la performance du modèle final.

Hyper-paramètres à régler En plus du modèle d'*Embedding* utilisé, plusieurs hyperparamètres sont à déterminer :

TABLE 3 – Répartition des données entre les 7 catégories

Catégorie	auto	coffee	movie	pizza	restaurant	uber	non-opening	total
Nombre d'entrées	1161	1376	1305	1400	1300	1098	1400	9040

- La fonction perte (*categorical crossentropy*, *cosine similarity*, *binary crossentropy*, *kullback leibler divergence*, *hinge*)

— L'optimiseur (*adam*, *sgd*)

— Le nombre d'étape d'entraînement (*epoch*).

Pour évaluer et comparer les différentes configurations, nous utilisons le f1-score (sur l'ensemble de validation).

$$F_1 = \left(\frac{2}{\text{recall}^{-1} + \text{precision}^{-1}} \right) = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

Avec $\text{precision} = \frac{TP}{TP+FP}$, $\text{recall} = \frac{TP}{TP+FN}$
TP : True Positives, *TN* : True Negatives, *FP* : False Positives, *FN* : False Negatives

Dans notre cas, vu que les données sont équilibrées entre les différentes catégories, nous utilisons le score F1 *macro*, qui consiste à calculer le score F1 relatif à chaque classe puis faire une moyenne de ces scores.

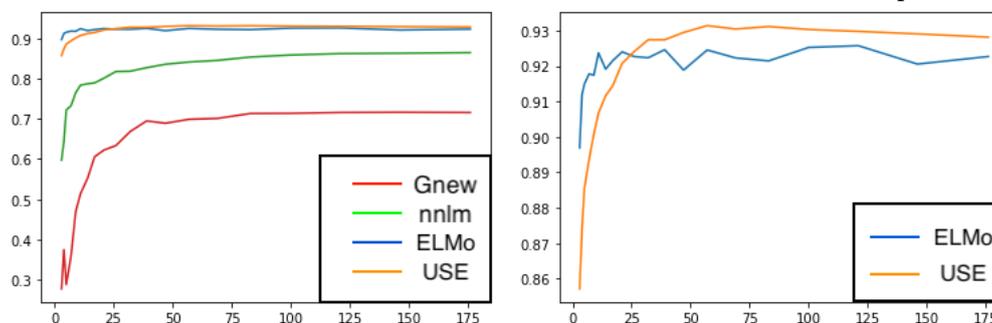
Pour trouver les bons hyperparamètres, nous comparons d'abord toutes les configurations possibles d'hyperparamètres en fixant le nombre d'*époques* à 3, puis, une fois la fonction perte et l'optimiseur trouvé, nous cherchons le meilleur résultat obtenu avec un nombre d'*époques* inférieur ou égal à 200.

3.2.3 • RÉSULTATS

Les résultats de la recherche des meilleurs hyperparamètres sont représentés sur le Tableau 5. Nous pouvons constater qu'ELMo réalise le meilleur score F1 sur l'ensemble de test.

TABLE 4 – Hyper-paramètres optimaux

	Fonction perte	Optimiseur	Nombre d'epoch
Universal Sentence Encoder	<i>cosine similarity</i>	adam	57
NNLM	<i>kullback leibler divergence</i>	adam	176
Gnew	<i>kullback leibler divergence</i>	adam	146
ELMo	<i>cosine similarity</i>	adam	121

FIGURE 4 – Évolution du f1-score en fonction du nombre d'*epoch*

Après avoir trouvé les hyperparamètres optimaux, nous calculons le f1-score sur l'ensemble de test, afin de comparer les modèles. Nous avons donc de très bons résultats pour les 4 mo-

TABLE 5 – f1-score sur l'ensemble de test

	USE	NNLM	Gnew	ELMo
f1-score	0.9356876075763586	0.8641705193028264	0.7197255038699577	0.9353244284408258

dèles, avec des résultats très proches pour *ELMo* et *Universal Sentence Encoding*. Les modèles convergent assez rapidement et l'entraînement des modèles se fait rapidement (moins d'une seconde par *epoch* avec un *batch size* de 64 sur une machine de l'École). Ceci est, bien sur, dû à la simplicité du modèle à entraîner, puisque que la partie *Embedding* du modèle est pré-entraîné.

3.3 DÉTECTION D'INTENTION

3.3.1 • MODÈLE

Introduction Après avoir classifié les phrases dans plusieurs contextes, nous cherchons à détecter les intentions des utilisateurs dans chaque contexte. Dans le contexte *restaurant*, la phrase *Somewhere in Southern NYC, maybe the East Village?* exprime l'intention de préciser l'endroit. La détection d'intention est cruciale pour comprendre plus précisément le sujet abordé et est une étape nécessaire pour construire un chat-bot.

Algorithme En premier lieu nous transformons notre entrée, une phrase, en un vecteur avec *ELMo*, présenté dans la section précédente. Ensuite nous construisons plusieurs classificateurs binaires pour chaque sujet. Chaque classificateur nous permet de détecter une intention dans le sujet. Un classificateur prédit la probabilité qu'une phrase donnée, dans un contexte donné, exprime une intention donnée. Par exemple, pour le sujet *restaurant*, nous avons 5 intentions :

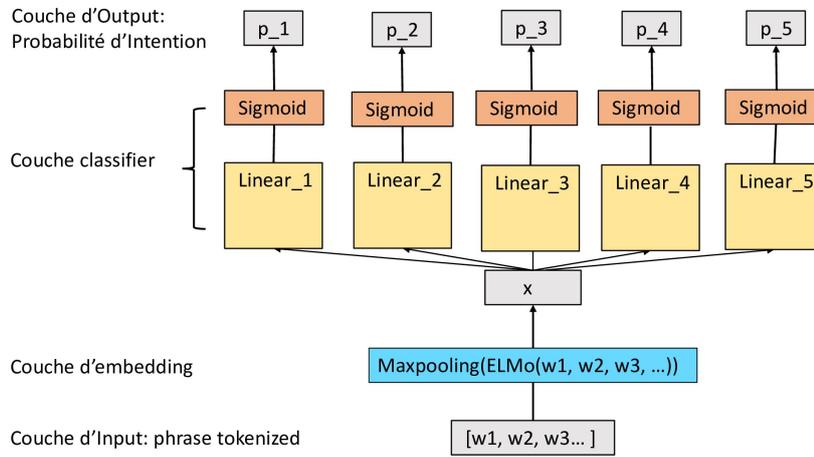


FIGURE 5 – Système de Détection d'Intention

time of service, location of the service, number of users, type of service, name of restaurant, donc cinq classificateurs liés au sujet *restaurant*.

La structure du système est illustrée dans la FIGURE 5

3.3.2 • EXPÉRIENCES

Données Pour cette tâche, nous fournissons à notre modèle un ensemble de données de 34349 phrases avec intentions et 7475 phrases sans intentions.

Notons que nous avons 6 sujets différents. Chaque sujet inclut entre 4 et 6 intentions. Au total, nous avons 31 classificateurs à entraîner.

Pour chaque classificateur d'intention, nous fournissons 50% d'exemples positifs (i.e. avec la bonne intention), 25% d'exemples avec la mauvaise intention et 25% exemples sans intention comme ensemble de données. Nous nous assurons que chaque ensemble de données est équilibré de la sorte et qu'il contient au moins 1000 phrases. Puis nous divisons aléatoirement cet ensemble en un ensemble d'entraînement de 80% et un ensemble de test de 20%.

Classificateur Pour le classificateur, nous choisissons la *régression logistique* avec *penalty = "l2"* et *optimizer = "lbfgs"* comme modèle de base (*baseline*). Nous entraînons ensuite un réseau de neurones linéaire à 3 couches, qui vient après un réseau ELMo, en cherchant le nombre

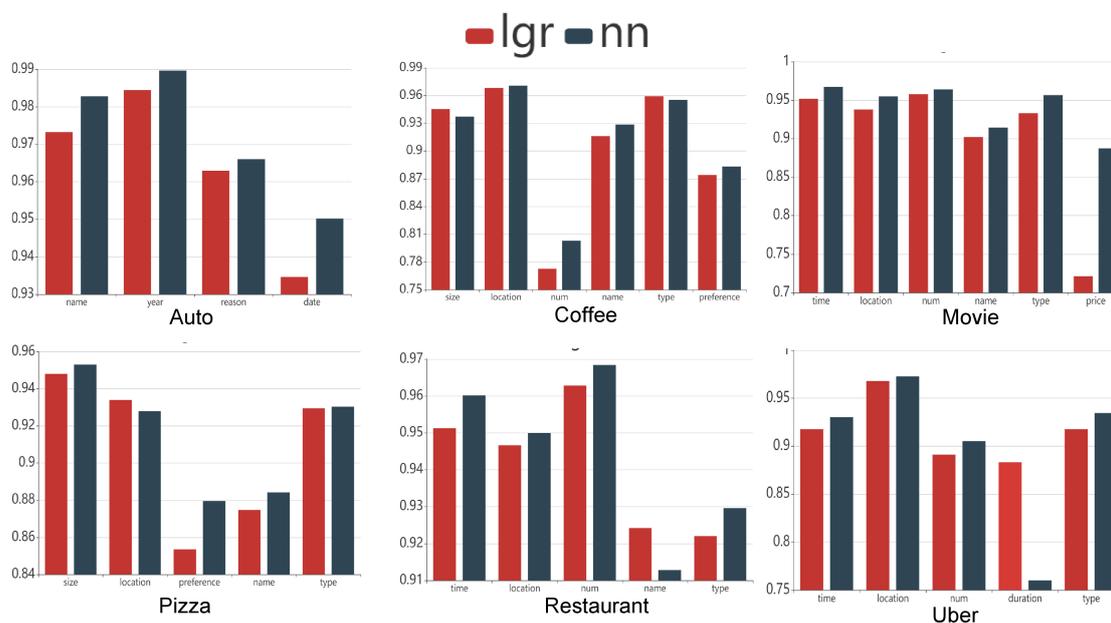


FIGURE 6 – f1-score de la régression logistique et du réseau de neurones

optimal d'*epoch* grâce à une technique d'*early stopping*, pour éviter le surapprentissage. Les f1-scores de la *régression logistique* et des réseaux de neurones sur l'ensemble de test sont présentés dans la FIGURE 6.

3.3.3 • ANALYSE DE RÉSULTATS

Comme donné dans la FIGURE 6, nous avons des résultats assez satisfaisants sur l'ensemble de test. Nous pouvons faire les remarques suivantes :

- Les macro f1-score sont tous plus grands que 0.75 pour le modèle de base et le modèle avec réseau de neurones. Plus précisément, dans 90% des situations, nous avons un macro f1-score supérieur à 0.9. Cela implique qu'un classificateur assez simple comme une *régression logistique* peut avoir une bonne performance dans notre système.
- Dans 83% des situations (26 parmi 31), le modèle avec réseau de neurones acquiert un meilleur macro f1-score que le *régression logistique*. Dans quelques situations la différence entre ces deux classificateurs est remarquable. Par exemple, *Topic Movie* avec *Intention Price*. Mais, mis à part quelques exceptions, les deux classificateurs ont des performances assez proches. Cela vient en partie du fait que les sous-tâches de chaque classificateur ne soient pas très compliquées, une fois que la transformation des mots en vecteurs est fait par l'embedding.

Pour notre système, nous dégagons plusieurs avantages et défauts en expérimentant et en observant les résultats obtenus.

Avantages

- Nous avons d'excellentes performances sur l'ensemble de test ;
- Pour chaque classificateur, le modèle n'est pas très compliqué. En effet, nous observons pendant les expériences que nos modèles convergent assez vite. Notre modèle a ainsi une faible complexité (computationnelle).

Défauts

- L'intention détectée de l'utilisateur dépend d'abord du sujet initial. Mais, pour des conversations plus compliquées, appliquer le même modèles nécessiterait de subdiviser encore certains sujets à des sous-sujets afin de bien comprendre à quoi se rattache l'intention détectée.

Par exemple, quand l'utilisateur dit *This evening at 8 O'clock.* sous le sujet *movie*. On peut détecter l'intention *time*. Mais nous ne pouvons pas savoir s'il s'agit de l'heure de départ du film, l'heure de la fin du film ou autre. Savoir l'intention correcte dépend donc d'une subdivision suffisante des sujets afin de lever toute ambiguïté sur à quoi se rattache l'intention.

Conclusion Nous prenons le système avec le réseaux de neurones comme modèle final. Nous essayons ensuite de l'intégrer avec le modèle construit dans la section précédente.

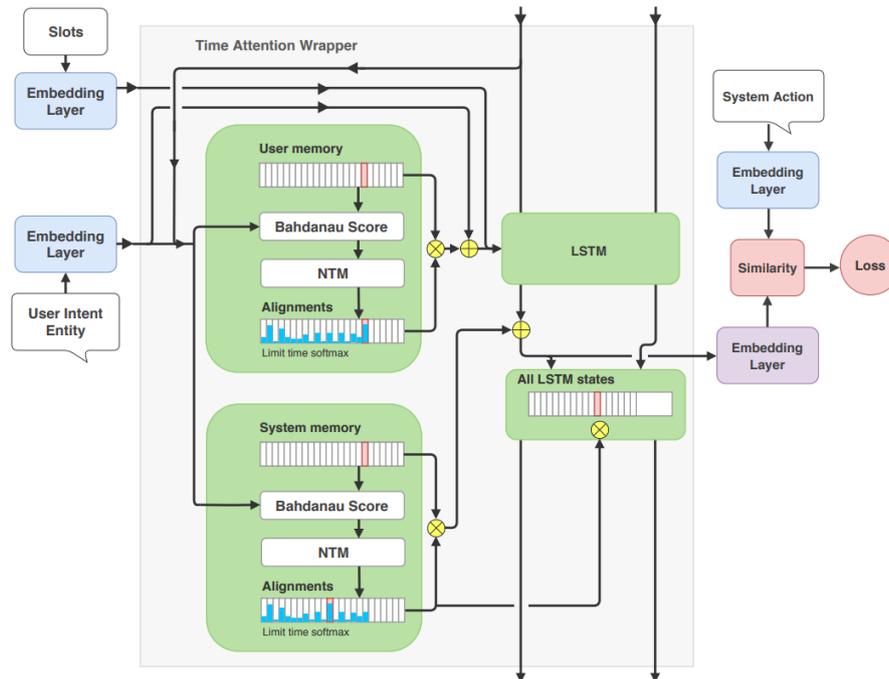


FIGURE 7 – Structure du REDP [17]

3.4 GÉNÉRATION DES RÉPONSES

Afin de compléter les deux modèles construits, nous nous sommes ensuite intéressés à la façon de générer des réponses de manière humaine, de telle façon que la différence entre la réponse d’un humain et d’un bot soit faible. Il est intéressant de s’attarder sur ce qui fait la différence entre un humain et un algorithme. En effet, si l’utilisateur choisit de bavarder, un humain tiendrait une conversation, un bot quand à lui aurait tendance à renvoyer une phrase exprimant son incompréhension. C’est donc dans ces cas limites que va se jouer la différence (si la compréhension est assez bonne).

3.4.1 • MÉCANISME D’ATTENTION : RECURRENT EMBEDDING DIALOGUE POLICY

Le principe d’un Recurrent Embedding Dialogue Policy (REDP) est de transformer des intentions (idées et informations principales de la phrase, décodées d’un dialogue) en actions (réponse et actions sur le sujet, comme par exemple le remplissage d’une procédure de réservation de restaurant) [17]. Les entrées sont donc de la forme “request_restaurant”, “inform{“location” : “Paris”}” (issues par exemple d’un algorithme de NLU) et les sorties sont des actions issues d’un ensemble défini par l’administrateur du chatbot. Contrairement à un LSTM, un REDP parvient à apprendre des dialogues non coopératifs afin de ne pas polluer les

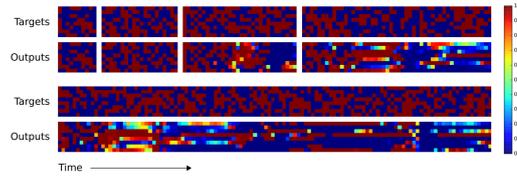


FIGURE 8 – Généralisation d’un LSTM pour un algorithme de copie [18]

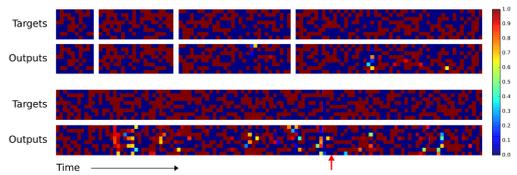


FIGURE 9 – Généralisation d’une NTM pour un algorithme de copie [18]

actions en cours par du bavardage (“chitchat”), problème majeur des chatbots d’aujourd’hui (Il suffit de voir ce qu’est devenu le chatbot de Twitter qui a appris malgré lui des phrases à connotation nazies). De plus, grâce à un mécanisme d’attention, le réseau est conscient du contexte de la conversation donc une question faisant référence implicitement au contexte de la conversation peut avoir une réponse adéquate. Dans le REDP, le mécanisme d’attention est réalisé par un score de Bahanau couplé avec une “Neural Turing Machine” (NTM).

3.4.2 • NEURAL TURING MACHINE

Les Neural Turing Machines (ou NTM) sont utilisées pour résoudre des problèmes nécessitant une mémoire et surtout un besoin de généralisation [18]. En effet, une NTM va pouvoir généraliser un problème bien plus facilement qu’un LSTM (FIGURE 8 et 9, plus il y a des couleurs différentes de bleu et rouge sombre plus il y a des erreurs dans l’algorithme de copie).

Une NTM est ainsi une machine de Turing différentiable (l’apprentissage nécessite en effet la différentiabilité). Elle est constituée d’un contrôleur, de têtes d’écriture et de lecture, et d’une mémoire. On retrouve les éléments d’une machine de Turing classique avec pour différence que la lecture et l’écriture se fait par des vecteurs et des probabilités. Le contrôleur est ainsi un réseau de neurones récurrent comme un LSTM (FIGURE 10).

La mémoire (une matrice de taille $N \times M$ avec N le nombre de cases mémoire et M la taille des vecteurs sauvegardés) peut être lue et écrite. A partir du contrôleur et de l’état précédent, une fonction de probabilité est produite selon le schéma (FIGURE 11).

Cela donne les fonctions suivantes qui diffèrent si on souhaite faire de l’adressage selon le contenu de la mémoire :

$$w_i^c(i) \leftarrow \frac{\exp(\beta_t K[\mathbf{k}_t \mathbf{M}_t(i)])}{\sum_j \exp(\beta_t K[\mathbf{k}_t \mathbf{M}_t(j)])} \quad (1)$$

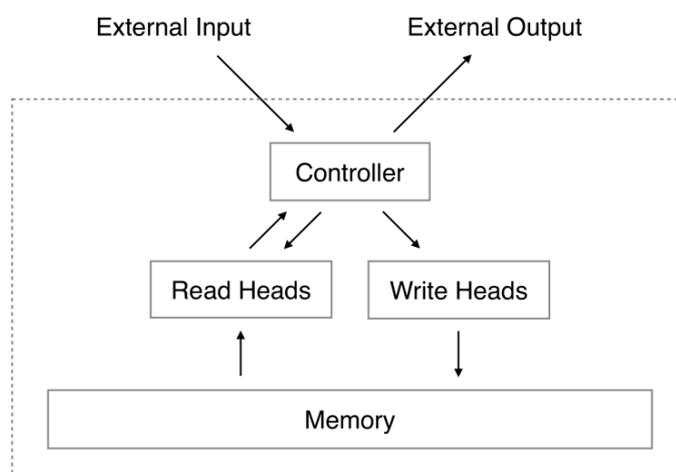


FIGURE 10 – Structure d’une machine de Turing neurale [18]

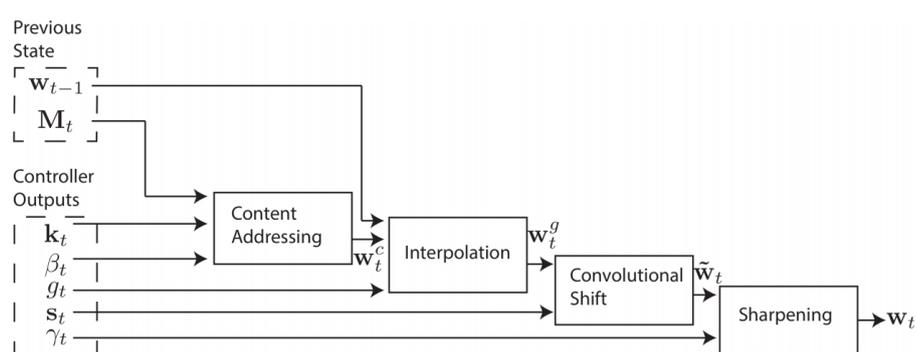


FIGURE 11 – Adressage d’une NTM [18]

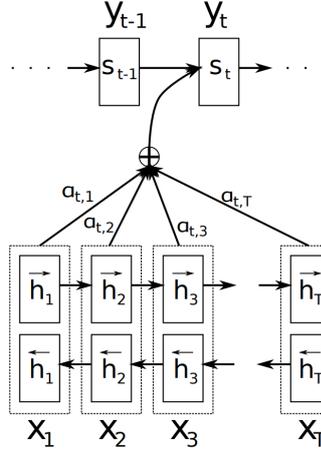


FIGURE 12 – Score de Bahdanau [19]

avec K étant le produit scalaire normalisé :

$$K[\mathbf{u}, \mathbf{v}] = \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\| \cdot \|\mathbf{v}\|}$$

ou selon un adressage relatif :

$$\mathbf{w}_t^g \leftarrow g_t \mathbf{w}_t^c + (1 - g_t) \mathbf{w}_{t-1}$$

$$\tilde{w}_t(i) \leftarrow \sum_{j=0}^{N-1} w_t^g(j) s_t(i - j)$$

$$w_t(i) \leftarrow \frac{\tilde{w}_t(i)^{\gamma_t}}{\sum_j \tilde{w}_t(j)^{\gamma_t}} \quad (2)$$

A partir ce vecteur de poids, il est possible de lire (équation 3) ou d'écrire (équation 4) dans la mémoire de la NTM selon les équations suivantes :

$$\mathbf{r}_t = \mathbf{M}_t^\top \mathbf{w}_t = \sum_i w_t(i) \mathbf{M}_t(i) \quad (3)$$

$$\mathbf{M}_t = \mathbf{M}_{t-1} \circ (\mathbf{E} - \mathbf{w}_t \mathbf{e}_t^\top) + \mathbf{w}_t \mathbf{a}_t^\top \quad (4)$$

avec \circ la multiplication terme à terme \mathbf{E} une matrice de même taille que \mathbf{M}_t mais composée uniquement de 1. Les vecteurs \mathbf{a} et \mathbf{e} sont, respectivement, les vecteurs d'ajout et de suppression.

3.4.3 • SCORE DE BAHDANAU

Le score de Bahdanau est un mécanisme d'attention qui consiste à calculer un poids pour chaque état caché par rapport à l'importance de celui-ci dans le contexte actuel (FIGURE 12)

[19]. On calcule l'importance relative par rapport à l'état précédent (selon une des trois fonctions de l'équation 5) qu'on normalise (équation 6) :

$$\text{score}(\mathbf{h}_{t-1}, \bar{\mathbf{h}}_i) = \begin{cases} \mathbf{h}_t^\top \bar{\mathbf{h}}_i \\ \mathbf{h}_t^\top \mathbf{W}_a \bar{\mathbf{h}}_i \\ \mathbf{v}_a^\top \tanh(\mathbf{W}_a [\mathbf{h}_t; \bar{\mathbf{h}}_i]) \end{cases} \quad (5)$$

$$\mathbf{a}_t(i) = \frac{\exp(\text{score}(i))}{\sum_j \exp(\text{score}(j))} \quad (6)$$

On peut alors en déduire un vecteur de contexte qui exprime l'importance de certains états cachés du réseau sur d'autres (par exemple dans le cas d'un dialogue, les messages relatifs au sujet auront plus d'importance).

$$\mathbf{c}_t = \sum_i \mathbf{a}_t(i) \bar{\mathbf{h}}_i \quad (7)$$

3.4.4 • REDP

Le Recurrent Embedding Dialogue Policy [17] est constitué de plusieurs blocs : deux NTM modifiées et un LSTM. Tout d'abord, le LSTM est un LSTM chrono-initialisé. La chrono initialisation consiste à considérer une certaine invariance des données par rapport au temps. Il est donc possible d'initialiser les biais à des valeurs qui vont améliorer la mémoire à long terme sans impacter la mémoire à cours terme :

$$b_i \sim -\log(\mathcal{U}([1, T_{max} - 1])) \quad (8)$$

C'est à partir des états cachés de ce LSTM et des entrées de l'utilisateur que le score de Bahdanau va pouvoir être calculé au sein des NTM modifiées.

Ces NTM modifiées sont donc des NTM avec quelques variations. En effet, par rapport aux NTM classiques, celles-ci ont une mémoire de taille variable : comme il n'est pas possible de savoir ce que va dire l'utilisateur au temps $t+1$ (en particulier s'il n'est pas coopératif), l'espace mémoire qui est situé au dessus du temps t n'est donc pas utilisé. Cela pose des problèmes au moment de l'adressage de la NTM. Pour résoudre cela, il est possible de modifier l'étape de décalage convolutionnel pour accommoder cette situation. De plus, lors de l'interpolation avec les étapes précédentes, comme la mémoire est tronquée, il n'est pas possible d'interpoler l'état précédent avec les \mathbf{w}_t^c . Il est possible d'utiliser les scores de Bahdanau.

Une des deux NTM modifiées (MNTM) envoie sa sortie à l'entrée du LSTM. L'autre l'ajoute à la sortie du LSTM. Cette somme est ensuite conservée pour le pas de temps $t+1$ et est aussi transformée en une action a qui est injectée dans la fonction objectif.

$$L_t = \max(\mu_+ - \text{sim}(a, b_+), 0) + \max\left(\mu_- + \max_{b_-}(\text{sim}(a, b_+)), 0\right) \quad (9)$$

L'action a déterminée précédemment est comparée aux actions disponibles. La fonction de objectif compare dans un premier terme a à l'action voulue b_+ (on cherche alors à maximiser $\text{sim}(a, b_+)$ pour atteindre μ_+), et dans le deuxième terme avec toutes les autres actions possibles mais non souhaitées (on cherche à minimiser la similitude entre a et tous les b_- de façon à ce que cette similitude prenne pour valeur μ_-).

3.4.5 • COMPARAISON AVEC UN LSTM

On observe qu'un REDP obtient de bien meilleurs résultats lorsque l'utilisateur n'est pas coopératif avec le chatbot (FIGURE 13). En effet, même avec peu de dialogues d'entraînement dans lesquels l'utilisateur ne serait pas coopératif, le REDP obtient de meilleurs résultats qu'un LSTM grâce au mécanisme d'attention et de généralisation que constitue notamment les deux MNTM.

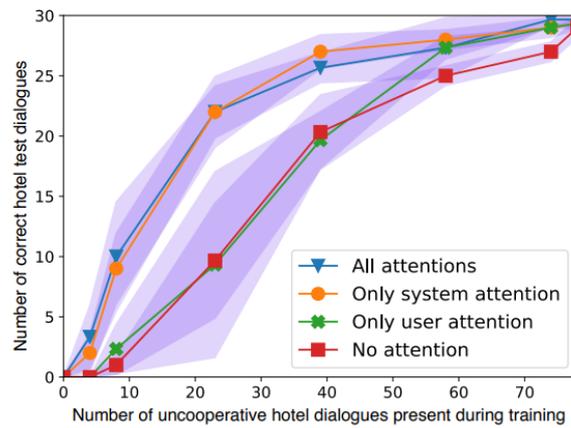


FIGURE 13 – REDP vs LSTM [17]

La FIGURE 14 montre une discussion avec un bot utilisant le REDP.

```
Bot loaded. Type a message and press enter (use '/stop' to exit):
request_hotel
Anything specific?

inform{"startdate": "January 29th 2020"}
When should your hotel stay end?

inform{"enddate": "in a month"}
What price range?

inform{"price": "cheap"}
Where?

explain
To show you relevant results, I'll have to know where you are going
Where?

correct{"startdate": "today"}
Ok, I've changed your check in date to today
Where?

chitchat
Bla bla
Where?

inform{"location": "Palaiseau"}

How many people will you be?

inform{"people": "4"}

Ok, I have all the information I need now.

Here is what I found: Hotel Berlin. Does that work?

affirm

Great!
```

FIGURE 14 – Exemple

3.5 INTÉGRATION DES DIFFÉRENTS MODÈLES

Enfin, nous essayons de construire un système de *NLU* : d'étiquetage du sujet (*Topic Labelling*) en utilisant les deux modèles construits dans les deux sections dessus.

D'abord, nous déterminons le sujet abordé par l'utilisateur avec notre modèle de *Topic Clustering*. Si le sujet prédit n'est pas *non-opening* et diffère du sujet courant, il y a un changement de sujet, et nous dans entrons le scénario d'autre sujet. Ensuite nous envoyons l'entrée et le sujet courant à notre modèle d'*Intent Detection*, ce qui nous donne une liste de probabilités d'intentions dans la phrase. La FIGURE 15 est une illustration de notre système de *NLU*.

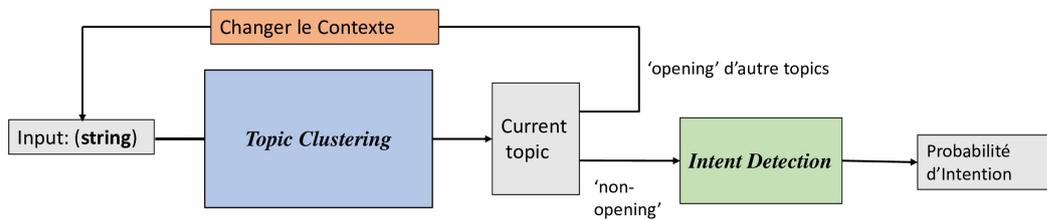


FIGURE 15 – système *NLU* combinant *Topic Clustering* et *Intent Detection*

Nous utilisons enfin le système *REDP* pour générer la réponse en utilisant les intentions possibles.

Le Tableau 1 illustre une situation possible d'utilisation de notre système.

Bot : *Hello, what can I do for you ?*

User : *I want some newest movies, please.*

Bot : *It seems you want to book movie tickets, What time do you prefer ?*
 topic predicted : movie... entering the scenario : movie
 Detected intents : name type

User : *This evening at 8'o clock*

Bot : Detected intents : time
What type do you prefer

User : *Wonder woman II is perfect*

Bot : Detected intents : name
It's noted. User : That's it! And could you book a table around the cinema ?

Bot : *It seems you want to reserve a table*
 topic predicted : restaurant... entering the scenario : restaurant
 Detected intents : location
It's noted. How many people do you have ?

User : *We have 4 totally.*

Bot : Detected intents : number
It's noted. What about : XXX

User : *Ok thank you.*

Bot : *Thank you for you attention*
 conversation end.

TABLE 6 – Implémentation du système

4

CONCLUSION

Ainsi, durant ce PSC, nous avons construit un système de *NLP* qui combine plusieurs fonctionnalités fondamentales des chatbots : la détection de sujet, la détection d'intention et la génération de réponse.

Nous avons étudié et testé différents algorithmes, ce qui nous a permis finalement d'en utiliser certains pour construire notre propre système qui, après les tests, a de bonnes performances.

La partie compréhension (NLU) de notre système répond à notre problématique de Topic Clustering. Nous avons des résultats satisfaisants sur l'ensemble de test, mais nous savons que pour détecter plus d'informations nous aurions besoin de travailler avec un ensemble de données bien plus fourni.

Enfin pour aller plus loin, nous avons étudié comment construire un chatbot complet avec la génération de réponse. Cela nous a ainsi aidé à mieux comprendre le fonctionnement des chatbots.

RÉFÉRENCES

- [1] Avinash Navlani. *Latent Semantic Analysis using Python*
[https://www.datacamp.com/community/tutorials/
discovering-hidden-topics-python](https://www.datacamp.com/community/tutorials/discovering-hidden-topics-python)
- [2] Pema Gurungland, Rupali Wagh. *A study on Topic Identification using K means clustering algorithm : Big vs. Small Documents*
https://www.ripublication.com/acst17/acstv10n2_07.pdf
- [3] Topic Modelling with LSA and LDA.
[https://forestforthetree.com/statistics/2018/01/28/
topic-modelling-with-lsa-and-lda.html](https://forestforthetree.com/statistics/2018/01/28/topic-modelling-with-lsa-and-lda.html)
- [4] Joyce Xu. *Topic Modeling with LSA, PLSA, LDA & lda2Vec*
[https://medium.com/nanonets/
topic-modeling-with-lsa-psla-lda-and-lda2vec-555ff65b0b05](https://medium.com/nanonets/topic-modeling-with-lsa-psla-lda-and-lda2vec-555ff65b0b05)
- [5] Language Models are Unsupervised Multitask Learners. Radford, Alec and Wu, Jeff and Child, Rewon and Luan, David and Amodei, Dario and Sutskever, Ilya. 2019
- [6] Adam King, build of gpt2 <https://talktotransformer.com/>
- [7] Coached Conversational Preference Elicitation [https://ai.google/tools/datasets/
coached-conversational-preference-elicitation](https://ai.google/tools/datasets/coached-conversational-preference-elicitation)
- [8] Ensemble de dialogue de films.
[http://www.cs.cornell.edu/~cristian/
Cornell_Movie-Dialogs_Corpus.html](http://www.cs.cornell.edu/~cristian/Cornell_Movie-Dialogs_Corpus.html)
- [9] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, Christian Jauvin. *A Neural Probabilistic Language Model*. Journal of Machine Learning Research, 3 :1137-1155, 2003.
- [10] Noam Shazeer, Ryan Doherty, Colin Evans, Chris Waterson. *Swivel : Improving Embeddings by Noticing What's Missing*. <https://arxiv.org/abs/1602.02215>.
- [11] Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St. John, Noah Constant, Mario Guajardo-Céspedes, Steve Yuan, Chris Tar, Yun-Hsuan Sung, Brian Strope, Ray Kurzweil. *Universal Sentence Encoder*. <https://arxiv.org/abs/1803.11175>.
- [12] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, Luke Zettlemoyer *Deep contextualized word representations*. <https://arxiv.org/abs/1802.05365>.
- [13] <https://tfhub.dev/google/elmo/3>
- [14] Francois Chollet *Deep Learning with Python*. p 196-206
- [15] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser *Attention Is All You Need*. <https://arxiv.org/abs/1706.03762>

- [16] Bill Byrne and Karthik Krishnamoorthi and Chinnadhurai Sankar and Arvind Neelakantan and Daniel Duckworth and Semih Yavuz and Ben Goodrich and Amit Dubey and Kyu-Young Kim and Andy Cedilnik *Taskmaster-1 : Toward a Realistic and Diverse Dialog Dataset*.
- [17] Vladimir Vlasov, Akela Drissner-Schmid, Alan Nichol *Few-Shot Generalization Across Dialogue Tasks*. <https://arxiv.org/abs/1811.11707>
- [18] Alex Graves, Greg Wayne, Ivo Danihelka *Neural Turing Machine*. <https://arxiv.org/abs/1410.5401>
- [19] Dzmitry Bahdanau, KyungHyun Cho, Yoshua Bengio *Neural Machine Translation by Jointly Learning to Align and Translate*. <https://arxiv.org/abs/1409.0473>