Ecole Nationale Supérieure d'Informatique et d'Analyse des Systèmes - RABAT

Rapport de PFA : Healthcare Chatbot design

Réalisé par :

El Mehdi Oudaoud
Fatima-Ezzahra Lahnine

Encadré par :

Pr. Azeddine el hassouny

Année Scolaire 2020/2021

Résumé

Ce Projet propose un design de chatbot pour soins de santé qui répond aux questions les plus fréquemment posées et fait une vérification des symptômes entrées par les clients.

Ce bots permette les utilisateurs de décrire leurs symptômes et leur état de santé en posant une suite de questions. Le bot ensuit comprend les inputs de l'utilisateur en utilisant le traitement du langage naturel (NLP), quelle que soit la variation de l'entrée. Cela est essentiel pour obtenir des réponses précises.

Le bot est intégré dans une application web qui permet l'utilisateur de préciser une partie du corps concernant sa demande, et liste les différents symptômes liés à cette partie. Il peut choisir une des réponses listées qui va être transmise au bot, puis ce dernier répond le client par la description du symptôme choisi et lui donne quelque précautions et conseils à prendre.

Abstract

This project proposes a healthcare chatbot design that answers frequently asked questions and verifies the symptoms entered by customers.

The bot allows users to describe their symptoms and conditions by asking a series of questions. The bot then understands the user's input using natural language processing (NLP), regardless of the variation in the input. This is essential to get accurate answers.

The bot is embedded in a web application that allows the user to specify a body part regarding his query, and lists the different symptoms related to that part. He can choose one of the listed answers which will be transmitted to the bot, then the latter responds to the customer with a description of the chosen symptom and gives him some precautions and advice to take.

Table des matières

Introduction

Les chatbots sont considérés comme une alternative plus fiable et plus précise aux recherches en ligne que les patients effectuent lorsqu'ils essaient de comprendre la cause de leurs symptômes. Les prestataires de soins de santé pensent que les chatbots pourraient aider les patients qui ne sont pas sûrs de savoir où ils doivent aller pour recevoir des soins.

Les prestataires de soins de santé mettent actuellement en œuvre des bots qui permettent aux utilisateurs de vérifier leurs symptômes et de comprendre leur état de santé depuis le confort de leur domicile. Les chatbots qui utilisent le traitement du langage naturel (NLP) peuvent comprendre les demandes des patients, quelle que soit la variation de l'entrée. Cela est essentiel pour atteindre une grande précision dans les réponses, ce qui est indispensable pour les vérificateurs de symptômes.

Grâce à la connaissance de l'entrée, le bot peut évaluer les informations et aider les utilisateurs à déterminer la cause de leurs symptômes. Avec toutes les données fournies par le robot, les utilisateurs peuvent déterminer si un traitement professionnel est nécessaire ou si des médicaments en vente libre sont suffisants.

Chapitre 1

Architectures générales des chatbots

Un chatbot est un système de dialogue homme-ordinateur via le langage naturel. Il s'agit donc d'un humain ayant une conversation naturelle avec un ordinateur ou un système.

Le chatbot doit être capable de dialoguer et de comprendre l'utilisateur ; on pourrait dire qu'il s'agit d'une fonction de compréhension. Cette compréhension comprend la reconnaissance des intentions et des entités. Les intentions peuvent être considérées comme des verbes et les entités comme des noms. Les robots basés sur le texte ont au minimum un composant de compréhension du langage naturel (NLU).

Toute l'intelligence ne relève pas des capacités de la NLU. Les robots doivent avoir accès à une base externe de connaissances et de bon sens via des API, de sorte qu'ils puissent assurer la fonction de compétence en répondant aux questions des utilisateurs. Enfin, l'agent incarné doit assurer une présence très fonctionnelle. Ironiquement, ces agents numériques n'existaient pas jusqu'à récemment et étaient considérés comme très facultatifs. Aujourd'hui, cette fonction s'avère cruciale dans le cas des utilisateurs ordinaires.

1.1 Conversion parole-texte

1.1.1 Reconnaissance vocale à vocabulaire étendu

La parole est l'un des modes de communication les plus naturels et les plus puissants, et elle est largement acceptée comme l'avenir de l'interaction avec les applications informatiques et mobiles. Elle fait partie de la nature humaine au point que des personnes dont le IQ est inférieur à 50 et dont le cerveau est trois fois plus petit que celui des humains peuvent parler. La recherche neurologique indique que la parole occupe une plus grande partie du cerveau que toute autre fonction de traitement. En intégrant le traitement de la parole, les chatbots seront capables de s'interfacer avec les téléphones et les radios.

La conversion de la parole en texte commence par un processus appelé reconnaissance automatique de la parole (ASR). Les améliorations de la LVCSR peuvent être mesurées selon un certain nombre de critères :

1. **Volume du vocabulaire :** Les vocabulaires, qui étaient à l'origine minuscules et ne comprenaient que des phrases de base (par exemple, oui, non, chiffres, etc.), comprennent aujourd'hui des millions de mots dans de nombreuses langues.
2. **Indépendance du locuteur :** la capacité à reconnaître certains locuteurs. Ceci n'est pertinent que si les chatbots utilisent l'identité du locuteur pour générer des réponses spécifiques à l'utilisateur.
3. **Coordination de l'articulation :**la capacité de traiter un flux continu de mots, qui ne contient pas nécessairement de pauses entre les mots. Il faut une tokénisation et une segmentation appropriées du flux d'entrée, abordées dans la section qui suit.
4. **Gestion du bruit :**la capacité de filtrer le bruit (par exemple, la circulation, la musique de fond ou la parole).
5. **Microphone :**la capacité de traiter la parole à différentes distances du microphone.

1.1.2 Automatic Speech Recognition (ASR)

La reconnaissance vocale ou Speech-To-Text (STT) est un processus de conversion de la parole audio en texte. L'objectif de la ASR est de parvenir à une reconnaissance vocale indépendante du locuteur et portant sur un large vocabulaire. Alors que les chatbots peuvent s'offrir le luxe de s'adresser à un domaine très étroit, le

STT/ASR doit être capable de traiter un large vocabulaire. S'assurer que tout ce qui est dit peut être converti en texte. Le chatbot peut ne pas être en mesure de répondre directement à la requête ou à la demande. Ce qui pourrait tomber en dehors du domaine du chatbot. Mais l'ASR doit au moins présenter un texte précis à la section chatbot/NLU.

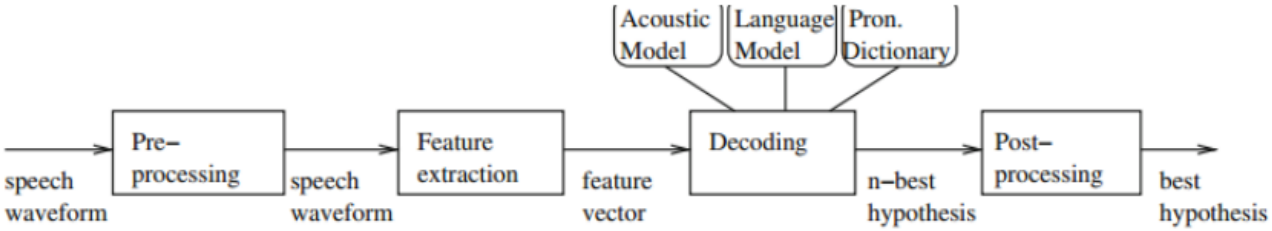

FIGURE 1.1 – Processus de la reconnaissance automatique de la parole

Premièrement, nous avons besoin d'un modèle acoustique qui nous donnera P(X|W) - étant donné un mot, la probabilité que nous entendions un son. Nous trouverons la valeur argmax sur tous les mots de notre langue w, ou le mot ayant la probabilité la plus élevée de représenter ce son. Les modèles acoustiques sont généralement formés sur des enregistrements sonores et les transcriptions correspondantes, qui peuvent être utilisés pour trouver de manière empirique ces probabilités. La représentation statistique de chaque mot (ou phonème) générée par l'analyse du corpus sonore est typiquement représentée par un modèle de Markov caché (HMM). Les HMM sont des processus de Markov où les états sont non observés/cachés (nous ne connaissons pas les phonèmes/mots réels utilisés, et nous nous approximation de cette information).

Deuxièmement, nous avons besoin d'un modèle de langage qui puisse nous indiquer la probabilité d'entendre un mot donné dans notre langue.

Troisièmement, nous avons besoin d'un dictionnaire contenant une liste de mots et de leurs phonèmes. Étant donné un son, nous pouvons décoder le mot en utilisant la règle de Bayes :

$$W = argmax_{(W \in L)} \frac{P(X|W)P(W)}{P(X)}$$

La quatrième étape est le post-traitement ; la règle de Bayes nous donne une liste des séquences de mots probables avec leur rang, et nous choisissons la plus probable comme étant la séquence de mots que nous avons entendue. Les autres hypothèses les plus probables sont stockées et peuvent être utilisées plus tard dans des algorithmes d'apprentissage par renforcement, où elles seront utilisées pour apprendre et corriger les erreurs de la phase ASR.

1.2 Natural Language Processing (NLP)

La compréhension du langage naturel est à la base des capacités du chatbot. Sans la détection des entités et la reconnaissance des intentions, tous les efforts déployés pour comprendre l'utilisateur sont vains.

La plupart des architectures de chatbot se composent de quatre piliers, à savoir les intentions, les entités, le flux de dialogue (machine à états) et les scripts.

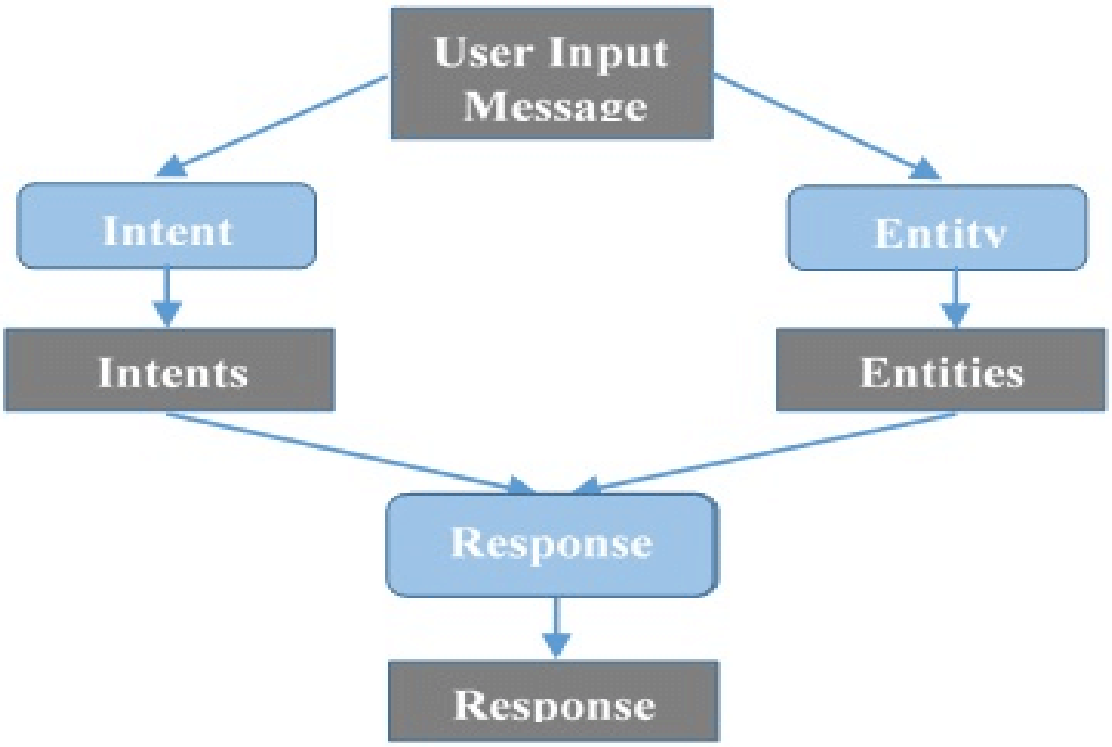

FIGURE 1.2 – Architecture traditionnelle des chatbots

Le dialogue contient les blocs ou états entre lesquels l'utilisateur navigue. Chaque dialogue est associé à une ou plusieurs intentions et/ou entités. Des variables de session peuvent également être utilisées pour décider quels états ou nœuds doivent être visités.

Les intentions et les entités constituent la condition d'accès à ce dialogue. La boîte de dialogue contient la sortie vers le client sous la forme d'un script, d'un message... ou d'une formulation si vous voulez.

C'est l'une des tâches les plus ennuyeuses et les plus laborieuses de la création d'un chatbot. Elle peut devenir complexe et les modifications apportées dans un domaine peuvent avoir un impact sur un autre domaine par inadvertance. Un manque de cohérence peut également conduire à des expériences utilisateur non planifiées. La mise à l'échelle de cet environnement est délicate, en particulier si on souhaite l'étendre à une grande organisation.

Dans la suite, on va introduire les techniques classiques de représentation des données textuelles.

1.2.1 Bag of Words (BoW)

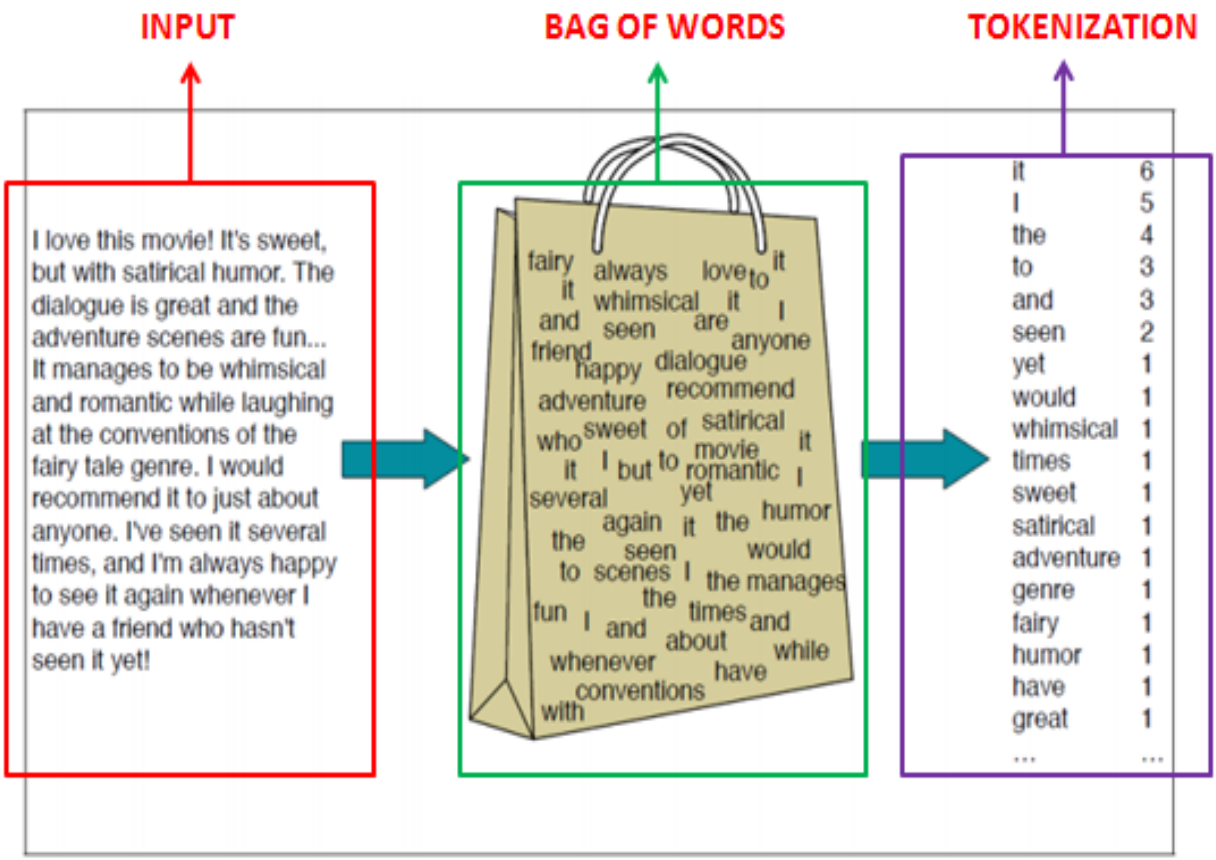

FIGURE 1.3 – Étape de base des algorithmes de classification de texte

Bag of Words (BOW) est un algorithme qui compte le nombre de fois qu'un mot apparaît dans un document. Ces nombres de mots nous permettent de comparer des documents et d'évaluer leurs similitudes pour des applications telles que la recherche, la classification de documents et la modélisation de sujets. Il est nommé ainsi car il ne concerne que l'occurrence du mot et non l'endroit où il est placé (c'est-à-dire l'ordre) dans le sac. De plus, l'intuition derrière une telle approche est que des documents similaires contiennent des mots similaires.

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

Une meilleure approche sera de créer un vocabulaire de mots groupés. Cela changera la taille du vocabulaire et permettra au sac de mots de capturer un peu plus de sens du document. La création d'un vocabulaire de paires de deux mots s'appelle un modèle bi-gramme. Un vocabulaire de triplés de mots est appelé un modèle de tri-gramme. Par exemple, les bi-grammes de "the cat sat in hat with" sont les suivants :

— "cat sat"
— "sat in"
— "in hat"
— "hat with"
— "the cat"

L'approche générale est appelée le modèle n-gramme, où n est la référence du nombre de mots groupés.

Avantages :simple à comprendre et à implémenter.

Inconvénients :
— Le vocabulaire se compose de tous les mots présents dans la base de données. Ainsi si les données sont trop volumineuses et contiennent de nombreux mots uniques, cela peut créer des matrices très larges et sparse ce qui augmente à la fois la complexité spatiale et temporelle.
— Le modèle BOW ne concerne que l'occurrence du mot et non l'endroit où il est placé (c'est-à-dire l'ordre). Cela conduit à la perte d'informations contextuelles et donc à la signification des mots dans le document (sémantique).

1.2.2 Tf-idf Vectorization

Tf-Idf est l'abréviation du terme fréquence-fréquence de document inverse. Donc, deux choses : la fréquence des termes et la fréquence inverse des documents.
La fréquence du terme (TF) est essentiellement la sortie du modèle BoW. Pour un document spécifique, il détermine l'importance d'un mot en regardant à quelle fréquence il apparaît dans le document. Si un mot apparaît plusieurs fois, le mot doit être important et sa fréquence grande.

Sa formule mathématiques est :

$$tf(t,d) = \frac{f_{t,d}}{\sum_{t'} f_{t',d}}$$

avec $f_{t',d}$ le nombre d'occurrences du mot t, dans le document d.
IDF (Inverse Document Frequency) utilisé pour calculer le poids des mots rares dans tous les documents. Les mots qui apparaissent rarement dans le corpus ont un score IDF élevé. Cependant, on sait que certains termes, tels que « je », « a » peuvent apparaître plusieurs fois mais ont peu d'importance. Nous devons donc alourdir les termes fréquents tout en augmentant les rares. Sa formule mathématique est donnée par :

$$idf(t,D) = log(\frac{N}{n_t})$$

avec t le mot dans le corpus D et N la taille du corpus D et n_t le nombre de document contenant t.

Ainsi la formule mathématique pour TF-Idf est :

$$tf - idf(t,d,D) = tf(t,d) \times idf(d,D)$$

Avantages :
— Facile à calculer.
— On dispose d'une métrique de base pour extraire les termes les plus descriptifs d'un document.
— On peut facilement calculer la similitude entre 2 documents en l'utilisant.

Chapitre 2

Le traitement de langage naturel avec des réseau de neurones

2.1 RNN

Les réseaux de neurones récurrents ou RNN sont une variante très importante des réseaux de neurones largement utilisés dans le traitement du langage naturel.

Conceptuellement, ils diffèrent d'un réseau de neurones standard car l'entrée standard dans un RNN est un mot au lieu de l'échantillon entier comme dans le cas d'un réseau de neurones standard. Cela donne au réseau la flexibilité de travailler avec des longueurs de phrases variables, ce qui ne peut pas être réalisé dans un réseau neuronal standard en raison de sa structure fixe. Il offre également un avantage supplémentaire de partage des fonctionnalités apprises à travers différentes positions de texte qui ne peuvent pas être obtenues dans un réseau neuronal standard.

Un RNN traite chaque mot d'une phrase comme une entrée distincte se produisant au temps « t » et utilise également la valeur d'activation à « t-1 » comme une entrée en plus de l'entrée au temps « t ».

2.1.1 Architectures des RNN

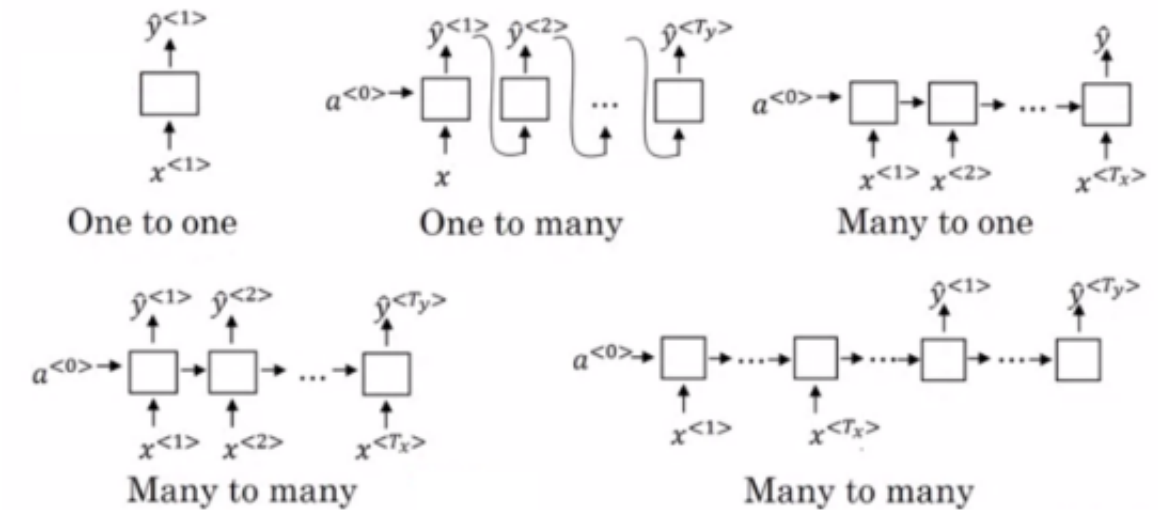

FIGURE 2.1 – Les différentes architectures des RNN

Il existe différentes architectures des RNN :

— **Plusieurs à plusieurs :**) c'est-à-dire nombre d'entrées = nombre de sorties. Une telle structure est très utile dans la modélisation de séquence.
— **Plusieurs à un RNN :** l'architecture plusieurs à un fait référence à une architecture RNN où de nombreuses entrées (Tx) sont utilisées pour donner une sortie (Ty). Un exemple approprié d'utilisation d'une telle architecture sera une tˆache de classification.
— **Un à plusieurs :** l'architecture un à plusieurs se réfère à une situation dans laquelle un RNN génère une série de valeurs de sortie basées sur une seule valeur d'entrée. Un excellent exemple d'utilisation d'une telle architecture sera une tâche de génération de musique, où une entrée est un jounre ou la première note.
— **Plusieurs à plusieurs** (Tx n'est pas égal à Ty) : Cette architecture fait référence à l'endroit où de nombreuses entrées sont lues pour produire de nombreuses sorties, où la longueur des entrées n'est pas égale à la longueur des sorties. Les tâches de traduction automatique constituent un excellent exemple d'utilisation d'une telle architecture. Le codeur fait référence à la partie du réseau qui lit la phrase à traduire, et le décodeur est la partie du réseau qui traduit la phrase dans la langue souhaitée.

2.1.2 Limitations de RNN

Outre toute son utilité, RNN présente certaines limitations dont les principales sont :

Les exemples d'architecture RNN mentionnés ci-dessus sont capables de capturer les dépendances dans une seule direction du langage. Fondamentalement, dans le cas du traitement automatique du langage naturel, il suppose que le mot suivant n'a aucun effet sur la signification du mot précédent. Avec notre expérience des langues, nous savons que ce n'est certainement pas vrai.

Les RNN ne sont pas non plus très bons pour capturer les dépendances à long terme et le problème de la disparition des gradients refait surface dans RNN.

2.2 LSTM

Les réseaux LSTM sont un type de RNN, capable d'apprendre les dépendances à long terme. Les réseaux LSTM représentent la majorité des RNN utilisés aujourd'hui. Face au problème de "vanishing gradient" des RNNs les LSTMs sont présentées comme solution. D'ailleurs la mémoire à long terme est leur comportement par défaut il ne faut pas apprendre à ce réseau de le faire.

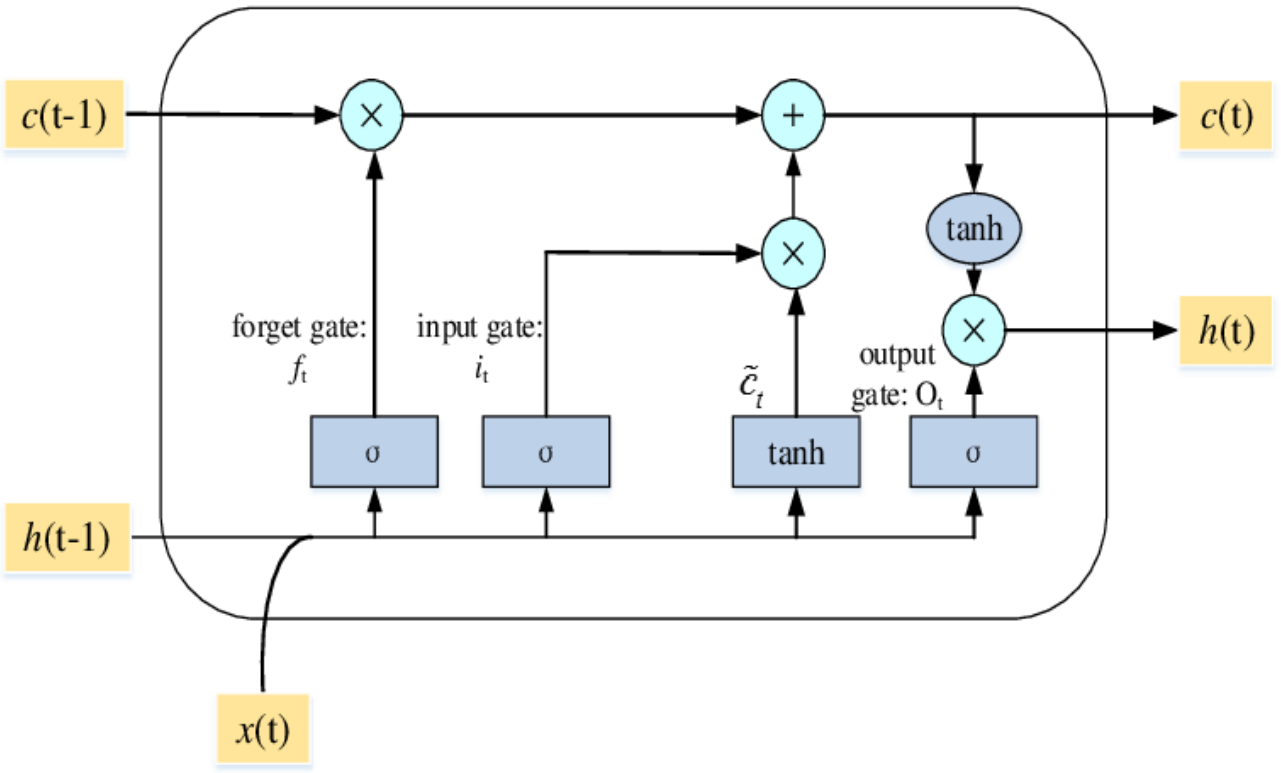

FIGURE 2.2 – La structure de l'unité LSTM

Dans un LSTM il existe trois portes : La porte d'entrée et la porte d'oubli déterminent comment le prochain état est influencé respectivement par l'entrée et le dernier état . La porte de sortie détermine comment la sortie du réseau est influencée par l'état. Parfois, il est utile de se souvenir des détails importants, mais de ne les utiliser que plus tard.

2.3 CNN

Lorsque nous entendons parler de CNN (Convolutional neural networks), nous pensons généralement à la vision par ordinateur. Les CNN ont été responsables de majeures avancées dans la classification des images et sont aujourd'hui au cœur de la plupart des systèmes de vision par ordinateur, du marquage automatique des photos de Facebook aux voitures autonomes. Plus récemment, nous avons également commencé à appliquer les CNN à des problèmes de traitement du langage naturel et obtenu des résultats intéressants pour des problème de catégorisation des phrases.
Le fonctionnement des CNN dans NLP ne diffère pas trop de leur fonctionnement sur les images, car ils prennent en entrée une matrice (moralement une image cf la figure ci-dessous) dont les lignes constituent la représentation vectorielle des différents tokens qui composent une phrase du corpus. matrice à utiliser comme entrée. Le les modèles n'ont pas besoin d'être complexes pour obtenir des résultats solides. Les filtres utilisés dans ce type d'application du CNN ont une largeur égale à la dimension du modèle utilisé pour faire la vectorisation des tokens.

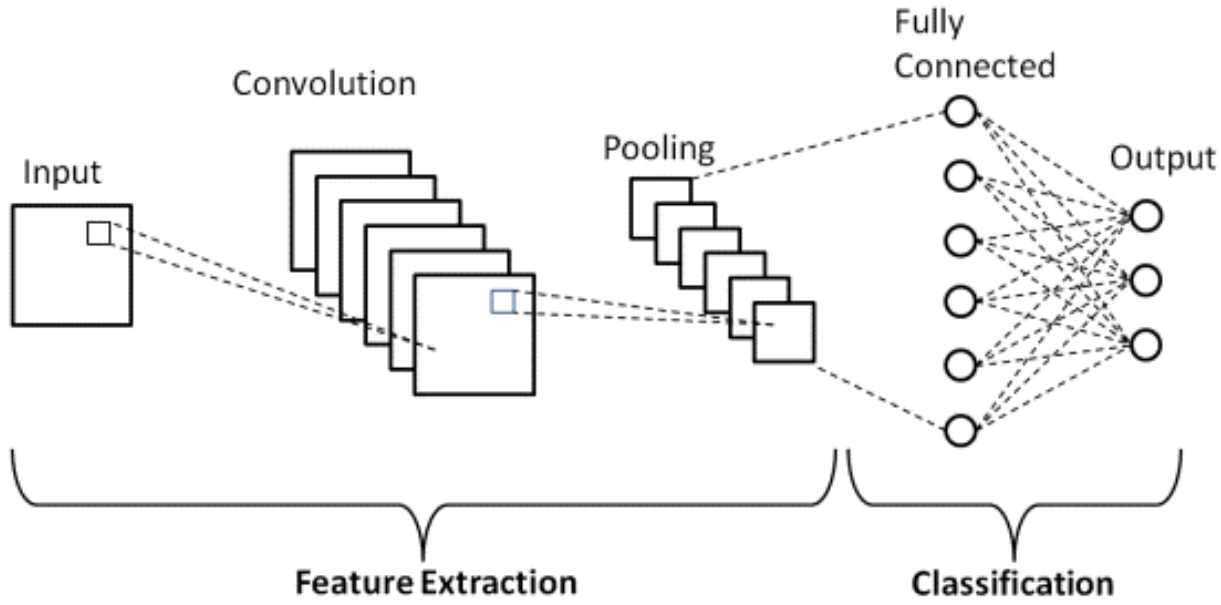

FIGURE 2.3 – Architecture basique du CNN

La figure suivante montre une configuration de base, d'autres configurations proposent l'emploie des canaux multiples en combinant plusieurs représentation vectorielles, ou juste en faisant la concaténation des ces représentations, d'autres proposent d'utiliser un entraînement statique (i.e fixer la matrice d'entrée).

2.4 Le modèle BERT

L'acronyme BERT correspond à "Bidirectional Encoder Representations from Transformers" a été développé par les chercheurs de Google AI . Comme son nom l'indique, il s'agit d'un modèle de représentation du langage qui s'appuie sur un module qu'on appelle "Transformer" pour le traitement naturel du langage (NLP).La principale innovation technique de BERT consiste à pré-entraîner des représentations bidirectionnelles profondes à partir de texte non étiqueté en conditionnant conjointement le contexte gauche et droit. En conséquence, le modèle BERT pré-entraîné peut être affiné avec une seule couche de sortie supplémentaire pour créer des modèles de pointe pour un large éventail de tâches NLP. Cela contraste avec les méthodes précédentes comme une combinaison d'apprentissage de gauche à droite et de droite à gauche.

2.4.1 L'Architecture de BERT

L'architecture BERT s'appuie sur Transformer. Nous avons actuellement deux variantes disponibles :
— **BERT Base :** 12 layers (transformer blocks), 12 attention heads, and 110 million parameters.
— **BERT Large :** 24 layers (transformer blocks), 16 attention heads and, 340 million parameters.

2.4.2 Traitement préalable de texte

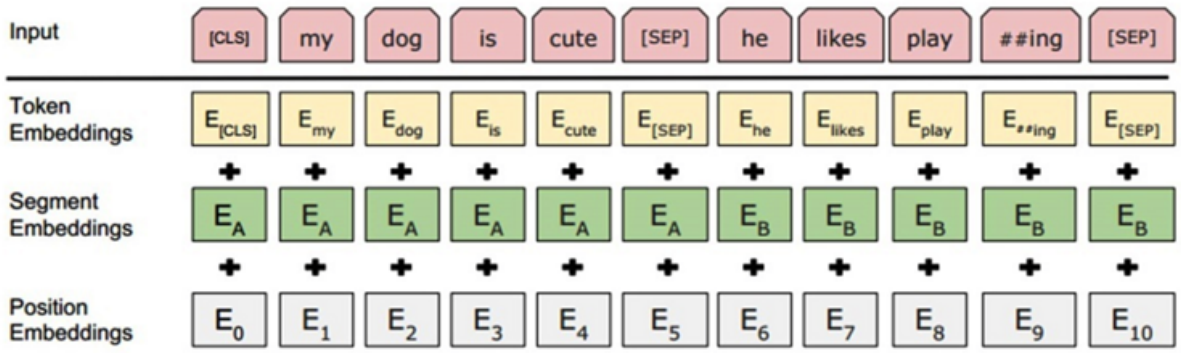

FIGURE 2.4 – Représentation d'entrée BERT

L'entrée est traitée de la manière suivante avant d'entrer dans le modèle :
— Un jeton [CLS] est inséré au début de la première phrase et un jeton [SEP] est inséré à la fin de chaque phrase.
— Incorporation de jetons : il s'agit des incorporations apprises pour le jeton spécifique à partir du vocabulaire de jeton WordPiece.
— Embeddings de segments : BERT peut également prendre des paires de phrases comme entrées pour les tâches (Question-Answering). C'est pourquoi il apprend une incorporation unique pour la première et la deuxième phrase afin d'aider le modèle à les distinguer. Dans l'exemple ci-dessus, tous les jetons marqués comme EA appartiennent à la phrase A (et de même pour EB).
— Embeddings de position : BERT apprend et utilise les incorporations de position pour exprimer la position des mots dans une phrase. Celles-ci sont ajoutées pour surmonter la limitation de Transformer qui, contrairement à un RNN, n'est pas capable de capturer des informations de « séquence » ou « d'ordre ».

2.4.3 Les tâches de pré-entraînement

BERT est pré-entraîné sur deux tâches NLP :

Modélisation de langage masqué(Masked Language Modeling)

Prédiction de la phrase suivante(Next Sentence Prediction)

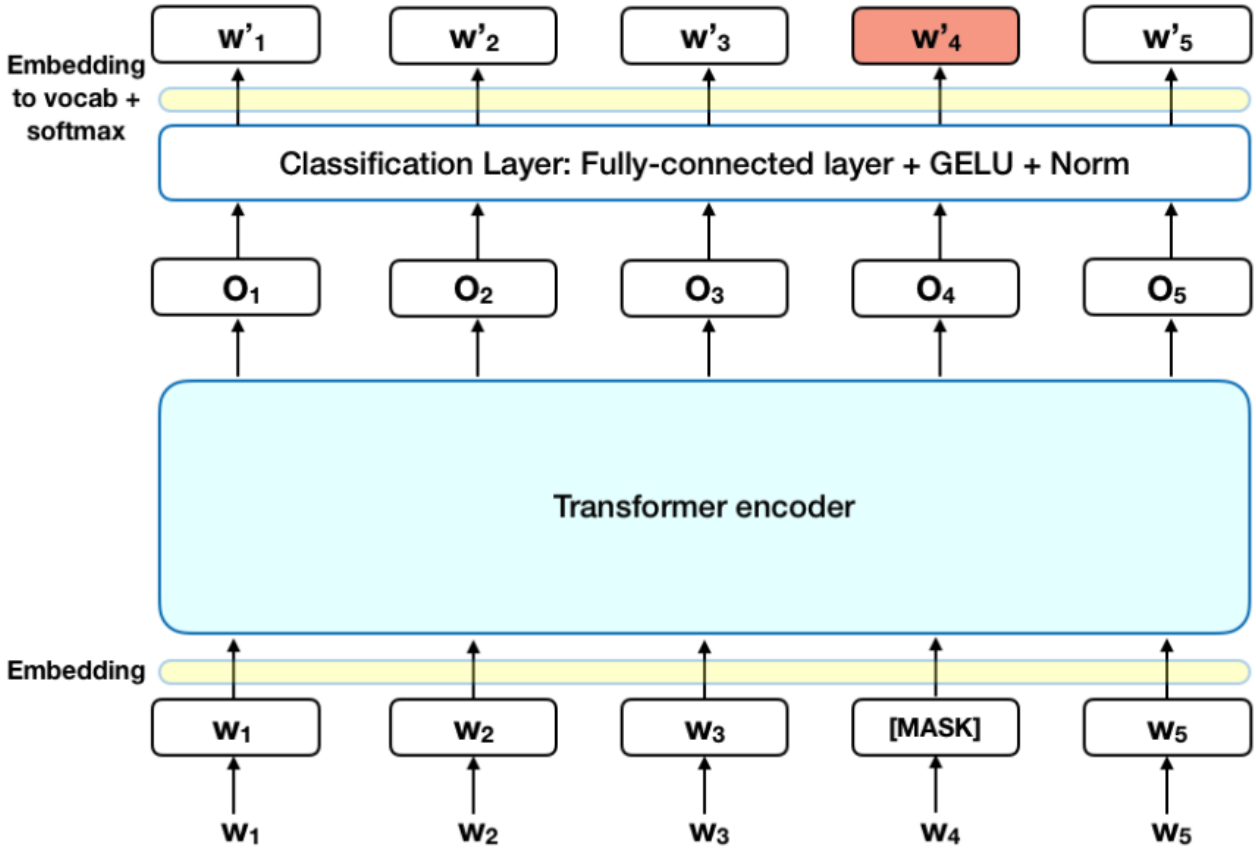

FIGURE 2.5 – Modélisation de langage masqué

Modélisation de langage masqué(Masked Language Modeling) : Avant d'alimenter des séquences de mots dans BERT, 15% des mots de chaque séquence sont remplacés par un jeton [MASK]. Le modèle tente ensuite de prédire la valeur d'origine des mots masqués, sur la base du contexte fourni par les autres mots non masqués de la séquence. En termes techniques, la prédiction des mots de sortie nécessite :

— Ajout d'une couche de classification au-dessus de la sortie de l'encodeur.
— Multiplier les vecteurs de sortie par la matrice d'incorporation, les transformer en dimension de vocabulaire.
— Calcul de la probabilité de chaque mot du vocabulaire avec softmax.

Bien que cela nous permette d'obtenir un modèle pré-entraîné bidirectionnel, un inconvénient est que nous créons un décalage entre le pré-apprentissage et le réglage fin, puisque le jeton [MASK] n'apparaît pas pendant le réglage fin. Pour atténuer cela, nous ne remplaçons pas toujours les mots « masqués » par le jeton [MASK] réel. Le générateur de données d'apprentissage choisit 15% des positions de jeton au hasard pour la prédiction. Si le i-ème jeton est choisi, on remplace le i-ème jeton par (1) le jeton [MASK] 80% du temps (2) un jeton aléatoire 10% du temps (3) le i-ème jeton inchangé 10% du temps. Ensuite, Ti sera utilisé pour prédire le jeton d'origine avec une perte d'entropie croisée.

Prédiction de la phrase suivante(Next Sentence Prediction)NSP : Dans le processus d'apprentissage BERT, le modèle reçoit des paires de phrases en entrée et apprend à prédire si la deuxième phrase de la paire est la phrase suivante dans le document original. Pendant la formation, 50% des entrées sont une paire dans laquelle la deuxième phrase est la phrase suivante dans le document original, tandis que dans les 50% restants, une phrase aléatoire du corpus est choisie comme deuxième phrase. L'hypothèse est que la phrase aléatoire sera déconnectée de la première phrase.
Pour prédire si la deuxième phrase est bien connectée à la première, les étapes suivantes sont effectuées :

— L'ensemble de la séquence d'entrée passe par le modèle Transformer.
— La sortie du jeton [CLS] est transformée en un vecteur en forme de 2×1, en utilisant une simple couche de classification (matrices apprises de poids et de biais).
— Calcul de la probabilité d'IsNextSequence avec softmax.

2.4.4 Réglage fin(Fine-tuning)

L'utilisation de BERT pour une tâche spécifique est relativement simple : BERT peut être utilisé pour une grande variété de tâches de langage, tout en ajoutant qu'une petite couche au modèle de base :

— Les tâches de classification telles que l'analyse des sentiments sont effectuées de la même manière que la classification de la phrase suivante, en ajoutant une couche de classification au-dessus de la sortie Transformer pour le jeton [CLS].

— Dans les tâches de réponse aux questions (par exemple, SQuAD v1.1), le logiciel reçoit une question concernant une séquence de texte et doit marquer la réponse dans la séquence. En utilisant BERT, un modèle de questions-réponses peut être formé en apprenant deux vecteurs supplémentaires qui marquent le début et la fin de la réponse.

— Dans la reconnaissance d'entités nommées (NER), le logiciel reçoit une séquence de texte et doit marquer les différents types d'entités (personne, organisation, date, etc.) qui apparaissent dans le texte. A l'aide de BERT, un modèle NER peut être formé en alimentant le vecteur de sortie de chaque jeton dans une couche de classification qui prédit l'étiquette NER.

Dans la formation de réglage fin, la plupart des hyper-paramètres restent les mêmes que dans la formation BERT, il reste donc un ensembles des hyper-paramètres qui nécessitent un réglage.

Chapitre 3

Concepts et implémentation du NLP

Un framework de chatbot a besoin d'une structure dans laquelle les intentions de conversation sont définies. Un moyen propre de le faire est d'utiliser un fichier JSON.

Nous allons créer un cadre de chatbot et construire un modèle conversationnel pour un domaine de soins de santé. Le chatbot doit répondre à des questions simples sur les symptômes, les maladies, etc.

Nous allons travailler en 3 étapes :

Nous allons transformer les définitions de l'intention conversationnelle en un modèle Pytorch.

Ensuite, nous allons construire un cadre de chatbot pour traiter les réponses.

Enfin, nous montrerons comment le contexte de base peut être incorporé dans notre processeur de réponses.

Nous utiliserons tflearn, une couche au-dessus de Pytorch, et bien sûr Python.

3.1 Créer des données d'entraînement

Nous avons créé des données d'entraînement dans un fichier Json (intents.json). Il a la structure suivante :

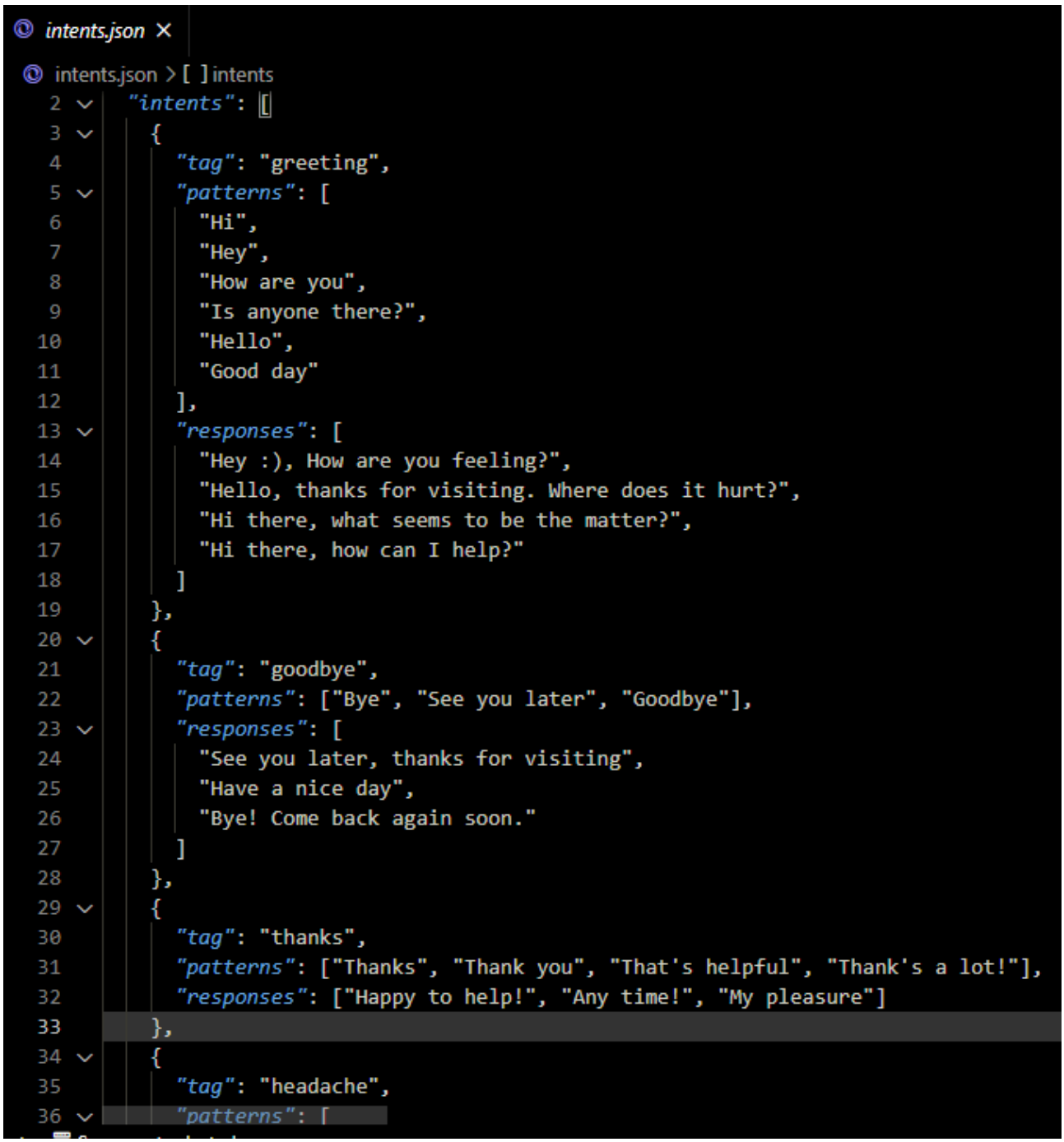

FIGURE 3.1 – Chatbot intents

3.2 Principes de base du NLP

Nous ne pouvons pas simplement transmettre la phrase d'entrée telle quelle à notre réseau neuronal. Nous devons d'une manière ou d'une autre convertir les chaînes de motifs en nombres que le réseau peut comprendre. Pour cela, nous convertissons chaque phrase en un bag of words (bow). Pour ce faire, nous devons collecter des mots d'entraînement, c'est-à-dire tous les mots que notre bot peut examiner dans les données d'entraînement. Sur la base de tous ces mots, nous pouvons ensuite calculer le sac de mots pour chaque nouvelle phrase. Un sac de mots a la même taille que le tableau de tous les mots, et chaque position contient un 1 si le mot est disponible dans la phrase entrante, ou 0 sinon. Voici un exemple visuel :

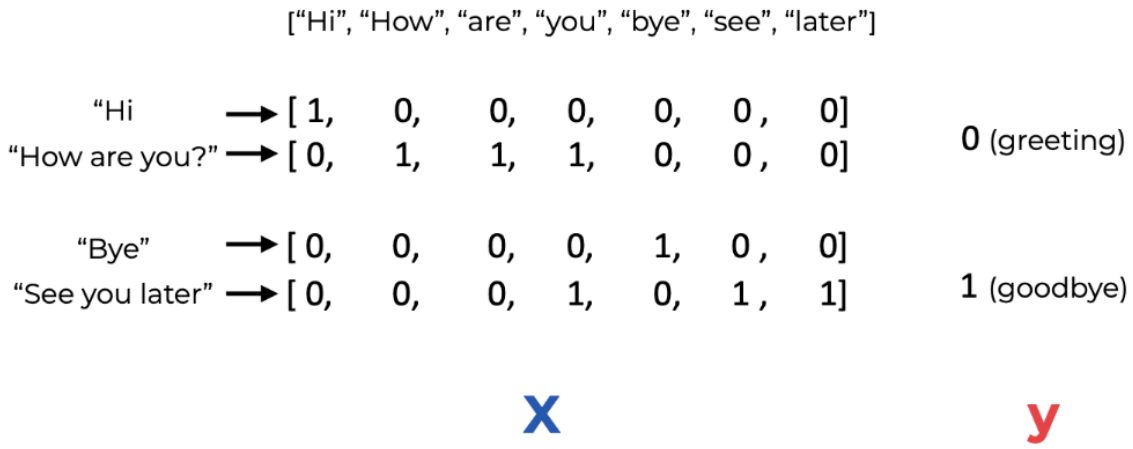

FIGURE 3.2 – Training data bag of words

Avant de pouvoir calculer l'arc, nous appliquons deux autres techniques NLP : la tokenisation et la séparation des mots.

— **Tokenisation :** Découpage d'une chaîne de caractères en unités significatives (par exemple, des mots, des caractères de ponctuation, des chiffres).

— **Stemming :** Un processus de normalisation linguistique, qui réduit les mots à leur racine ou supprime les affixes de dérivation. Par exemple, connexion, connecté, mot de connexion se réduisent à un mot commun "connecter".

Pour nos étiquettes, nous les trions par ordre alphabétique, puis nous utilisons l'index comme label de classe. L'ensemble de notre pipeline de traitement initial ressemble à ceci :

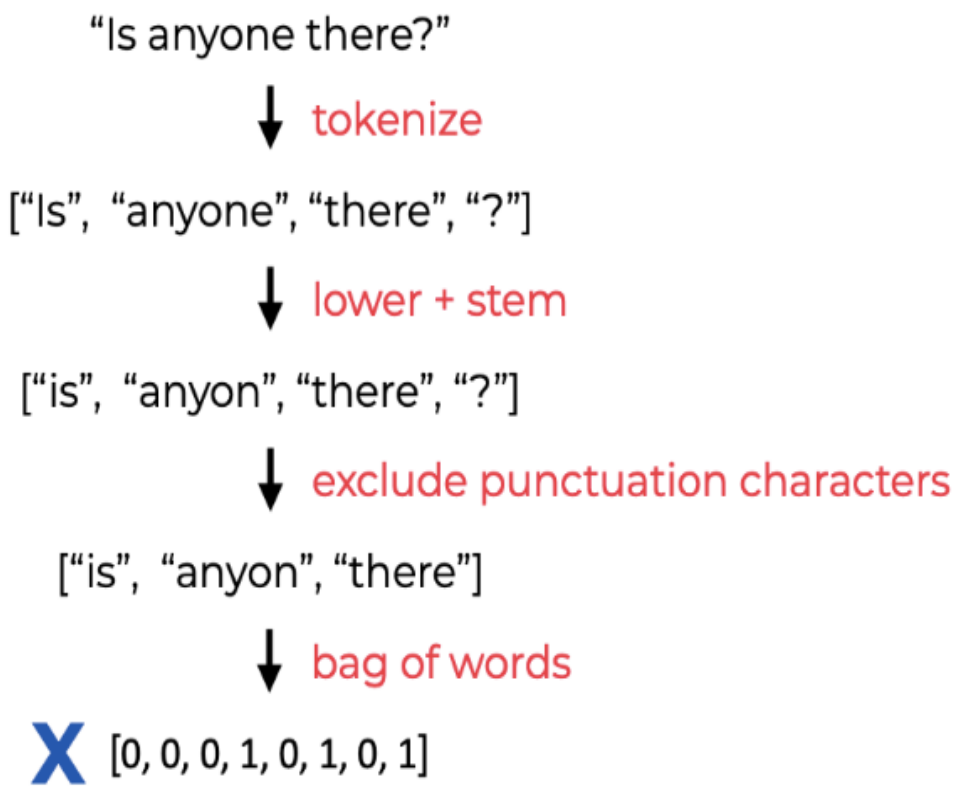

FIGURE 3.3 – NLP preprocessing pipeline

3.3 Implémenter les outils du NLP

Pour cela, nous utilisons le module nltk. NLTK (Natural Language Toolkit) est une plateforme de pointe pour la création de programmes Python destinés à travailler avec des données sur le langage humain.

C'est un package Python puissant qui fournit un ensemble d'algorithmes divers pour les langues naturelles. Il est gratuit, opensource, facile à utiliser, avec une grande communauté et bien documenté. NLTK comprend les algorithmes les plus courants tels que la tokenisation, l'étiquetage des parties du langage, la formation de la racine, l'analyse des sentiments, la segmentation des sujets et la reconnaissance des entités nommées. NLTK aide l'ordinateur à analyser, prétraiter et comprendre le texte écrit.

nltk_utils.py > ...

```python
import numpy as np
import nltk
from nltk.stem.porter import PorterStemmer

stemmer = PorterStemmer()

def tokenize(sentence):

    return nltk.word_tokenize(sentence)

def stem(word):

    return stemmer.stem(word.lower())

def bag_of_words(tokenized_sentence, words):

    # stem each word
    sentence_words = [stem(word) for word in tokenized_sentence]
    # initialize bag with 0 for each word
    bag = np.zeros(len(words), dtype=np.float32)
    for idx, w in enumerate(words):
        if w in sentence_words:
            bag[idx] = 1

    return bag

```

FIGURE 3.4 – Implémentation des outils NLP

3.4 Implémentation du Neural Network

L'implémentation est simple avec un réseau neuronal Feed Forward à deux couches cachées :

model.py > NeuralNet > forward

```python
import torch
import torch.nn as nn


class NeuralNet(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super(NeuralNet, self).__init__()
        self.l1 = nn.Linear(input_size, hidden_size)
        self.l2 = nn.Linear(hidden_size, hidden_size)
        self.l3 = nn.Linear(hidden_size, num_classes)
        self.relu = nn.ReLU()

    def forward(self, x):
        out = self.l1(x)
        out = self.relu(out)
        out = self.l2(out)
        out = self.relu(out)
        out = self.l3(out)
        # no activation and no softmax at the end
        return out
```

FIGURE 3.5 – Implémentation du Neural Network

3.5 Implémentation du Training Pipeline

Dans cette partie, on va expliquer le chargement du modèle formé et les prédictions pour les nouvelles phrases.

On commence par charger notre training data qui existe dans le fichier **intents.json**, puis on passe au traitement de chaque phrase du fichier, en appliquant les méthodes du fichier **nltk_utils.py** : Tokenizing et Stemming :

train.py > ...

```python

with open('intents.json', 'r') as f:
    intents = json.load(f)

all_words = []
tags = []
xy = []
# loop through each sentence in our intents patterns
for intent in intents['intents']:
    tag = intent['tag']
    # add to tag list
    tags.append(tag)
    for pattern in intent['patterns']:
        # tokenize each word in the sentence
        w = tokenize(pattern)
        # add to our words list
        all_words.extend(w)
        # add to xy pair
        xy.append((w, tag))

# stem and lower each word
ignore_words = ['?', '.', '!']
all_words = [stem(w) for w in all_words if w not in ignore_words]
# remove duplicates and sort
all_words = sorted(set(all_words))
tags = sorted(set(tags))

print(len(xy), "patterns")
print(len(tags), "tags:", tags)
print(len(all_words), "unique stemmed words:", all_words)
```

FIGURE 3.6 – Chargement et traitement du training data

Ensuit, on crée notre training data d'une manière à avoir les deux vecteurs X et Y, pour avoir à la fin notre Bag of Words list :

```python
# create training data
X_train = []
y_train = []
for (pattern_sentence, tag) in xy:
    # X: bag of words for each pattern_sentence
    bag = bag_of_words(pattern_sentence, all_words)
    X_train.append(bag)
    # y: PyTorch CrossEntropyLoss needs only class labels, not one-hot
    label = tags.index(tag)
    y_train.append(label)

X_train = np.array(X_train)
y_train = np.array(y_train)
```

FIGURE 3.7 – Création du bag of words

Enfin, on applique le training model au dataset :

```python
dataset = ChatDataset()
train_loader = DataLoader(dataset=dataset,
                          batch_size=batch_size,
                          shuffle=True,
                          num_workers=0)

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

model = NeuralNet(input_size, hidden_size, output_size).to(device)

# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)

# Train the model
for epoch in range(num_epochs):
    for (words, labels) in train_loader:
        words = words.to(device)
        labels = labels.to(dtype=torch.long).to(device)

        # Forward pass
        outputs = model(words)
        # if y would be one-hot, we must apply
        # labels = torch.max(labels, 1)[1]
        loss = criterion(outputs, labels)

        # Backward and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    if (epoch+1) % 100 == 0:
        print (f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}')
```

FIGURE 3.8 – Application du training model

Vers la fin du processus du trainig, on sauvegarde notre data résultante dans un fichier **data.pth**.

3.6 Implémentation du Chat

Dans cette partie, on va charger le modèle entraîné et faites des prédictions pour les nouvelles phrases.
On commence d'abord par récupérer les données du fichier **data.pth**.

La fonction chatbot_response prend comme paramètre la phrase entrée par l'utilisateur du chatbot et applique les outils du nltk, Tokenizing et Stemming. Elle calcule ensuite la probabilité d'avoir un élément existant parmi les tags de notre training data, et fait un test pour savoir si le tag du pattern entré par l'utilisateur correspond à un tag dans data.pth. Enfin elle renvois une réponse selon le résultat du test.

chat.py > ...

```python
def chatbot_response(sentence):

    #if sentence == "results":
        #res = listToString(what_are_your_symptoms(filter_disease(intent_tags_list)))

    sentence = tokenize(sentence)
    X = bag_of_words(sentence, all_words)
    X = X.reshape(1, X.shape[0])
    X = torch.from_numpy(X).to(device)

    print("\n *************")
    print(sentence)
    print("\n *************")

    output = model(X)
    _, predicted = torch.max(output, dim=1)

    tag = tags[predicted.item()]
    print(tag)

    probs = torch.softmax(output, dim=1)
    prob = probs[0][predicted.item()]
    if prob.item() > 0.75:
        for intent in intents['intents']:
            if tag == intent["tag"]:
                res = random.choice(intent['responses'])
                intent_tags_list.append(tag)
    elif "results" in sentence:
        dic= test.what_are_your_symptoms(filter_disease(intent_tags_list))
        res=dic
        print("\n ****/////***")
        print(dic)
        print("\n ***/////****")
    else:
```

FIGURE 3.9 – Génération des réponses du chatbot

Lorsque l'utilisateur envoie "results" comme phrase, il reçoit comme réponse le symptôme le plus proche aux description donnés. La fonction what_are_your_symptoms prend en paramètre la liste de symptômes entrés par l'utilisateur, fait un calcule de probabilité d'avoir une des maladies citées dans le fichier **disease.json** en ce basant sur les symptômes de chaque maladies, et retourne à la fin un dictionnaire qui contient une clé indiquant la maladie et sa valeur, et d'une autre clé qui indique la probabilité et sa valeur.

test.py > what_are_your_symptoms

```python
def what_are_your_symptoms(patient_symptoms):
    print(patient_symptoms)
    """patient_symptoms=[
      "pain chest",
      "shortness of breath",
      "dizziness",
    ]"""

    potential_diagnosis = []#[[desease,percent]]
    proba = []#these two are parrallel lists
    with open('disease.json','r') as f:
        diseases = json.load(f)
        for patient_symptom in patient_symptoms:
            for disease in diseases:
                if patient_symptom in diseases[disease]['symptoms']:
                    total_symp = len(diseases[disease]['symptoms'])
                    print("symptoms",diseases[disease]['symptoms'])
                    if diseases[disease]['disease'] in potential_diagnosis:
                        i = potential_diagnosis.index(diseases[disease]['disease'])
                        proba[i] += 1/total_symp
                    else:
                        potential_diagnosis.append(diseases[disease]['disease'])
                        proba.append(1/total_symp)

        if proba == []:
            j = 0
            potential_diagnosis.append(["you are maybe exausted",])
            mx = "?"
        else:
            mx = max(proba)
            mx = "{:.4f}".format(mx)
            j = proba.index(mx)
        #print(potential_diagnosis)
    return {"illness":potential_diagnosis[j][0],"probability":mx}
```

FIGURE 3.10 – Le résultat final du conseil

Chapitre 4

Réalisation finale du Chatbot

4.1 Front-end

Conclusion

Pour conclure, il suffit de dire que le but ultime des chatbots est d'imiter et de s'aligner autant que possible sur une conversation naturelle entre humains. En outre, lorsque nous concevons le flux conversationnel d'un chatbot, nous oublions souvent les éléments qui font partie intégrante d'une véritable conversation humaine.

La digression est une partie importante de la conversation humaine, avec la désambiguïsation bien sûr. La désambiguïsation annule dans une certaine mesure le danger de la prolifération de fallback où le dialogue n'avance pas vraiment.

Avec la désambiguïsation, un bouquet d'options réellement liées et contextuelles est présenté à l'utilisateur, parmi lesquelles il peut choisir, ce qui est sûr de faire avancer la conversation.

Enfin, la pire chose que vous puissiez faire est de présenter un ensemble d'options qui n'est pas lié au contexte actuel ou encore un ensemble d'options prédéfinies et finies qui se répètent constamment. La conscience contextuelle est essentielle dans tous les éléments d'un chatbot.

Bibliographie

[1] Pytorch documentation. <https://pytorch.org/docs/stable/index.html>.

[2] Jack Cahn. Chatbot : Architecture, design, development.

[3] NLTK 3.6.2 documentation. <https://www.nltk.org/>.

[4] gk. Contextual chatbots with tensorflow. <https://chatbotsmagazine.com/contextual-chat-bots-with-tensorflow-4391749d0077>.

[5] L. Guti´errez and B. (2019). Keith. A systematic literature review on word embeddings.

[6] Salma Jamoussi. Une nouvelle représentation vectoriellepour la classi[U+FB01]cation sémantique. <https://www.researchgate.net/publication/228790563_Une_nouvelle_representation_vectorielle_pour_la_classification_semantique>.

[7] Y. (2014) Kim. Convolutional neuralnetworks for sentence classification.

[8] Chen K. Corrado G. Mikolov, T. and J. (2013). Dean. Efficient estimation of word representations in vector space.

[9] E. G. Nal Kalchbrenner and P. (2014). Blunsom. A convolutional neural network for modelling sentences.

[10] B. C. W. (2016). Ye Zhang. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification.