École Centrale Lyon

Option Informatique
Traitement des langues naturelles
Questions/Réponses

Projet Chatbot

Élèves :
Hamza Ed-Dbiri
Said Khaboud
Hugo Mailfait
Salaheddine Mesdar

Tuteur :
Alexandre Saidi

24 mars 2021

Table des matières

Table des figures

Introduction

Depuis plusieurs années, l'intelligence artificielle a bouleversé plusieurs aspects de nos activités quotidiennes via la conception d'applications et de dispositifs avancés qui peuvent remplir diverses fonctions. Parmi ces bouleversements, on retrouve les chatbots. Un chatbot est un modèle d'interaction homme-machine qui repose sur de nombreuses techniques d'intelligence artificielle. De manière simplifiée, il s'agit d'un programme informatique conçu pour simuler une conversation avec des utilisateurs humains. Il utilise en particulier la science du traitement du langage naturel (NLP) et l'analyse des sentiments pour communiquer en langage humain par texte ou par voie orale.

Les progrès en intelligence artificielle ont permis l'explosion de ces nouveaux outils de conversation homme-machine. Ces outils se retrouvent dans la vie quotidienne depuis que les spécialistes en marketing ont par exemple compris qu'ils pouvaient tirer parti des agents conversationnels pour améliorer l'expérience client. Les chatbots sont alors devenus un outil incontournable du marketing conversationnel, une technique qui permet d'établir une discussion personnalisée en temps réel pour capturer, qualifier et se connecter avec un client.

Dans le cadre du projet d'option informatique en 3ème année à l'École Centrale de Lyon, notre groupe s'est intéressé aux technologies d'intelligence artificielle derrière ces outils conversationnels et à la mise en pratique de certains concepts pour réaliser notre propre chatbot.

1 Contexte, objectifs et livrables

Les modèles de conversation sont au coeur des sujets de recherche de ces récentes années. En particulier, les chatbots aident dans de très nombreuses tâches de services après-vente ou pour des guides d'assistance en ligne. Ces bots reposent souvent sur des modèles *retrieve*, qui renvoient des réponses pré-définies. Cependant, pour certaines activités, de tels modèles sont trop contraignants [1]. Dans ce cas, des modèles *generative* existent. Comme leur nom l'indique, ces modèles sont capables de générer des conversations en fonction des propos en entrée de l'utilisateur. Les technologies au coeur de ces méthodes *generative* sont les réseaux de neurones récurrents, les associations Encoder-Decoder et les modèles Seq2Seq conditionnels.

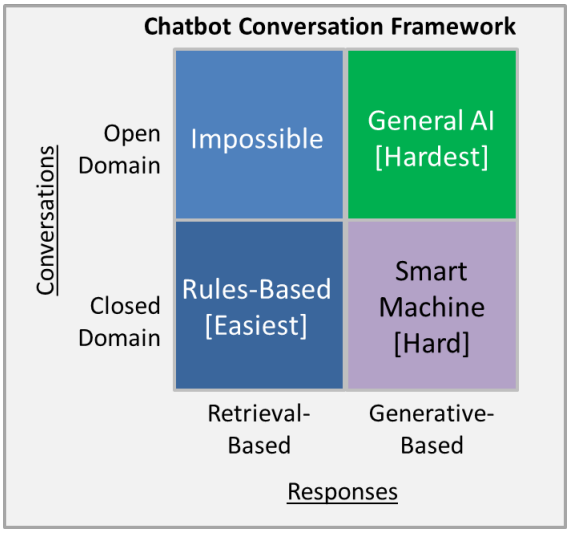


FIGURE 1 – Chatbot RETRIEVE VS GENERATIVE

L'objectif de ce projet est d'étudier la mise en place d'un chatbot à l'aide de différentes techniques d'apprentissage artificiel. Pour cela, une première phase du projet consiste en la réalisation d'un état de l'art sur le sujet des chatbots et des outils technologiques sous-jacents. Ensuite, l'idée est de développer un prototype de chatbot, de le tester sur un corpus et d'en évaluer les résultats. On étudiera plusieurs prototypes pour les mettre en comparaison et présenter les avantages et inconvénients de chacun.

Les livrables du projet sont les différents prototypes de chatbot mis en place, accompagnés de ce rapport qui détaillera les technologies utilisées, la mise en place des solutions et l'évaluation des modèles

2 Présentation générale des travaux réalisés

Pour ce projet, nous utiliserons les deux grandes catégories de modèles différents, à savoir les modèles génératifs et les modèle retrieve. Le détail de leur implémentation et des technologies sous-jacentes sera explicité dans les sections suivantes.

Nos travaux se sont donc concentrés sur plusieurs axes de travail. En premier lieu, une étape de formation aux grands concepts du NLP et aux techniques d'intelligence artificielle était nécessaire. Celle-ci a permis d'appréhender les notions de réseaux de neurones récurrents et leurs spécificités, leur application à la génération de séquences, sujet au coeur des chatbots, ou encore de découvrir les modèles de *word embedding* développés par Google avec Word2Vec [2].

A la suite de cette première phase, une sélection s'est opérée parmi toutes les technologies différentes disponibles. Par rapport aux retours d'expérience des élèves des années précédente sur ce même projet, nous avons jugé pertinent de développer plusieurs prototypes différents, pour ne pas risquer de se retrouver avec un modèle non fonctionnel. Ainsi, nous avons choisi de développer deux modèles retrieve et un modèle génératif.

Le premier modèle retrieve s'attache à utiliser la méthode TF-IDF et exploite des

calculs de similarités entre les phrases. Le second modèle retrieve utilise pour sa part la méthode "bag-of-Words" et un réseau de neurones profond préalablement entraîné pour répondre de manière pertinente à l'utilisateur. Enfin, le modèle génératif se concentre sur la génération conditionnel de séquences en réponse aux questions posées. Elle repose notamment sur les notions de RNN et d'association Encoder-Decoder.

3 Choix et extraction des données

A l'origine de ce projet, l'objectif était la réalisation d'un système questions/réponses que les élèves pourraient employer sur le règlement de la scolarité ou sur une formation de master. Cependant, pour développer un chatbot, l'état de l'art s'accorde à dire qu'un grand nombre de données est nécessaire pour obtenir des résultats convaincants. L'année dernière, le projet d'option sur le même sujet avait tenté l'expérience en utilisant le règlement de la scolarité comme corpus. Pour le modèle génératif, les réponses obtenues par le chatbot n'étaient pas toujours pertinentes.

De fait, l'équipe PROJET a décidé de se consacrer sur une autre thématique, à savoir le cinéma. Diverses bases de données de films et de leurs différentes caractéristiques existent en ligne :

— CORNELL MOVIE-DIALOGUE CORPUS contient par exemple plus de 200 000 extraits de dialogue de fiction [3] ;
— IMDB MOVIES EXTENSIVE DATASET sur Kaggle avec des informations sur plus de 80 000 films et plus de 170 000 acteurs et actrices [4] ;
— etc ...

A l'origine, on souhaitait réaliser un chatbot en français pour faciliter son usage. Pour cela, on a construit une base de données à partir du site AlloCiné. Ce site internet contient un très grand nombre de films différents et leurs caractéristiques.

FIGURE 2 – Extrait de la page web AlloCiné du film *Pulp Fiction*

Pour concaténer l'ensemble des données du site, nous avons réalisé un *parser* à l'aide du framework BeautifulSoup de Python. Ce framework rend possible la fouille du code HTML des pages Web. Ainsi, le programme développé renvoie en sortie un fichier csv avec l'ensemble des informations récupérées sur les différents films.

4 L'étape de preprocessing

Dans les processus de traitement de texte, les phrases/paragraphes/documents ne sont pas directement exploités dans les modèles. En effet, on préfère généralement utiliser des vecteurs. De plus, il est nécessaire de nettoyer le contenu du texte car toutes les informations qu'il contient ne sont pas forcément pertinentes vis-à-vis de notre utilisation. On appelle cette étape le *preprocessing* [5].

D'après les différentes études de l'état de l'art, plusieurs étapes sont indispensables pour réaliser le *preprocessing*. Elles sont illustrées dans le tableau ci-dessous :

I feel so LUCKY to have found this (used) phone.

I. Eliminer la ponctuation	I feel so LUCKY to have found this used phone
II. Convertir en minuscules	i feel so lucky to have found this used phone
III. *Tokenization*	[i, feel, so, lucky, to, have, found, this, used, phone]
IV. Eliminer les *StopWords*	[feel, lucky, have, found, used, phone]
V. *Lemmatization*	[feel, lucky, have, find, use, phone]

Par ailleurs, à la place de la lemmatization, il est possible de réaliser un stemming. Le stemming consiste à réduire un mot dans sa forme “racine”. Par exemple, une fois que l’on applique un stemming sur “Lire” ou “Lirait”, le mot résultant est le même, à savoir "lir". Cela permet également de réduire la taille du vocabulaire. Cependant, cette méthode est plus brute et moins précise que la lemmatization, car elle crée des mots qui pourraient ne pas avoir le même sens dans certains cas.

5 Le modèle Retrieve

5.1 La méthode TF-IDF

Calcul de la fréquence

De manière générale, la méthode TF-IDF permet de déterminer dans quelles proportions certains mots d’un document texte apparaissent par rapport au reste des documents. TF est l’abréviation de l’anglais *term frequency* (fréquence du terme). Il détermine la fréquence relative d’un mot dans un document. Prenons l’exemple du corpus suivant :

Demain, dès l’aube, à l’heure où blanchit la campagne
Je partirai. Vois-tu, je sais que tu m’attends.
J’irai par la forêt, j’irai par la montagne.
Je ne puis demeurer loin de toi plus longtemps.

Dans ce poème de Victor Hugo, le nombre de mots total vaut 31. On remarque par exemple que le mot **irai** est présent 2 fois, **demain** 1 fois et **je** 3 fois. On peut alors calculer la fréquence avec la formule :

$$\frac{\text{Nombre d'apparitions du terme}}{\text{Nombre total de termes dans le document}}$$

Il s’agit ici de la méthode de calcul de fréquence "brute", considérée comme étant la plus simple. De nombreux autres techniques existent [6]. On peut par exemple normaliser logarithmiquement cette fréquence brute pour amortir les écarts.

Inverse Document Frequency

La fréquence inverse de document (*Inverse Document Frequency*) permet de mesurer l’importance de chaque terme dans l’ensemble du corpus. Elle consiste à calculer le logarithme de l’inverse de la proportion de documents du corpus qui contiennent le terme :

$$\mathrm{IDF_i} = \log \frac{D}{\{d_j : t_i \in d_j\}}$$

D correspond au nombre total de documents dans le corpus. Le dénominateur indique le nombre de documents où le terme étudié apparaît. Enfin, le poids final de chaque terme s’obtient en multipliant les deux mesures.

Application au corpus AlloCiné

Pour appliquer cette méthode à notre problématique, on a choisi de créer un ensemble de questions prédéfinies. Ainsi, pour un même sujet (par exemple le réalisateur d'un film), on rédige une dizaine de questions différentes pour interroger cette thématique. Un fichier JSON avec toutes les questions et leur label a donc été créé.

```
{
    "intent": "synopsis",
    "examples": [
        "décris moi le film",
        "decris le film",
        "description",
        "détails du film",
        "details",
        "dis m'en plus sur ce film",
        "description du film",
        "de quoi parle le film ?",
        "quel est le synopsis ?",
        "histoire racontée",
        "de quoi parle le film ?",
        "résumé du film",
        "resume"]
},
{
    "intent": "box_office",
    "examples": [
        "nombre d'entrées",
        "entrées",
        "entrees",
        "box office",
        "revenus du film",
        "combien a rapporté le film ?",
        "combien d'entrées au box-office ?",
        "combien de personnes ont vu ce film au cinéma ?",
        "le film est-il populaire ?"]
},
```

FIGURE 3 – Extrait du fichier JSON pour la méthode TF-IDF

Dans le programme, la première étape consiste à comprendre de quel film l'utilisateur souhaite discuter. Pour cela, on applique la méthode TF-IDF avec comme documents chacun des titres de films sur lesquels on dispose d'informations. On obtient ainsi une matrice TF-IDF, de largeur le vocabulaire de l'ensemble des titres et comme longueur le nombre de films disponibles. Lorsque l'utilisateur rentre le film choisi, on le transforme en un vecteur TF-IDF et on calcule sa similarité avec chacune des lignes de la matrice. La formule de calcul de similarité est donnée ci-dessous :

$$\text{similarity} = \frac{A.B}{||A|| * ||B||} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} * \sqrt{\sum_{i=1}^{n} B_i^2}}$$

On propose alors à l'utilisateur les 3 meilleurs résultats. Il précise alors si son film est parmi les 3 et duquel il s'agit. Cette étape de confirmation est très importante pour éviter de se tromper dans les réponses aux questions à venir.

```
Bonjour, je suis Alex, votre chatbot AlloCine.
Je peux répondre à toutes vos questions sur de nombreux films :
le réalisateur, le genre, la durée, la note des spectateurs, le
synopsis etc ...


Tout d'abord, pourriez-vous rentrer le film sur lequel vous
souhaitez des informations : Le silence des agneaux

Le film choisi est-il l'un des trois ci-dessous ?
['Le Silence des agneaux', 'Le Labyrinthe du silence', 'Les Jeux
des nuages et de la pluie']

Si oui, indiquez son numéro dans la liste, si non écrivez
"erreur" : 1
```

FIGURE 4 – Illustration de la sélection du titre de film

Ensuite, le programme récupère toutes les questions prédéfinies dans le fichier JSON. Pour l'ensemble de ces questions, on procède en premier lieu à l'étape indispensable de *preprocessing*, avec l'usage d'un stemming. Ensuite, on applique à nouveau la méthode TF-IDF à ce corpus où chaque question préalablement *preprocessed* constitue un document. On obtient donc pour chacune des questions un vecteur TF-IDF correspondant.

Lorsque l'utilisateur parle dans le chatbot, sa question est elle aussi *preprocessed* et transformée en un vecteur TF-IDF. On détermine alors de quel document du corpus la question posée par l'utilisateur est la plus proche avec le calcul de similarité. On récupère ensuite le document du corpus le plus proche de la question de l'utilisateur. Avec son label, on connaît maintenant le sujet de la question et on peut alors lui répondre grâce aux données dans le fichier CSV.

```
Quelle est votre question ? : réalisateur
Jonathan Demme

Quelle est votre question ? : qui sont les acteurs principaux
Anthony Hopkins, Jodie Foster, Scott Glenn

Quelle est votre question ? : quel score au box-office ?
3 119 085 entrées

Quelle est votre question ? : QUEL BUDGET pour le film ?
$19.000.000 dlls

Quelle est votre question ? : au revoir
Au revoir et à bientôt j'espère !
```

FIGURE 5 – Chatbot de la méthode TF-IDF

5.2 Utilisation de bag-of-words

Rappel de la théorie

La représentation par sacs de mots (bag-of-words en anglais) est une description de document très utilisée en *text mining*. Elle repose principalement sur l'utilisation d'un

dictionnaire de mots. Pour un document donné, chaque mot se voit affecté son nombre d'apparitions dans le document. Ainsi, un document est représenté par un vecteur de même taille que le dictionnaire, avec le terme i qui indique le nombre d'occurences du i-ème mot du dictionnaire dans le document [7].

Ensuite, il s'agit de normaliser le bag-of-words selon son usage :

— on peut le ramener à une norme unitaire pour simplement traduire la présence ou l'absence d'un mot dans le dictionnaire ou,
— on peut le pondérer selon divers schémas de modèles probabilistes de pertinence (comme dans le cas TF-IDF).

Document	the	cat	sat	in	hat	with
the cat sat	1	1	1	0	0	0
the cat sat in the hat	2	1	1	1	1	0
the cat with the hat	2	1	0	0	1	1

FIGURE 6 – Illustration de la méthode bag-of-words sur 3 documents simples

Par la suite, on utilisera un modèle intelligent car reposant sur un réseau de neurones simple pour attribuer chaque question à une classe et essayer de donner la réponse la plus adéquate.

Nettoyage et préparation des données

Après l'extraction des données des films du site IMDb en un fichier CSV, il convient de commencer par supprimer les colonnes qui contiennent beaucoup de valeurs manquantes et de trier les films par la note moyenne donnée par les spectateurs en prenant en compte le nombre de spectateurs de chaque film. Pour obtenir des temps d'entraînement acceptables, nous avons décidé de travailler sur les 1 000 meilleurs films de cette base de données.

On crée de cette manière un fichier JSON à partir des données IMDb :


```
  {'tag': 'Suvarna Sundari actors',
   'patterns': ['actors of Suvarna Sundari',
    'Who are the principle actors of Suvarna Sundari',
    'Who are the actors of Suvarna Sundari',
    'Who are the main actors of Suvarna Sundari',
    'actors of Suvarna Sundari?'],
   'responses': ['Shamna Kasim, Sakshi Chaudhary, Jaya Prada, Avantika Vandan
apu, Avinash, Indra, Srinivasa Rao Kota, Sai Kumar, Nagineedu, Sathya Prakas
h, Raam'],
   'context': ['']},
  {'tag': 'Suvarna Sundari description',
   'patterns': ['description of Suvarna Sundari',
    'What is the description of Suvarna Sundari',
    'description of Suvarna Sundari?'],
   'responses': ['The movie revolves around an idol, Suvarna Sundari and the
effects of it. The idol, also known as Trinetri, dates back to 15th century.
Whoever possesses the idol, becomes a victim of ...'],
   'context': ['']},
```


FIGURE 7 – Extrait du fichier JSON pour la méthode bag-of-words

Le contenu de ce fichier JSON est un ensemble de messages que l'utilisateur est susceptible de saisir. On les associe également à un groupe de réponses appropriées. La balise de chaque dictionnaire du fichier indique le groupe auquel appartient chaque message. Avec ces données, l'entraînement du réseau de neurones consistera à prendre une phrase de mots et à la classer comme l'une des étiquettes de notre fichier. Ensuite, on prend une réponse parmi ces groupes et on l'affiche à l'utilisateur. Plus le nombre de balises et de réponses fourni au réseau est important, plus il sera performant et complexe.

Chargement des données JSON et pré-traitement

Pour commencer, on récupère du fichier JSON l'ensemble des modèles et de leurs classes ou étiquettes à laquelle ils appartiennent. On boucle ensuite sur chacun des motifs extrait. L'objectif est de transformer la liste de mots en un vecteur utilisable pour notre réseau de neurones. La première étape de la construction de ce bag-of-Words est le *preprocessing*. Pour rappel, ce prétraitement est composé d'une succession d'étapes, parmi lesquelles on retrouve la *tokenization* et la lemmatisation par exemple. Le code responsable de cette étape permet donc de préparer les questiosn pré-définies du fichier JSON et de définir le vocabulaire de notre sujet.

Construction du bag-of-words et alimentation du réseau neuronal

Comme découvert lors de la phase d'état de l'art, les réseaux neuronaux et les algorithmes d'apprentissage automatique nécessitent une entrée numérique. Une liste de chaînes de caractères ne suffit donc pas. Pour cela, nous choisissons de représenter chaque phrase avec un bag-of-words. Pour rappel, la phrase est transformée en une liste de longueur la quantité de mots dans le vocabulaire. Ainsi, chaque position dans la liste représente un mot du vocabulaire. Si la position dans la liste est un 1, cela signifie que le mot existe dans notre phrase, si elle est un 0, alors le mot n'est pas présent. Cela s'appelle un "sac de mots" car l'ordre dans lequel les mots apparaissent dans la phrase est perdu, seule leur présence dans la phrase est évaluée.

En plus de formater les entrées du réseau neuronal, il est impératif de prévoir la sortie de ce réseau. De la même manière qu'un sac de mots, des listes de sortie sont créées, dont la longueur correspond à la quantité d'étiquettes/de balises dans l'ensemble de données. Chaque position dans la liste représentera une étiquette distincte, un 1 dans n'importe laquelle de ces positions indiquera quelle étiquette est représentée.

```
# create our training data
training = []
# create an empty array for our output
output_empty = [0] * len(classes)
# training set, bag of words for each sentence
for doc in documents:
    # initialize our bag of words
    bag = []
    # list of tokenized words for the pattern
    pattern_words = doc[0]
    # lemmatize each word - create base word, in attempt to represent related words
    pattern_words = [lemmatizer.lemmatize(word.lower()) for word in pattern_words]
    # create our bag of words array with 1, if word match found in current pattern
    for w in words:
        bag.append(1) if w in pattern_words else bag.append(0)

    # output is a '0' for each tag and '1' for current tag (for each pattern)
    output_row = list(output_empty)
    output_row[classes.index(doc[1])] = 1

    training.append([bag, output_row])
```

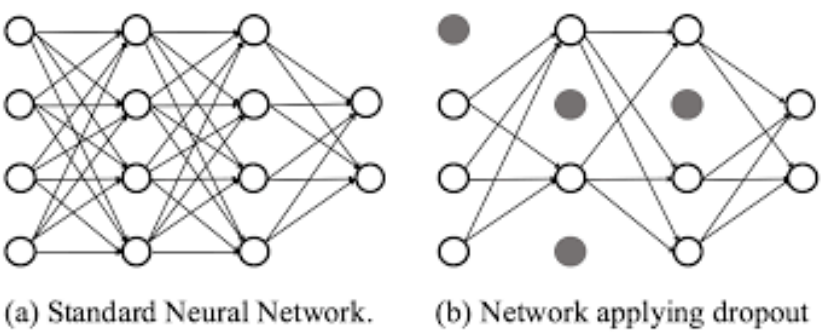
FIGURE 8 – Préparation des sacs de mots et des données pour le réseau neuronal

A noter également que les données d'entraînement et en sortie sont ensuite converties en tableaux numpy.

Développement d'un modèle

Maintenant que les données d'entraînement et de test sont prêtes, il est nécessaire de préparer le modèle d'apprentissage profond. Pour cela, le modèle Sequential de Keras sera utilisé, permettant la construction d'un réseau de neurones simple, à savoir un perceptron multicouche.

En particulier, ce réseau possède 3 couches, avec la première ayant 256 neurones, la deuxième ayant 128 neurones, et la troisième ayant le nombre d'*intents* comme nombre de neurones. Le modèle sera entraîné avec la méthode de descente du gradient stochastique qui est plus efficace que la descente de gradient normale. Un *dropout* est également implanté pour les deux premières couches [8]. Cela a pour effet de supprimer temporairement un neurone avec une certaine probabilité et permet d'empêcher le sur-ajustement sur les données d'entraînement.



(a) Standard Neural Network. (b) Network applying dropout

FIGURE 9 – Schéma d'un réseau de neurones perceptron mulitcouche avec et sans *dropout*

Une fois le modèle entraîné, les poids des différents neurones sont transformés en un tableau numpy et le tout est sauvegardé dans un seul fichier sous le nom de **chatbot_model.h5**.

```
# Create model - 3 layers. First layer 128 neurons, second layer 64 neurons and 3rd
# output layer contains number of neurons
# equal to number of intents to predict output intent with softmax
model = Sequential()
model.add(Dense(256, input_shape=(len(train_x[0]),), activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(len(train_y[0]), activation='softmax'))

# Compile model. Stochastic gradient descent with Nesterov accelerated gradient gives
#  good results for this model
sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])

#fitting and saving the model
hist = model.fit(np.array(train_x), np.array(train_y), epochs=200, batch_size=5, verbose=1)
model.save('chatbot_model.h5', hist)

print("model created")
```

FIGURE 10 – Construction du réseau de neurones avec Keras

Fonctions utiles pour l'interaction avec le chatbot

Cette partie présente les fonctions en charge des différents processus nécessaires au fonctionnement de l'interface graphique. Par exemple, il y a :

- La fonction **clean_up_sentence()** : Elle nettoie et pré-traite les phrases saisies dans l'interface.
- La fonction **bow()** : Elle prend la sortie de la fonction **clean_up_sentence()** et crée un sac de mots pour prédire les classes auxquelles appartiennent les différentes intentions (*intents*).
- La fonction **predict_class()** : Cette fonction produit une liste d'intentions et de probabilités, c'est-à-dire les probabilités que ces intentions correspondent aux intentions correctes. Nous utilisons un seuil d'erreur de 0,25 pour éviter un trop grand ajustement.
- La fonction **getResponse()** : Elle prend en entrée la liste des prédictions et renvoie la réponse avec la probabilité la plus élevée ;
- La fonction **chatbot_response ()** : Elle rassemble et lie les fonctions précédentes avec en entrée la phrase brute de l'utilisateur et en sortie la réponse du chatbot.

Interface graphique

Pour gérer l'interaction avec le chatbot, une interface graphique a été construite avec **tkinter**, une bibliothèque Python qui permet de concevoir des interfaces personnalisées [9]. Nous créons notamment une fonction appelée **send()** qui met en place la fonctionnalité de base duchatbot. Si le message rentré dans le chatbot n'est pas une chaîne vide, le bot produira une réponse basée sur la fonction **chatbot_response()**. Ensuite, une fenêtre de chat, une barre de défilement, un bouton pour envoyer les messages et une zone de texte pour créer le message sont construits. Les éléments sont placés dans l'espace graphique selon différentes coordonnées et hauteurs simples.

```
base = Tk()
base.title("Hello")
base.geometry("400x500")
base.resizable(width=FALSE, height=FALSE)

#Create Chat window
ChatLog = Text(base, bd=0, bg="white", height="8", width="50", font="Arial",)

ChatLog.config(state=DISABLED)

#Bind scrollbar to Chat window
scrollbar = Scrollbar(base, command=ChatLog.yview, cursor="heart")
ChatLog['yscrollcommand'] = scrollbar.set

#Create Button to send message
SendButton = Button(base, font=("Verdana",12,'bold'), text="Send", width="12", height=5,
                    bd=0, bg="#32de97", activebackground="#3c9d9b",fg='#ffffff',
                    command= send )

#Create the box to enter message
EntryBox = Text(base, bd=0, bg="white",width="29", height="5", font="Arial")
#EntryBox.bind("<Return>", send)


#Place all components on the screen
scrollbar.place(x=376,y=6, height=386)
ChatLog.place(x=6,y=6, height=386, width=370)
EntryBox.place(x=128, y=401, height=90, width=265)
SendButton.place(x=6, y=401, height=90)

base.mainloop()
```

FIGURE 11 – Code pour la création de l'interface graphique avec TKINTER

Présentation d'interactions avec le chatbot

Le chatbot peut répondre à plusieurs questions générales relatives aux salutations et à l'adieu. Il est également capable de répondre gentiment aux insultes.

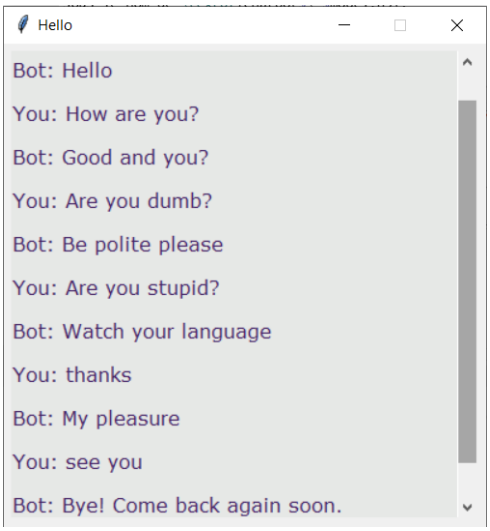


FIGURE 12 – Réaction du chatbot bag-of-words aux insultes et à l'adieu

Il propose des réponses concernant le titre, le genre, la langue, le pays, le titre original, le distributeur, le réalisateur, l'écrivain, le synopsis, le budget, les acteurs principaux, la

note moyenne, le nombre de spectateurs ayant notés le film, la note des spectateurs et la note du film. Par ailleurs, le chatbot répond NAN si l'information demandée n'existe pas dans la base de données.

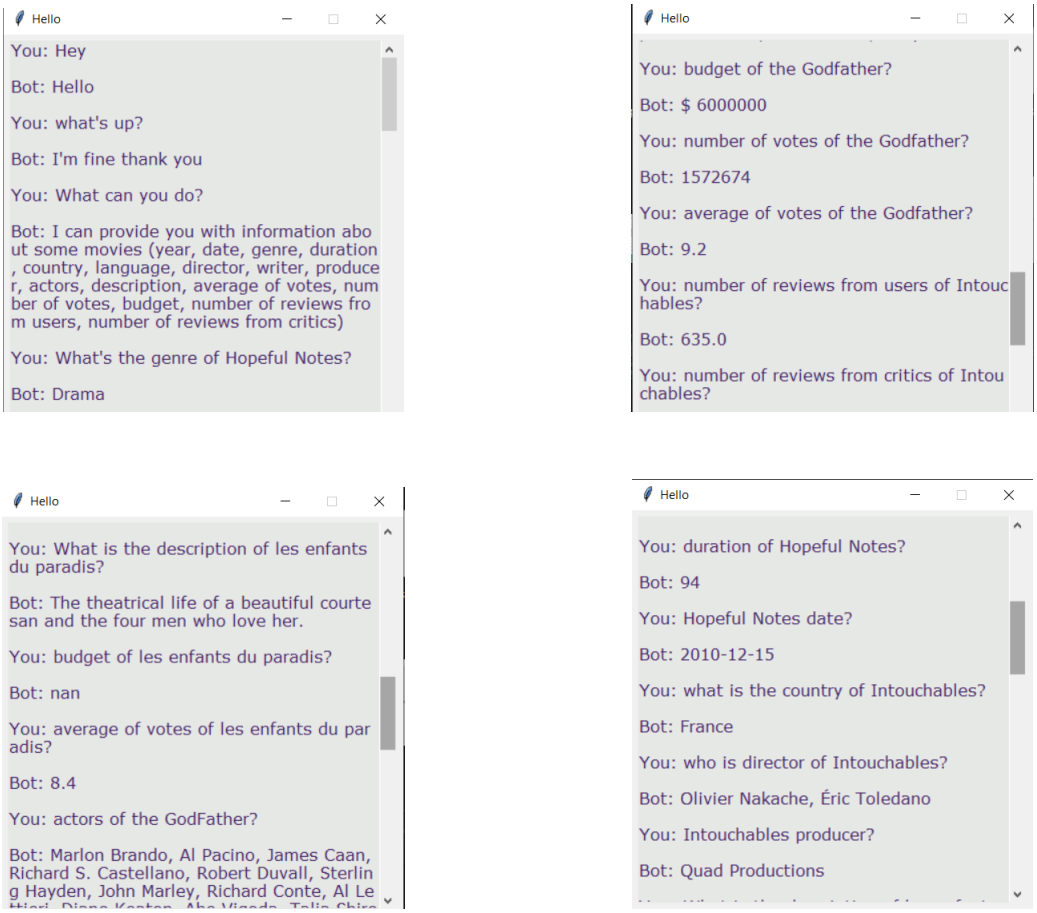


FIGURE 13 – Exemples d'interactions possibles avec le chatbot bag-of-words

6 Le modèle Generative

Concernant les données utilisées pour ce modèle, on utilisera là aussi le corpus IMDb vu précédemment. Cependant, à la place du fichier JSON utilisé dans le modèle *retrieve* de bag-of-words, il faut créer des paires questions/réponses. On génère donc un fichier TXT composé de la succession de ces paires, elles seront utilisées au cours de l'entraînement de ce modèle.

```
with open("mov_imdb.txt", "w",encoding='utf-8') as file:
    for i in range(len(mov["intents"])):
        paires = mov["intents"][i]
        qst = paires["patterns"]
        ans = paires["responses"]
        for e in qst:
            for f in ans:
                line = ''+e+'\t'+f+'\n'
                file.write(line)
```

FIGURE 14 – Code pour générer le fichier des paires questions/réponses

6.1 Théorie autour du prototype

Le coeur de ce prototype de chatbot génératif est un modèle *sequence-2-sequence*. L'intérêt de ce modèle est de pouvoir prendre en entrée une séquence de longueur variable et de retourner en sortie une nouvelle séquence dépendante de l'entrée, elle aussi de taille variable [10].

Pour cela, on utilise deux réseaux de neurones récurrents séparés. Le premier se comporte comme un encoder : il transforme la séquence en entrée en un vecteur de taille fixe appelé *context*. Ce vecteur contient l'ensemble des informations sémantiques de la phrase en entrée. Le second RNN agit lui comme un decoder : il prend en entrée ce vecteur *context* et un mot d'entrée et il renvoie une proposition pour le mot suivant dans la séquence générée. Ce mot généré est récupéré et utilisé comme entrée de la cellule suivante.

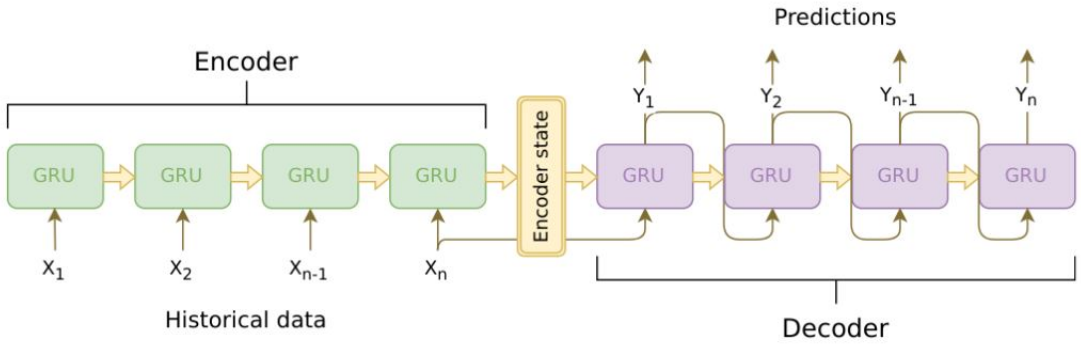


FIGURE 15 – Illustration du fonctionnement d'un système Encoder-Decoder

Encoder

Un encoder itère sur les tokens de la séquence d'entrée, en les prenant un par un et en renvoyant à chaque étape un vecteur de sortie et un vecteur "hidden state". Ce vecteur est transmis lors de l'itération du token suivant. Ce processus est au coeur de la compréhension des séquences et permet de rendre compte le contexte des mots précédents. A la fin du processus, on obtient le vecteur *context* qui synthétise le sens de la phrase.

Dans notre cas, nous avons utilisé une variante bidirectionnelle du réseau de neurones récurrent GRU (Gated Recurrent Unit). Cet Encoder est donc composé de deux RNN indépendants, avec l'un parcourant les tokens de gauche à droite dans le sens de lecture et l'autre de gauche à droite. Cela permet d'encoder à la fois le contexte passé et futur d'un terme.

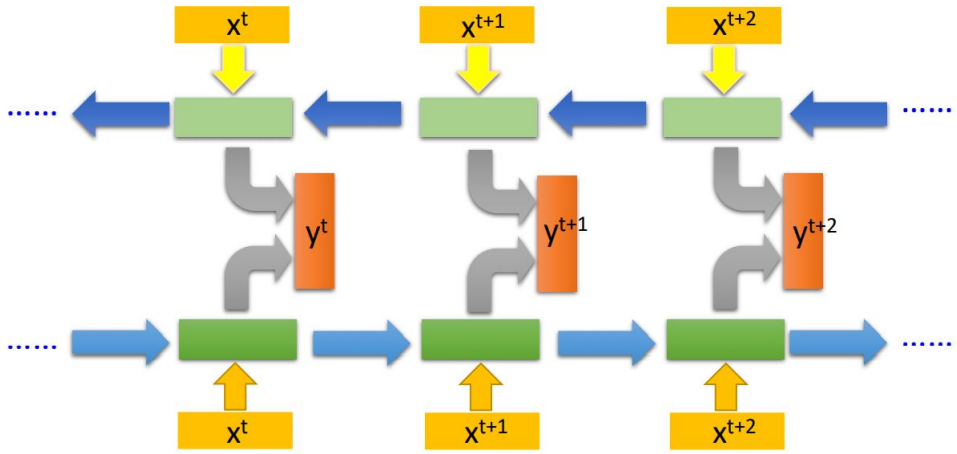


FIGURE 16 – Schéma explicatif d'un réseau de neurones récurrent bidirectionnel

Decoder

Pour sa part, le decoder génère une réponse à la phrase en entrée de l'encoder mot par mot. Pour cela, il récupère le vecteur *context* et les états cachés des différentes cellules du decoder pour générer le mot suivant dans la phrase. Il continue jusqu'à ce que la génération produise un token <EOS>, indiquant la fin de la phrase.

Un problème fréquent rencontré par les decoder de modèles seq2seq est la forte dépendance au vecteur *context*. Les pertes d'informations sont fréquentes, notamment lorsque les séquences en entrée de l'encoder sont longues. Pour limiter cela, on utilisera donc un mécanisme d'attention, qui permet au decoder de se concentrer spécifiquement à certaines parties du vecteur *context* plutôt qu'à son entièreté. En effet, certaines parties du vecteur rendent plus compte du sens de la phrase en entrée que d'autres s'attachant aux détails.

Loss function

Sachant que les séquences générées ne sont pas toutes de la même longueur initialement, on utilise un *padding* pour contrer ce point. Dès lors, il est impossible de considérer tous les éléments du tenseur lors du calcul de la perte. Ainsi, on définit le calcul de perte sur la base du tenseur de sortie de notre decoder, du tenseur cible et d'un tenseur de masque binaire décrivant le remplissage du tenseur cible. Cette fonction de perte calcule la log-vraisemblance négative moyenne des éléments qui correspondent à un 1 dans le tenseur de masque.

6.2 Construction et entraînement du modèle

Pour l'entraînement du modèle, nous allons utiliser deux astuces pour faciliter la convergence :

— le *teacher forcing* [11] : à une certaine probabilité, on utilisera le mot attendu comme input des cellules du Decoder à la place du mot en sortie de la cellule précédente. Il faut cependant veiller à utiliser une probabilité adaptée pour éviter l'instabilité du modèle et des faux-semblant de convergence

— le *gradient clipping* : il s'agit ici d'empêcher le gradient de dépasser une certaine valeur pour éviter les soucis fréquents d'explosion du gradient.

Ensuite, l'entraînement se réalise selon le schéma suivant :

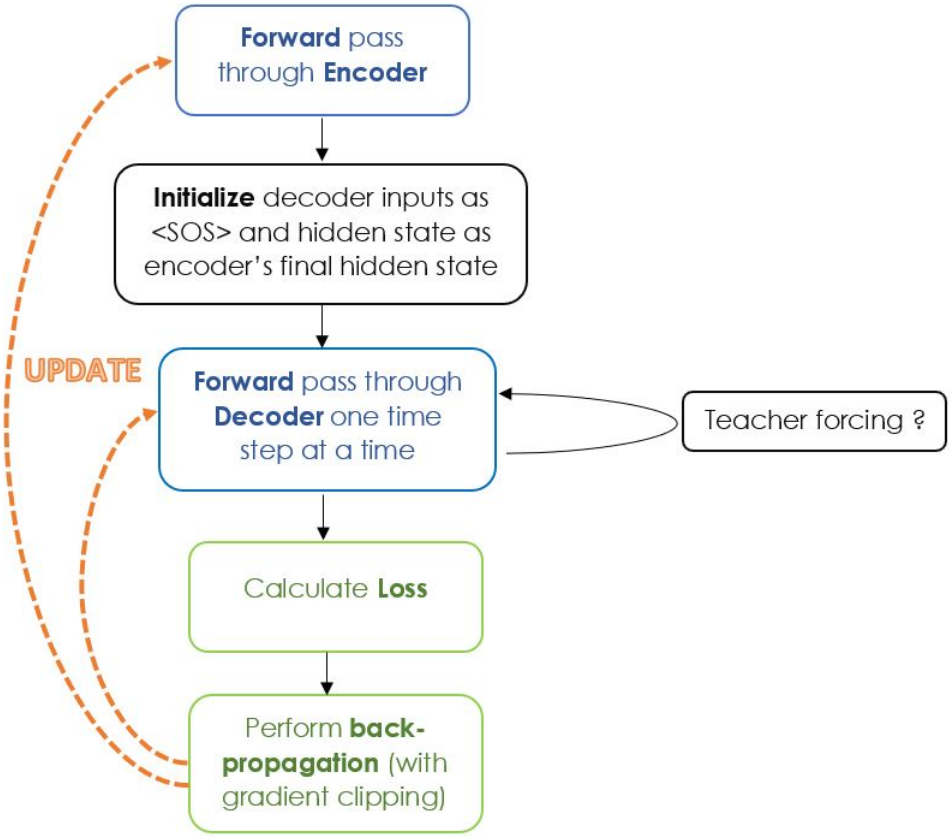


FIGURE 17 – Processus d'entraînement du réseau de neurones du modèle génératif

6.3 Présentation des résultats

Après avoir développé le modèle, on réalise l'entraînement expliqué dans la partie précédente. Pour suivre son évolution, on trace le graphique de la fonction de perte au cours des *epochs* de l'entraînement, les valeurs affichées correspondant à la valeur moyenne de la perte calculée sur la séquence entière. Ainsi on a la figure suivante :

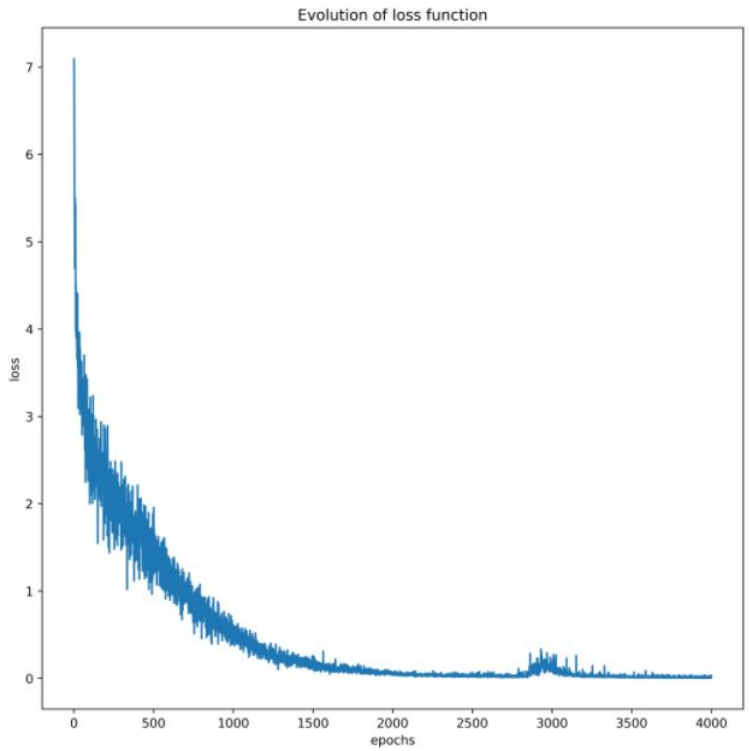


FIGURE 18 – Progression de la fonction de perte au cours des *epochs* d'entraînement

Intéressons nous maintenant aux résultats obtenus :

```
In [27]: evaluateInput(encoder, decoder, searcher, voc)
> hello
Bot: hello you ?
> how are you ?
Bot: i m fine thank you you you
> genre of american beauty
Bot: drama
> country of american beauty
Bot: usa
> language of american beauty
Bot: english
> producer of pope
Bot: film village
> language of pope
Bot: english
> see you
Bot: bye ! come back again soon . dellums
> quit
```

FIGURE 19 – Exemple de discussion avec le chatbot génératif

Ce chatbot semble capable de répondre aux mêmes questions que le modèle *retrieve* avec les sacs de mots. Cependant on remarque parfois que des mots supplémentaires apparaissent du fait que le modèle génère lui-même les mots et peut donc parfois en ajouter ou retirer un ou deux par rapport à l'attendu. Cela s'explique par l'utilisation d'un *greedy search decoder* alors que d'autres choix pourraient aboutir à de meilleurs résultats. D'autres limites seront discutés dans la partie suivante.

7 Evaluation des modèles

Modèle retrieve TF-IDF

Il est assez complexe de trouver des règles universelles pour l'évaluation des prototypes créés. Au lieu de s'intéresser à des scores métriques, on a choisi pour ce modèle d'évaluer sa pertinence et ses capacités à travers un test grandeur nature du chatbot par 2 personnes ignorant tout des principes technologiques sous-jacents. Le prototype a notamment été proposé sans guide d'utilisation pour estimer son caractère instinctif.

De manière générale, les retours obtenus sont plutôt positifs. Les fonctionnalités présentées avec la première ligne de dialogue sont bien disponibles et conformes aux attentes. L'outil est facile à prendre en main et son usage est pertinent pour découvrir des informations sur les films de la base de données. Les deux testeurs affirment notamment qu'ils seraient susceptibles de l'utiliser pour leur consommation personnelle de contenu cimématographique.

Néanmoins, deux points d'amélioration principaux ont été relevés :

— Tout d'abord, il arrive que le chatbot ne réponde pas correctement à la question de l'utilisateur. Cela traduit la forte dépendance du modèle aux questions pré-établies dans le fichier JSON. Pour améliorer ce point, une phase de test conséquente est nécessaire pour relever les remarques de chacun et les échecs observés face aux questions pour pouvoir supprimer certaines questions, en ajouter de nouvelles et obtenir une liste encore plus pertinente.
— L'aspect très robotique des conversations peut également mettre mal à l'aise l'utilisateur. Contrairement à l'autre modèle retrieve, il n'y a qu'une seule réponse possible à un type de question ce qui rend les discussions avec le chatbot "sans vie". Ce problème est inhérent au formattage des réponses en sortie.

Modèle retrieve bag-of-words

La façon idéale pour évaluer un chatbot est de mesurer s'il remplit pleinement sa tâche, par exemple résoudre un problème de support client, dans une conversation donnée. Mais de telles étiquettes sont coûteuses à obtenir car elles nécessitent un jugement et une évaluation humaine. Parfois, il n'y a pas d'objectif bien défini, comme c'est le cas avec les modèles à domaine ouvert.

Dans notre cas, le chatbot a généré des réponses correctes pour tous les cas de test que nous avons fait et il permet d'avoir une discussion vive avec l'utilisateur en proposant plusieurs réponses. Il existe aussi des méthodes d'évaluation en python. Nous avons utilisé la métrique **précision** dans ce cas pour évaluer la capacité du système à classer les réponses pertinentes parmi les k réponses les mieux classées par le chatbot. Nous avons eu une précision de 0.6998 avec le modèle proposé.

```
Epoch 200/200
2127/2127 [==============================] - 13s 6ms/step - loss: 1.4138 -
accuracy: 0.6998
model created
```

FIGURE 20 – Précision du modèle retrieve bag-of-words

Pour améliorer la précision, nous pourrions ajouter des couches cachées avec plus de neurones ou fournir plus de données d'entraînement. En effet, ce dernier point a surement un impact important : un nombre adapté d'intentions peut conduire à une solution de chatbot puissante. Un autre axe d'amélioration est l'ajout d'un correcteur automatique car le modèle proposé ne répond pas s'il n'a pas compris une question ou s'il y a des fautes d'orthographe dans le titre d'un film donné. Ainsi, nous pourrions pensé par la suite à utiliser un correcteur pour améliorer l'expérience utilisateur.

Modèle génératif

Pour le modèle génératif, l'évaluation est encore plus délicate. Alors qu'un modèle retrieve récupère un choix parmi des réponses prédéfinies, ce modèle génère les réponses depuis le corpus grâce aux *embeddings*. D'ailleurs, l'évaluation des modèles génératifs est toujours un sujet de recherche.

Cependant, pour notre modèle on peut déjà citer quelques limitations et comment on pourrait y remédier :

— Pour améliorer la précision du modèle, on pourrait augmenter la taille des *embeddings* ou même utiliser un modèle pré-entraîné comme *en_core_web_lg* de spacy.
— Si la phrase rentrée par l'utilisateur contient un mot que le modèle n'a pas rencontré lors de la phase d'entraînement, il ne saura pas le traiter et il renverra un message tel que : "Error : Encountered unknown word". Il faudrait donc un corpus suffisamment grand pour résoudre ce soucis.
— Le modèle présente des difficultés à générer de longues réponses, par exemple ce modèle va renvoyer une chaine de caractère vide si on lui demande la description d'un film.

Conclusion

Au cours de ce projet d'option informatique, l'équipe s'est attaché à concevoir un prototype de chatbot permettant de répondre à des questions relatives au cinéma et en particulier aux informations générales sur de nombreux films (genre, acteurs, box-office, etc ...). Pour réaliser cette tâche, trois techniques différentes ont été implémentées, chacune avec leurs avantages et leurs défauts :

- un premier modèle de type *retrieve*, utilisant des calculs de similarité entre les phrases selon des vecteurs traduisant la fréquence d'apparition des mots ;
- un second modèle *retrieve*, dans lequel les phrases ont été transformées en sacs de mots et utilisant un réseau de neurones pour apprendre intelligemment les réponses à renvoyer à l'utilisateur ;
- un modèle génératif, utilisant des techniques de *sequence-to-sequence* avec des associations de réseaux de neurones récurrents, afin de pouvoir générer entièrement des réponses plutôt que d'exploiter des solutions pré-définies.

Les trois prototypes se sont avérés globalement plutôt efficaces. Tandis que la technologie TF-IDF permet d'obtenir de manière simple des résultats acceptables, la solution "bag-of-words" et réseau de neurones présente une meilleure performance. Enfin, la solution *generative* renvoie des réponses majoritairement correctes, malgré des soucis de gestion des nouveaux termes et de la génération de longues séquences.

Références

[1] CloudBoost, Generative vs Retrieval Based Chatbots, Leah Fainchtein, URL : https://blog.cloudboost.io/generative-vs-retrieval-based-chatbots-a-quick-guide-8d19edb1d645

[2] TensorFlow Tutorials, Word2Vec, URL : https://www.tensorflow.org/tutorials/text/word2vec

[3] Cornell Movie-Dialogs Corpus, Cristian Danescu, URL : https://www.cs.cornell.edu/ cristian/Cornell_Movie-Dialogs_Corpus.html

[4] Kaggle Website, IMDb movies extensive dataset, Stefano Leone, URL : https://www.kaggle.com/stefanoleone992/imdb-extensive-dataset

[5] Medium, Text Preprocessing for NLP, Ujjawal Verma, URL : https://medium.com/analytics-vidhya/text-preprocessing-for-nlp-natural-language-processing-beginners-to-master-fd82dfecf95

[6] A New Term Frequency Normalization Model for Probabilistic Information Retrieval, Fanghong Jian, Jimmy Xiangji Huang, Jiashu Zhao and Tingting He, URL : https://www.researchgate.net/publication/326137319-A-New-Term-Frequency-Normalization-Model-for-Probabilistic-Information-Retrieval

[7] TowardsDataScience, Simple Explanation of the BoW Model, Victor Zhou, URL : https://towardsdatascience.com/a-simple-explanation-of-the-bag-of-words-model-b88fc4f4971

[8] Computer Science Departement, University of Toronto, Yoshua Bengio, URL : https://www.cs.toronto.edu/ hinton/absps/JMLRdropout.pdf

[9] Python GUI Programming With Tkinter, David Amos, URL : https://realpython.com/python-gui-tkinter/

[10] Analytics Vidhya, Sequence to Sequence modelling with Attention, URL : https://www.analyticsvidhya.com/blog/2018/03/essentials-of-deep-learning-sequence-to-sequence-modelling-with-attention-part-i/

[11] Machine Learning Mastery, Teacher Forcing for Recurrent Neural Networksk, URL : https://machinelearningmastery.com/teacher-forcing-for-recurrent-neural-networks/