

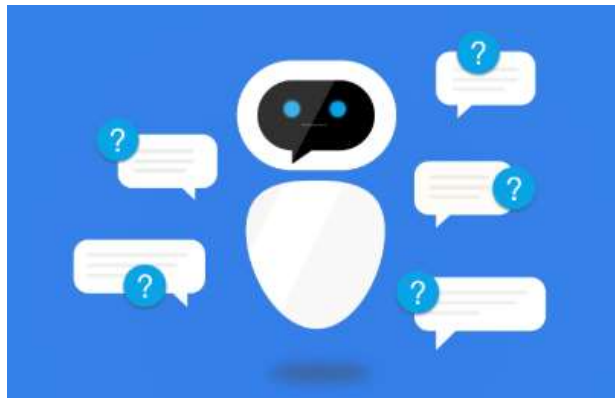


Stage de fin d'études Licence Professionnelle

Ingénierie des Systèmes Informatiques et logiciels

Thème :

**Agent conversationnel par apprentissage supervisé
et par renforcement**



Réalisé par : El mahdaouy Zahira

Encadré par : Pr. **Azidine Guezzaz**

2019/2020

Remerciement

En guise de reconnaissance, Je souhaite témoigner mes sincères remerciements à toutes les personnes qui ont contribué de près ou de loin au bon déroulement de l'élaboration de ce travail.

Mes sincères gratitudes à mon encadrent Pr. Azidine Guezzaz pour ses conseils et pour le suivi et l'intérêt qu'il avait pour mon travail.

Je tiens à remercier aussi, toute l'équipe pédagogique de l'école supérieure de technologie et les intervenants professionnels responsables de la formation Ingénierie des Systèmes Informatiques et logiciels, pour avoir assuré la partie théorique de celle-ci.

Table des matières

Remerciement	2
Table des matières.....	3
Introduction.....	5
1. Mots clé	7
2. Les réseaux de neurones	8
a. Introduction.....	8
b. Le fonctionnement de réseau de neurones	9
c. Types de réseau de neurones.....	9
d. Réseau de neurones récurrentes	10
e. Relation de RNN avec les réseaux de neurones à action directe	11
f. Fonctions d'activation	11
g. Problèmes récurrents.....	13
h. Les cellules à mémoire interne : LSTM (Long short time memory)	13
i. Modèle séquence à séquence (Sequence to Sequence Learning) :	14
3. PyTorch	16
a. Unité de PyTorch	16
b. Installation de PyTorch	16
c. La base de PyTorch : Tenseurs (Tensor)	17
4. Cahier des charges	18
5. Conception.....	18
a. Langage UML.....	18
b. Diagramme de cas d'utilisation	18
c. Diagramme de classe	19
d. Diagrammes de séquence.....	19

e.	Diagramme d'activités	20
1.	Etude de modèle existant.....	22
a.	Data set.....	22
b.	Encodeur	22
c.	Décodeur	23
d.	Phase d'apprentissage	23
e.	Phase d'évaluation	25
2.	Amélioration avec l'apprentissage par renforcement.....	25
a.	Simulation de dialogue entre deux agents	26
b.	Les fonctions de récompense	26
c.	Phase de test.....	28
	Conclusion	29
	Bibliographie.....	30

Introduction

La stratégie client est devenue le maître mot des entreprises qui cherchent sans cesse à améliorer et humaniser le relationnel client sur leur site internet. C'est pour cela que, depuis quelques années, des nouveaux agents proposant de l'aide aux clients connectés apparaissent sur les sites. En réalité, ce sont des agents conversationnels autrement appelés Chatbot, le robot simule une conversation amicale avec le client tout en tentant d'apporter des réponses pré-enregistrées à ses questions. Ce système a déjà atteint ses limites et beaucoup d'agents conversationnels sont disparus. Lors des questions trop précises les agents conversationnels ont une tendance à être hors-sujet. L'atout majeur dans cette course reste l'innovation technologique. Il est donc nécessaire de perfectionner le fonctionnement du Chatbot en détectant si la question dépasse leur champ d'action et agir en conséquence. C'est ce que je réalise à travers ce projet, construire un prototype d'agent conversationnel avec un rôle bien défini et montrer sa capacité à distinguer plusieurs types de questions on se basant sur l'apprentissage supervisé et l'apprentissage renforcement.

I. L'historique des agents conversationnels

Les premiers agents conversationnels fonctionnaient grâce à une base de données avec un système simple de question/réponse. Par la suite, les chatbot sont évolués au rythme des avancées de l'intelligence artificielle (IA). D'après le Larousse, l'IA est définie comme un ensemble de théories et de techniques mises en œuvre en vue de réaliser des machines capables de simuler l'intelligence humaine. Le premier chatbot de l'Histoire est Eliza, un programme informatique inventé par Joseph Weizenbaum en 1966 qui simulait un psychiatre. Il fonctionnait par simple reconnaissance de forme et par reformulation des affirmations en question grâce à un système de mots-clés. Ce programme constituait déjà à l'époque de nombreuses limites mais il est ce qu'on pourrait appeler aujourd'hui le géniteur des chatbot actuels qui envahissent notre quotidien. Ces innovations à l'époque étaient évaluées grâce au célèbre test de Turing, toujours d'actualité, aussi appelé « le jeu de l'imitation ». Il a été imaginé par Alan Turing, l'un des pionniers de l'IA. Il consiste à faire se confronter verbalement un humain A à l'aveugle avec un autre humain B d'un côté et l'IA que l'on veut tester de l'autre. Le concept est simple : l'humain A pose des questions à l'humain B et à l'IA et doit ensuite déterminer lequel est une machine. Si l'humain A n'est pas capable de cibler qui est l'ordinateur, alors cela signifie que le logiciel a réussi à avoir une apparence sémantique humaine.

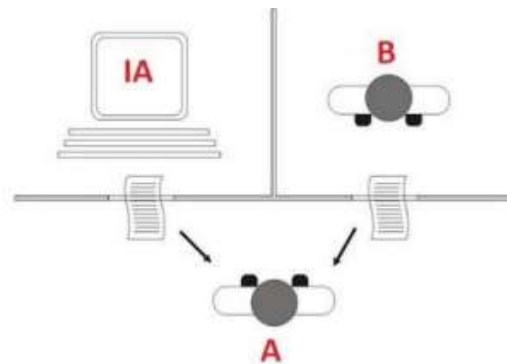


Figure 1: le jeu de l'imitation

Aujourd'hui, les chatbot sont évolués et sont devenus beaucoup plus performants grâce au « machine learning » (ou apprentissage automatique). Ce système permet de fournir une analyse prédictive du langage humain capable de s'adapter grâce à des données. Thomas Sabatier, CEO de « The Chatbot Factory » qui se positionne comme l'un des leaders dans la conception d'agents conversationnels "non -linéaires" explique que ces chatbot répondent aux attentes de la demande

d'hyperpersonnalisation. Ses outils sont définis comme « non-linéaires » car ils sont dotés d'intelligence artificielle, notamment dans la compréhension du langage naturel. L'IA, associée au machine learning, a donné lieu à de nouveaux usages. Grâce à l'hyperpersonnalisation du message, les nouveaux agents conversationnels ont impacté la relation client et depuis les marques ont l'obligation de s'adapter pour rester compétitives. Ainsi, au cours des années 2010, les géants du web Apple, Microsoft et Google ont développé leurs propres assistants conversationnels.



Figure 2: Exemples des agents conversationnels

II. Méthodes et technologies utilisées

Pour la réalisation de mon projet je vais utiliser le package du langage de programmation python « PyTorch » qui implémente plusieurs notions de machine learning afin de faciliter la conversation entre un utilisateur et l'agent conversationnel : l'agent doit répondre dans un laps de temps, doit contrôler ses actions, et il ne doit pas attendre qu'une tierce personne lui donne l'autorisation de réagir. Dans cette partie je vais discuter les technologies de machine learning qui me permet de créer un agent conversationnel avec ces caractéristiques par l'apprentissage supervisé et l'apprentissage par renforcement, et je vais mentionner aussi les outils que je vais utiliser pour chaque partie de réalisation.

1. Mots clé

L'apprentissage supervisé : on se base sur des données annotées par un ou plusieurs humains pour pouvoir l'imiter par la suite. Prenons par exemple des images de chats et des images de chiens. Pendant l'apprentissage nous allons superviser le modèle de machine learning en lui montrant des images de chats en lui disant que ce sont des chats puis des images de chiens en lui disant que ce sont des chiens. L'objectif est d'entraîner le modèle à reconnaître et interpréter une image inédite et de pouvoir distinguer s'il s'agit d'une image de chien ou d'une image de chat.

La contrainte principale de cette méthode est qu'elle implique l'intervention d'un humain pour annoter les données avant que l'entraînement du modèle de machine learning soit réalisé.

L'Apprentissage par Renforcement, ou *Reinforcement Learning* en anglais, (RL) est une forme de Machine Learning qui vise à construire des agents capables de prendre des décisions autonomes dans des environnements complexes ou même aléatoires. Les décisions que prend un agent modifient son environnement qui lui transmet un signal de récompense lui indiquant en retour si l'action était bonne ou mauvaise au regard d'un objectif à atteindre.

Les deux formes d'apprentissage, supervisé et par renforcement, peuvent naturellement être panachés. Pour éviter les problèmes de **cold start** par exemple, on peut initialiser un agent conversationnel avec un apprentissage supervisé basé sur un corpus conversation, puis à mesure que s'accumulent les échanges avec de vrais utilisateurs, on pourra progressivement enclencher un apprentissage par renforcement qui affinera le comportement de l'agent.

2. Les réseaux de neurones

a. Introduction

Le réseau de neurones est une méthode de construction de modèle qui peut, par le biais de données d'entraînement, "apprendre" à prédire des valeurs par la suite. On peut s'en servir aussi bien pour analyser des données simples (données chimique, choix d'utilisateur, capteurs de température...) que des données très complexes comme des images, des vidéos, le cours de la bourse, etc...

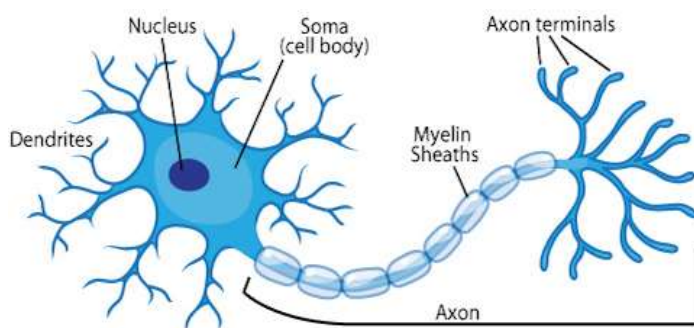


Figure 3: Anatomie d'un neurone biologique

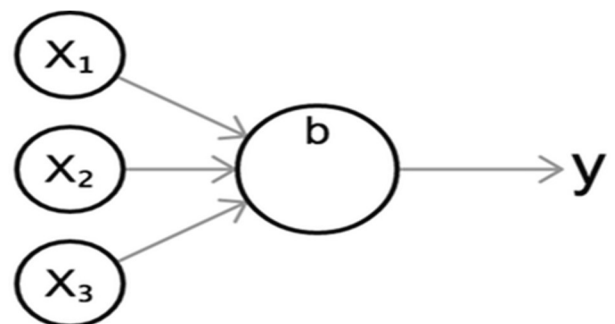


Figure 4: Un neurone artificiel

Réseau de neurones artificiels : est un système informatique s'inspirant du fonctionnement du cerveau humain pour apprendre.

b. Le fonctionnement de réseau de neurones

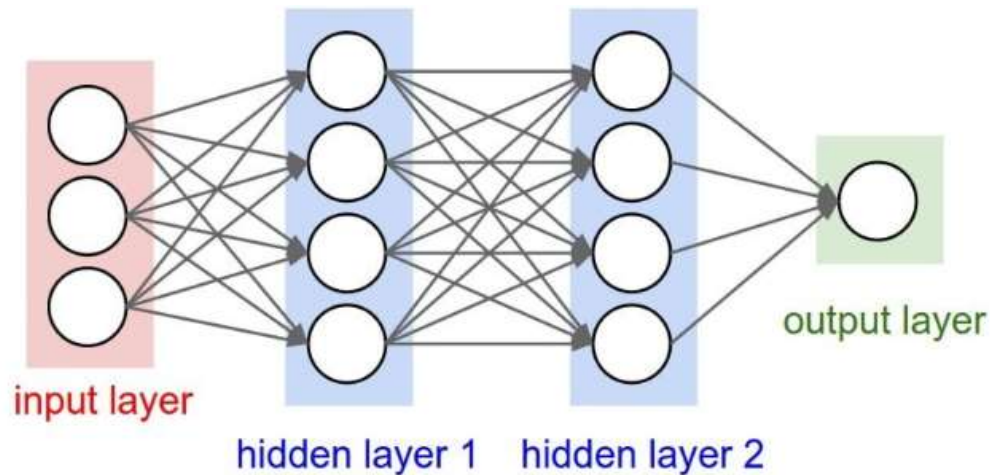


Figure 3: Le fonctionnement de réseau de neurones

En règle générale, un réseau de neurones repose sur un grand nombre de processeurs opérant en parallèle et organisés en tiers. Le premier tiers reçoit les entrées d'informations brutes, un peu comme les nerfs optiques de l'être humain lorsqu'il traite des signaux visuels. Par la suite, chaque tiers reçoit les sorties d'informations du tiers précédent. On retrouve le même processus chez l'Homme, lorsque les neurones reçoivent des signaux en provenance des neurones proches du nerf optique. Le dernier tiers, quant à lui, produit les résultats du système.

Par le biais d'un algorithme, le réseau de neurones artificiels permet à l'ordinateur **d'apprendre à partir de nouvelles données**. L'ordinateur doté du réseau de neurones apprend à effectuer une tâche en analysant des exemples pour s'entraîner.

c. Types de réseau de neurones

On distingue **différents types de réseaux de neurones**. En règle générale, les Neural Networks sont catégorisés en fonction du nombre d'épaisseurs qui séparent l'entrée de données de la

production du résultat, en fonction du nombre de nœuds cachés du modèle, ou encore du nombre d'entrées et de sorties de chaque nœud.

- **Réseaux neuronaux à action directe** : les informations passent directement de l'entrée aux nœuds de traitement puis aux sorties.
- **Réseau de neurones récurrents**, quant à eux, sauvegardent les résultats produits par les nœuds de traitement et nourrissent le modèle à l'aide de ces résultats. Ce mode d'apprentissage est un peu plus complexe. Ils ont une mémoire qui capture les informations sur ce qui a été calculé jusqu'à présent.
- **Réseaux de neurones convolutifs** sont de plus en plus utilisés dans différents domaines : numérisation de texte, traitement naturel du langage...

Dans la réalisation de ce projet je vais utiliser le réseau de neurones récurrents parce qu'il m'offre une prédiction basée sur les données précédentes, pas seulement les données actuelles. Ils utilisent des informations séquentielles.

d. Réseau de neurones récurrentes

Les connexions récurrentes permettent au réseau de neurones à un instant t de "**voir**" la fenêtre correspondante à cet instant, mais aussi de se "**souvenir**" de sa décision à un instant précédent (figure numéro 4).

Si l'on reprend l'exemple de la reconnaissance d'écriture, le réseau reconnaît désormais des caractères en se "**souvenant**" du caractère reconnu à l'instant précédent :

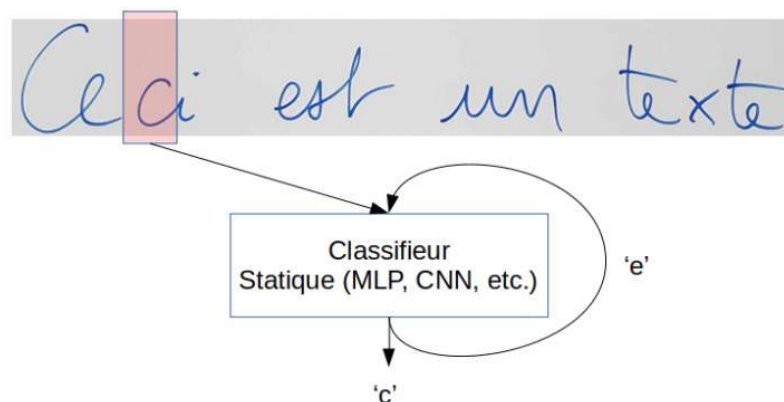


Figure 4: la reconnaissance d'écriture

Le réseau récurrent parcourt le signal gauche à droite à l'aide d'une fenêtre glissante. Il décide du caractère à reconnaître, en se basant sur le contenu de la fenêtre et sur la décision sur la fenêtre précédente

Un RRN est une copie multiple du même réseau qui reçoit des entrées à des moments différents ainsi que son état caché précédent. Nous pouvons le défiler pour la visualisation

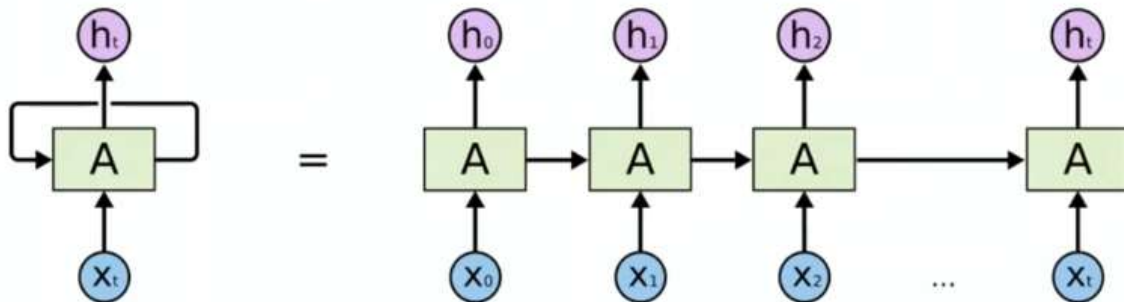


Figure 5: Réseau de neurones récurrentes

e. Relation de RNN avec les réseaux de neurones à action directe

Chacune des copies qui construit le réseau de neurones récurrentes est un réseau de neurone normal

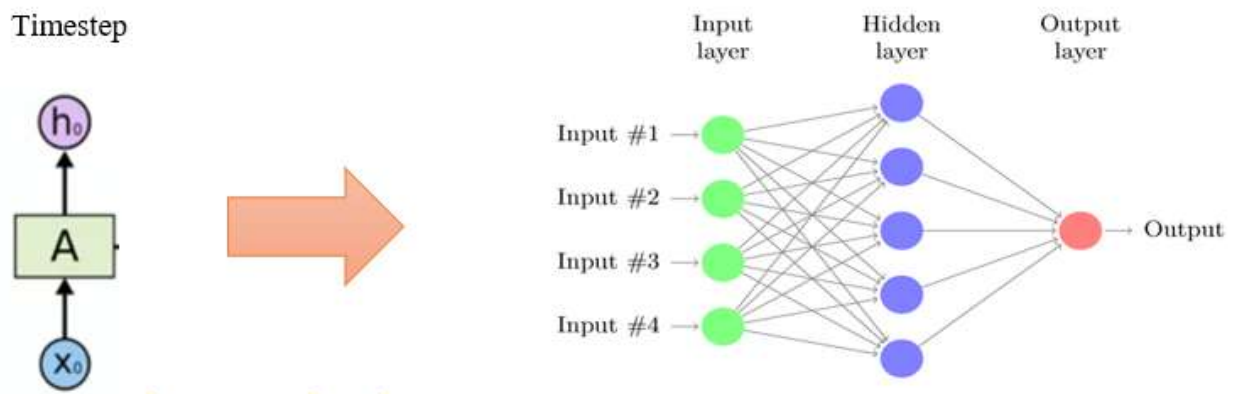


Figure 6: Relation de RNN avec les réseaux de neurones normaux

f. Fonctions d'activation

Une fonction d'activation est une fonction mathématique appliquée à un signal en sortie d'un neurone artificiel afin d'aider le réseau à apprendre des schémas complexes dans les données. Il

prend le signal de sortie de la cellule précédente et le convertit en une forme qui peut être prise en entrée dans la cellule suivante.

Les valeurs entrantes dans un neurone ($x_1, x_2, x_3, \dots, x_n$) sont multipliées avec leur poids (référence au poids synaptique) qui leur sont associés ($w_1, w_2, w_3, \dots, w_n$). On fait ensuite la somme de ces multiplications et on ajoute enfin le biais (référence au seuil).

$$z(x) = \sum_i^N (w_i x_i) + b$$

La fonction $Z(x)$ correspond à la pré-activation, c'est à dire l'étape qui précède l'activation. Ensuite, la fonction d'activation intervient. Le résultat z de la fonction de pré-activation $Z(x)$ est interprété par une fonction d'activation $A(z)$ produisant en sortie un résultat y .

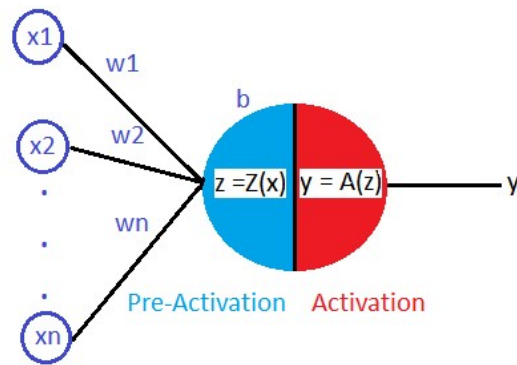


Figure 7: fonctionnement des fonctions d'activation

L'activation a pour but de transformer le signal de manière à obtenir une valeur de sortie à partir de transformations complexes entre les entrées. Pour ce faire la fonction d'activation doit être non linéaire.

L'utilisation de fonctions d'activation non linéaires est tout simplement indispensable pour la raison que les fonctions linéaires ne fonctionnent qu'avec une seule couche de neurone. Car au-delà d'une couche de neurones, Les données traitées par les neurones peuvent atteindre des valeurs étonnamment grandes. L'utilisation d'une fonction linéaire, ne modifiant pas la sortie, les valeurs des données transmises de neurones en neurones peuvent devenir de plus en plus grandes et rendant les calculs beaucoup plus complexes. Afin d'y remédier, les fonctions d'activation non linéaires réduisent la valeur de sortie d'un neurone le plus souvent sous

forme d'une simple probabilité. Etudions maintenant les fonctions d'activation les plus utilisées...

g. Problèmes récurrents

Les fonctions standards amènent au réseau la disparition ou l'explosion de gradient, et donc une saturation et entraîne un ralentissement de la back propagation dans les couches basses du réseau. Voici le problème que nous pouvons rencontrer.

Problème de disparition de gradient : L'algorithme progresse vers les couches inférieures du réseau, rendant les gradients de plus en plus petits. La mise à jour donc par descente de gradient ne modifie que très peu les poids des connexions de la couche inférieure, empêchant une bonne convergence de l'entraînement vers la solution.

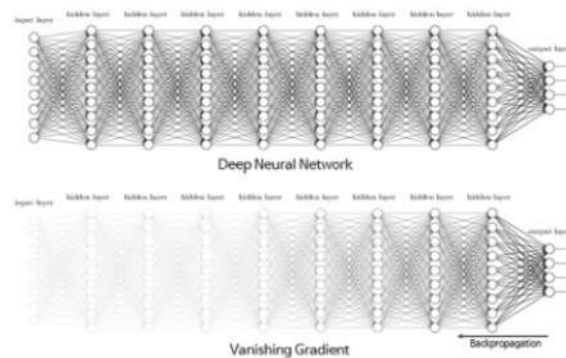


Figure 8: Problème de disparition de gradient

Pour éviter ce problème on va utiliser ce qu'on appelle Les cellules à mémoire interne.

h. Les cellules à mémoire interne : LSTM (Long short time memory)

LSTM possèdent une mémoire interne appelée *cellule* (ou *cell*). La cellule permet de maintenir un état aussi longtemps que nécessaire. Cette cellule consiste en une valeur numérique que le réseau peut piloter en fonction des situations. Il est utilisé pour savoir ce qui s'est passé précédemment afin de prédire une action à l'instant t est très important. Une couche LSTM peut être illustré de la manière suivante :

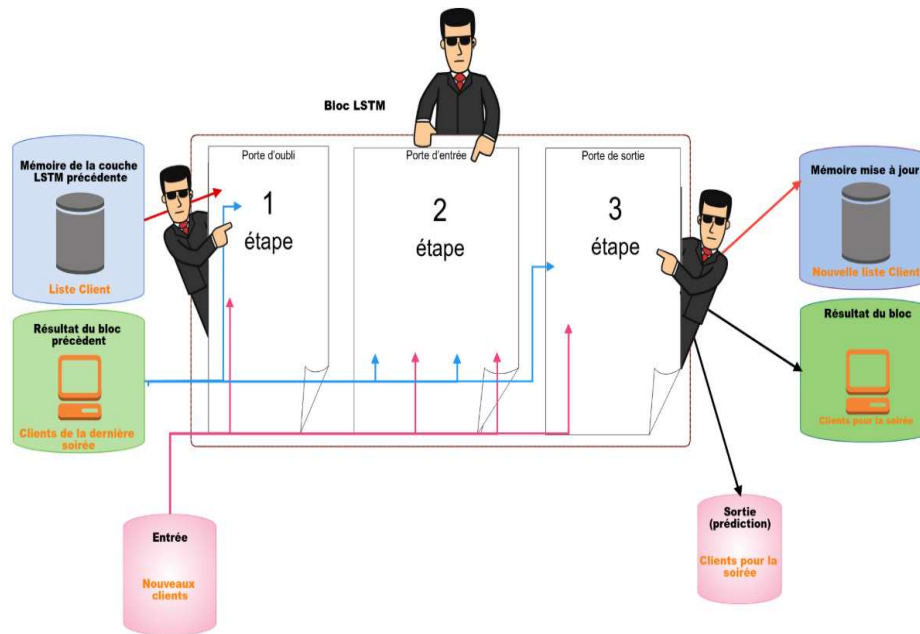


Figure 9: Structure du réseaux LSTM

Comme on peut le constater sur le schéma, la cellule mémoire peut être pilotée par trois portes de contrôle qu'on peut voir comme des vannes :

- **La porte d'oubli** va se charger de filtrer les informations contenues dans la cellule mémoire précédente. Parmi ces informations, certaines vont être plus pertinentes que d'autre.
- **La porte d'entrée** va se charger de décider quelles nouvelles valeurs entrantes du LSTM) vont être autorisées à être stockées dans la cellule mémoire.
- **La porte de sortie** décide si le contenu de la cellule doit influencer sur la sortie du neurone.

Les équations régissant les trois portes de contrôle sont l'application de la somme pondérée des entrées, des sorties et de la cellule, par des poids ω_{ab} spécifiques à chaque connexion entre les signaux a et b, suivie de l'application d'une fonction d'activation, typiquement la sigmoïde.

i. **Modèle séquence à séquence (Sequence to Sequence Learning) :**

Réseau de neurones récurrent qui convertit une séquence de données d'un domaine en entrée vers une nouvelle séquence de données dans un autre domaine en sortie.

Généralement, un modèle séquence à séquence est implémenté en utilisant deux réseaux de neurones récurrents, un premier réseau est un encodeur et le second est un décodeur. On parle aussi d'une architecture encodeur-décodeur. Dans ces modèles, l'entrée et la sortie ne sont pas nécessairement de la même longueur. Un bon exemple d'utilisation d'un modèle séquence à séquence est la traduction neuronale d'une phrase d'une langue d'origine vers une langue d'arrivée.

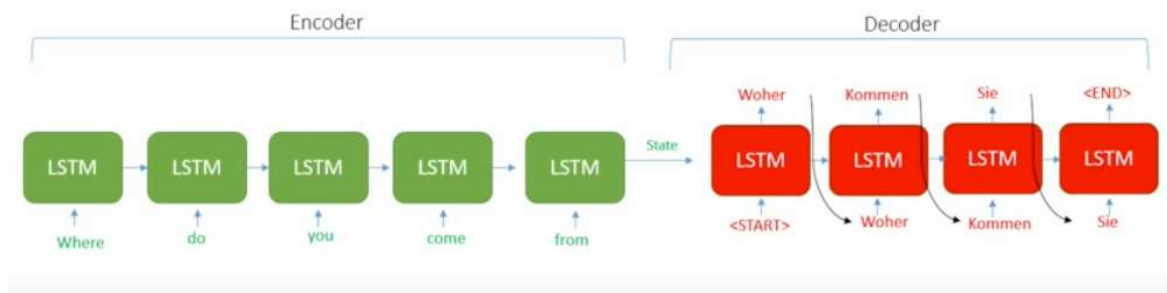


Figure 10: Exemple de la traduction d'une phrase de l'anglais vers l'allemand

Dans cet exemple l'input est la phrase « where do you come from » et l'output est sa traduction en allemand « woher kommen sie ». L'encodeur passe une représentation de la phrase (state : contiennent en principe le sens complet de la phrase à traduire) au décodeur via la dernière LSTM. La première LSTM de décodeur reçoit le state et un Start Token et génère le premier mot « woher », et la deuxième LSTM reçoit ce mot et l'état de LSTM précédente comme input, ainsi de suite jusqu'à la fin. Lors d'une traduction d'une nouvelle phrase chaque mot fourni en entrée du décodeur à l'instant t correspond au mot trouvé en sortie à l'instant précédent $t-1$.

i. Mécanisme d'attention dans un modèle SeqToSeq

Un mécanisme d'attention permet à un réseau de neurones récurrentes de porter son attention sur différentes parties de son entrée $\{x_0, \dots, x_i\}$.

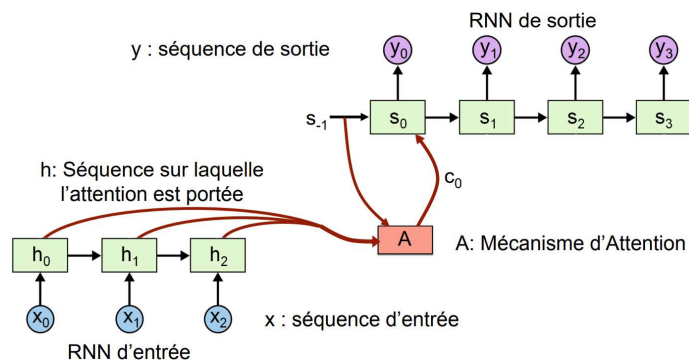


Figure 11: Illustration d'un RNN avec mécanisme d'attention

Cette architecture se base sur un réseau de neurones récurrentes comme encodeur. Ce RNN calcul une annotation h_i pour chaque mot x_i de la séquence d'entrée $\{x_0, \dots, x_i\}$. Chaque annotation contient le résumé d'à la fois les mots précédents et les mots suivants.

Cette séquence d'annotations $\{h_0, \dots, h_i\}$ sera utilisée par le décodeur pour calculer un vecteur de contexte c_t . Un vecteur de contexte est recalculé après chaque émission d'une étiquette en sortie.

Le mécanisme d'attention A génère l'entrée suivante du décodeur à partir de l'état $s-1$ et de toute la séquence h sur laquelle il porte son attention.

3. PyTorch

PyTorch (Janvier 2016) est une bibliothèque open source permettant aux développeurs de créer des modèles de machine learning et plus particulièrement de deep learning. Elle est développée par les membres du laboratoire en intelligence artificielle de Facebook. Il permet d'effectuer les calculs tensoriels nécessaires pour l'apprentissage profond. Ces calculs sont optimisés et effectués soit par le processeur (CPU) soit par un processeur graphique (GPU) supportant CUDA.

a. Unité de PyTorch

Le framework est conçu et assemblé pour fonctionner avec Python au lieu de se pousser souvent contre lui. Les modèles et les couches sont simplement des classes Python, tout comme tout le reste : optimiseurs, chargeurs de données, fonctions de perte, transformations, etc.

PyTorch a également l'avantage d'une API stable qui n'a subi qu'un seul changement majeur depuis les premières versions vers la version 1.3 (c'est-à-dire le changement des variables en tenseurs). Bien que cela soit sans aucun doute dû à son jeune âge, cela signifie que la grande majorité du code PyTorch est reconnaissable et compréhensible, quelle que soit la version pour laquelle il a été écrit.

b. Installation de PyTorch

Je travaille sur Anaconda Python 3.7.4. L'installation de la librairie ne pose absolument aucune difficulté avec le gestionnaire de packages « conda ». La commande à utiliser est indiquée sur le site de PyTorch en fonction de notre configuration.


PyTorch Build	Stable (1.4)		Preview (Nightly)	
Your OS	Linux	Mac	Windows	
Package	Conda	Pip	LibTorch	Source
Language	Python		C++ / Java	
CUDA	9.2	10.1	None	
Run this Command:	<pre>conda install pytorch torchvision cpuonly -c pytorch</pre> 			

Figure 12: Commande à utiliser pour installer la librairie (Anaconda - Conda)

C. La base de PyTorch : Tenseurs (Tensor)

Tenseur est un type de données spécialement conçu afin de déclarer des tableaux multidimensionnels.

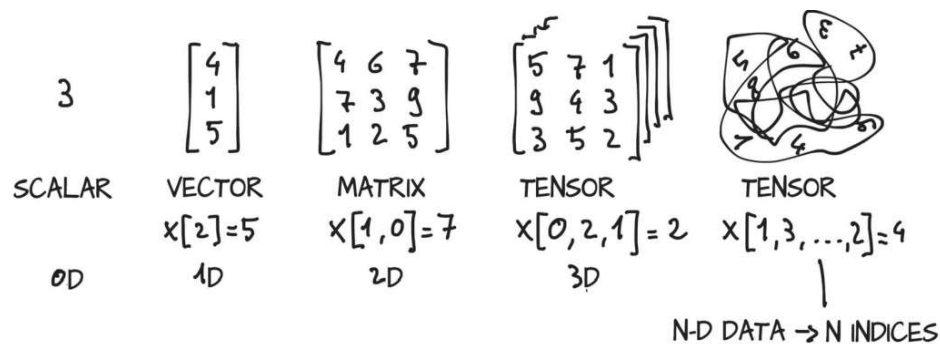


Figure 13: Les tenseurs sont les blocs de construction pour représenter les données dans PyTorch

```
Tensor0= torch.empty(28, 28) #Créer un Tensor Vide
Tensor1 = torch.rand(28, 28) #Créer un Tensor initialisé aléatoirement
Tensor2 = torch.rand(28, 28)
tensor3 = torch.add(tensor1, tensor2) #méthode 1 pour additionner
tensor3 = tensor1+tensor2 #méthode 2 plus simple
```

III. Description et analyse des besoins

La réalisation d'un système doit principalement passer d'une phase importante et indispensable qui est l'analyse des besoins et la conception. Cette phase a pour objectif d'expliquer le déroulement de notre système, de lister les résultats attendus en termes de fonctionnalités et

d'assurer une bonne compréhension des besoins des utilisateurs. Avant de se pencher sur la conception de système, La première étape sera la présentation d'un cahier des charges afin de spécifier le besoin : que va faire le bot ? Quelle est son intérêt ? La seconde étape sera la conception du système tout en respectant le cahier des charges.

4. Cahier des charges

Le cahier des charges décrit précisément les opérations à prendre en charge. Voici les fonctionnalités attendues de ce projet :

- Création d'un chatbot intelligent avec l'apprentissage par renforcement et qui se base sur un système existant qui implémente l'apprentissage supervisé
- L'agent doit contrôler ses actions, il ne doit pas attendre qu'une tierce personne lui donne l'autorisation de réagir
- Assurer la capacité d'apprendre au fil des conversations avec leur interlocuteur et de mémoriser les informations.
- L'agent doit répondre dans un court laps de temps
- Montrer la capacité de Chatbot à distinguer plusieurs types de questions.
- La conversation doit rester fluide et que l'utilisateur pense parler à une personne réelle

5. Conception

Après avoir défini les problèmes et précisé les besoins auxquels le projet doit répondre, nous arrivons à la phase de conception.

Nous avons utilisé la modélisation objet avec le langage UML.

a. Langage UML

UML est l'abréviation de Unified Modeling Language, c'est-à-dire langage unifié pour la modélisation. C'est un ensemble d'outils permettant la modélisation de la future application informatique.

b. Diagramme de cas d'utilisation

Les diagrammes de cas d'utilisation sont des diagrammes utilisés pour donner une vision globale du comportement fonctionnel d'un système logiciel. Ils sont utiles pour des présentations auprès de la direction ou des acteurs d'un projet.

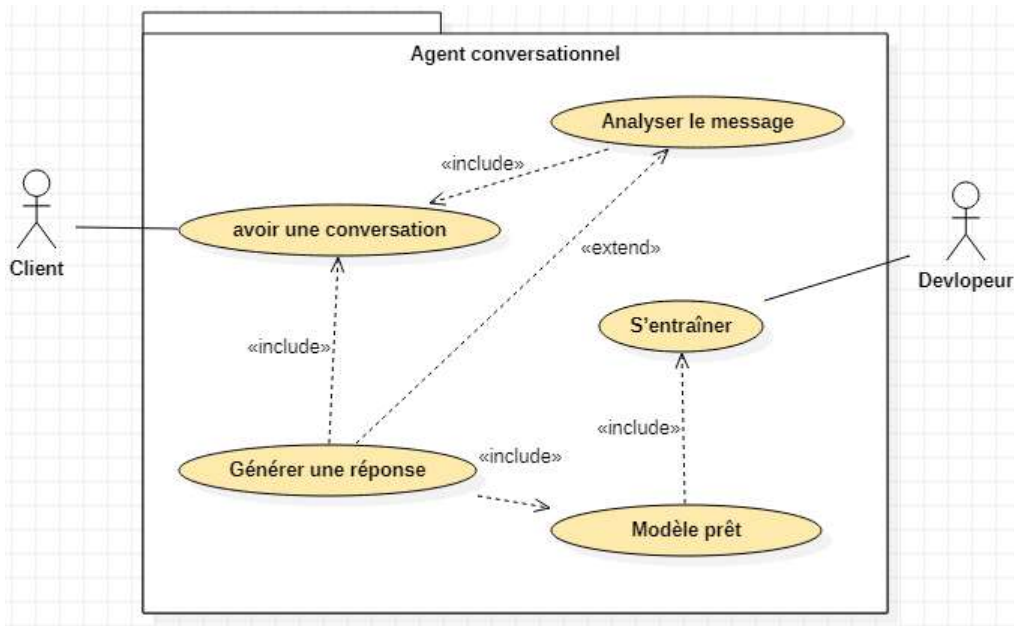


Figure 14: Diagramme de cas d'utilisation

c. Diagramme de classe

Les diagrammes de classes décrivent clairement la structure d'un système particulier en modélisant ses classes, ses attributs, ses opérations et les relations entre ses objets.

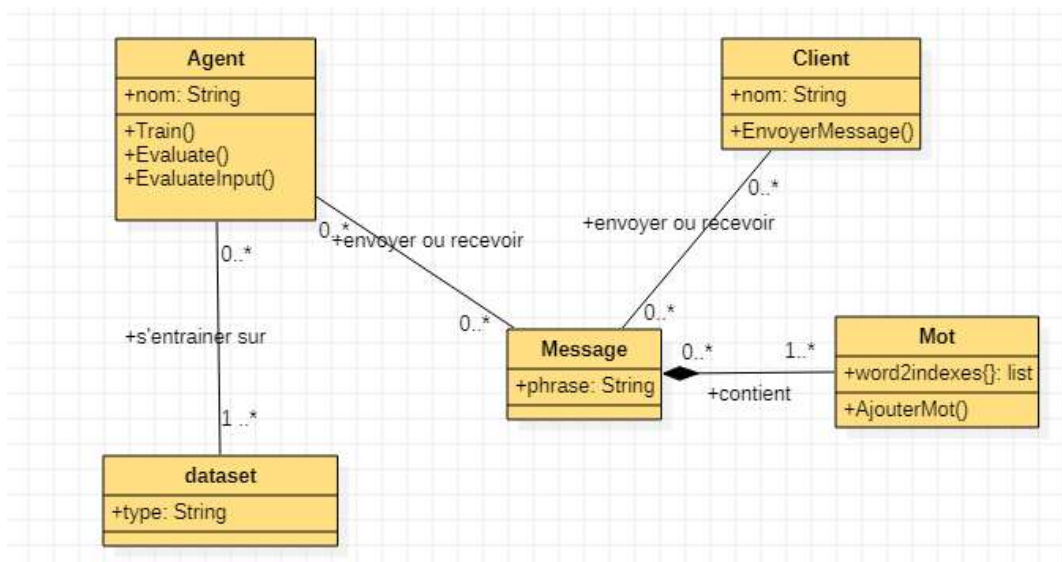


Figure 15: Diagramme de classe présentant l'ensemble des classes du projet

d. Diagrammes de séquence

Le diagramme de séquence permet de montrer les interactions d'objets dans le cadre d'un scénario prédéfini.

i. Conversation

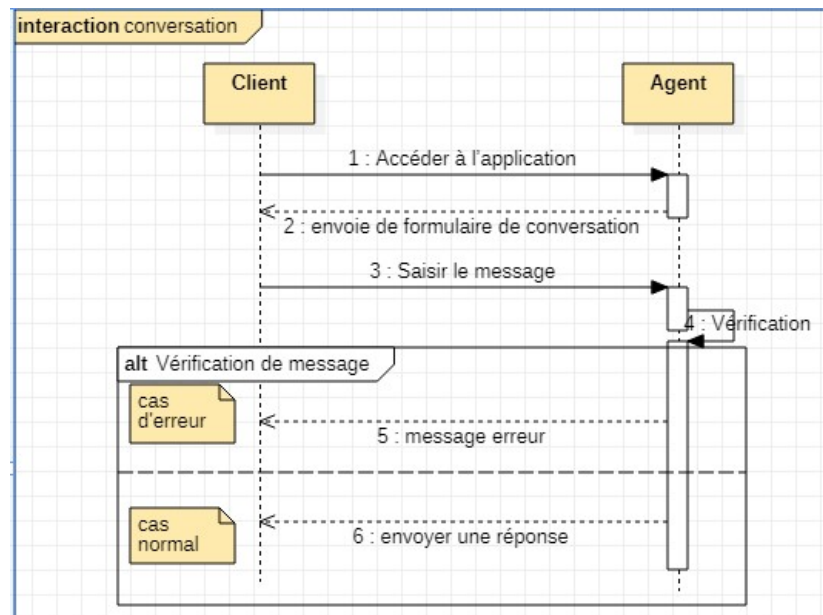


Figure 16: Diagrammes de séquence de conversation

ii. Entraînement

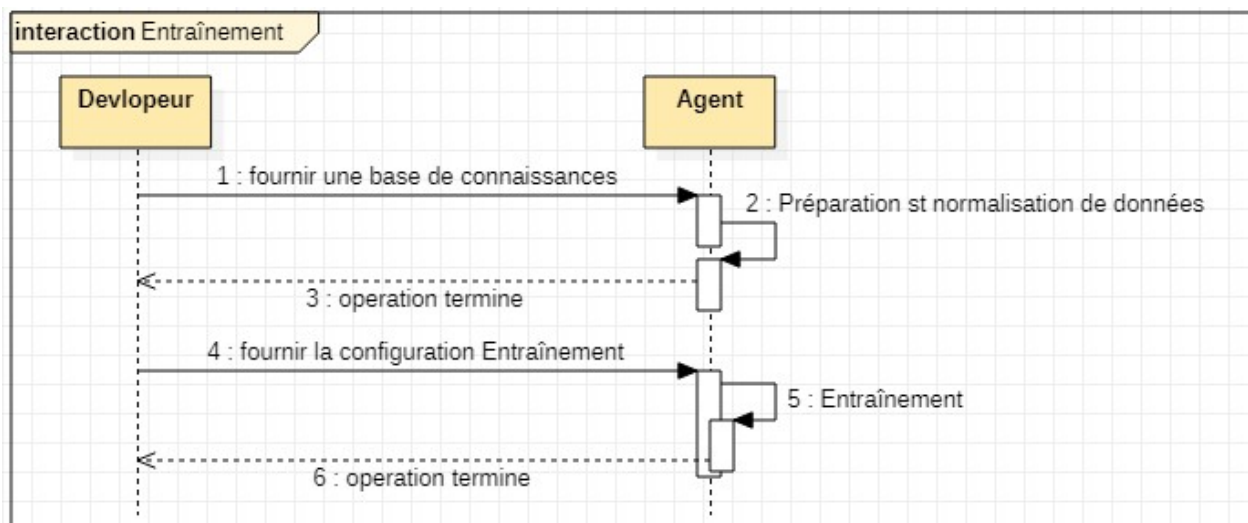


Figure 17: Diagrammes de séquence d'entraînement

e. Diagramme d'activités

Diagramme d'activité est une formalisation graphique des actions qui sont réalisées dans un cas d'utilisation. il est donc organisé en actions réalisées soit par un acteur, soit par le système, relié par une flèche indiquant l'enchaînement des actions.

i. Conversation

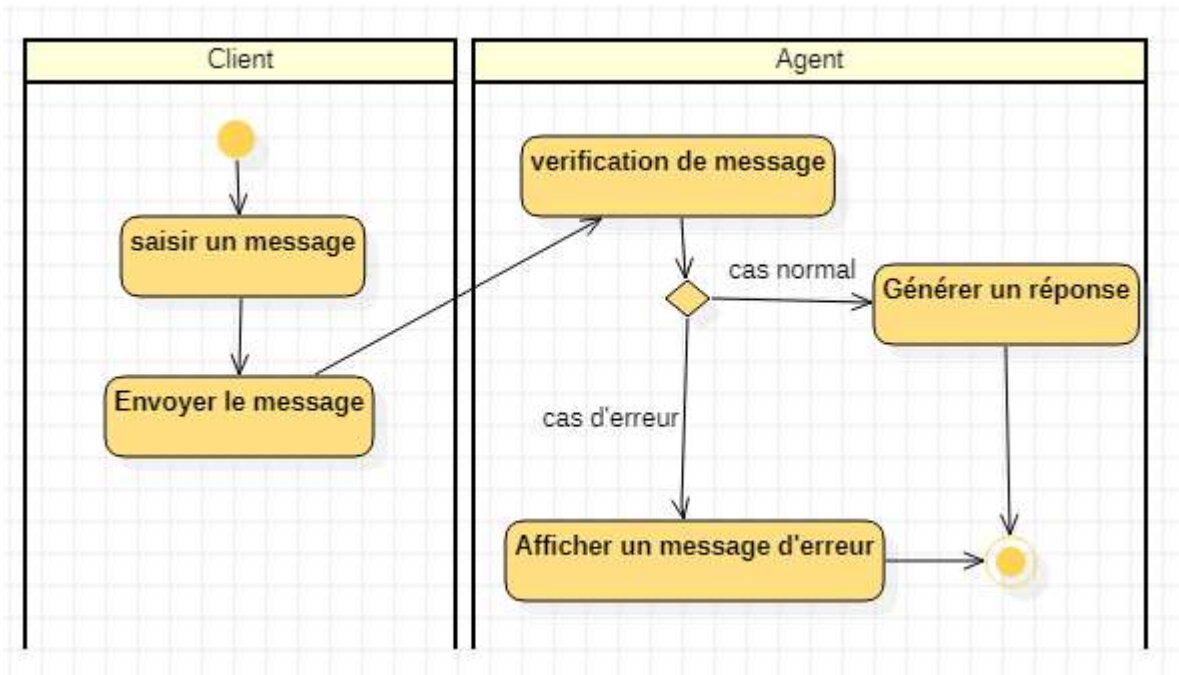


Figure 18:Diagramme d'activité de conversation

ii. Entraînement

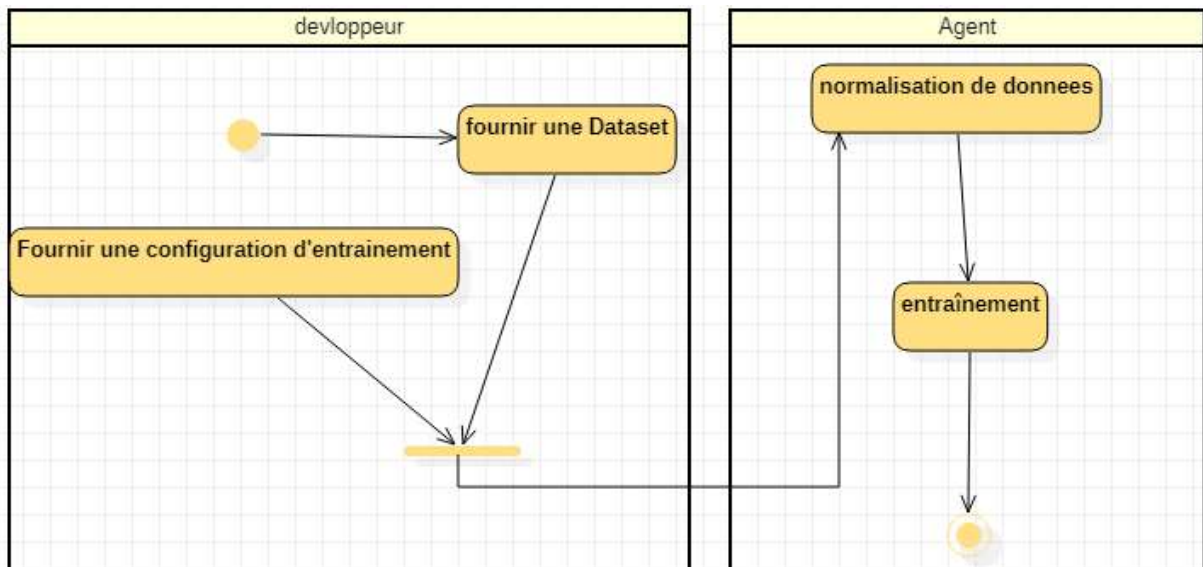


Figure 19:Diagramme d'activité d'entraînement

IV. Implémentation et réalisation

Cette phase décrit une étude d'un cadre de réseau neuronal unifié disponibles dans le lien : https://pytorch.org/tutorials/beginner/chatbot_tutorial.html, qui permet au système d'apprendre par supervision à partir d'un ensemble de données de dialogue (dataset), et elle décrit aussi l'amélioration de ce modèle via l'apprentissage par renforcement. La politique de former sur un ensemble de données fixe peut ne pas se généraliser correctement. Dans le dialogue parlé, le niveau de bruit peut varier selon les conditions et peut donc affecter considérablement les performances. Par conséquent, je vais implémenter une formation vise à améliorer le réseau formé par l'apprentissage supervise existant en utilisant un l'apprentissage par renforcement.

1. Etude de modèle existant

L'étude porte sur l'apport d'un réseau de neurones récurrent (Recurrent Neural Network - RNN) bidirectionnel encodeur/décodeur avec mécanisme d'attention, qui a le but de prendre une séquence de longueur variable en entrée et de renvoyer une séquence de longueur variable en sortie à l'aide d'un modèle de taille fixe.

a. Data set

Le modèle étudié utilise comme data set les scripts des films Corpus Cornell Movie-Dialogs disponibles dans le lien :

https://www.cs.cornell.edu/~cristian/Cornell_Movie-Dialogs_Corpus.html

Le Corpus Cornell Movie-Dialogs est un riche ensemble de données, il contient :

- 220 579 échanges conversationnels entre 10 292 paires de personnages de films
- 9035 personnages de 617 films
- 304 713 énoncés totaux

Beaucoup d'efforts est déployé pour préparer les données de ce corpus dans une liste de paires de phrases, et convertir les mots de ces paires de phrases en leurs index numériques correspondants et les transmettre aux modèles sous forme des tenseurs de nombres.

b. Encodeur

L'encodeur RNN utilisé parcourt la phrase d'entrée par un mot à la fois, à chaque timestep produisant un vecteur "de sortie" et un vecteur "d'état masqué ". Le vecteur d'état masqué est

ensuite passé au pas de temps suivant, tandis que le vecteur de sortie est enregistré. L'encodeur transforme le contexte qu'il a vu à chaque point de la séquence en un ensemble de points dans un espace de grande dimension, que le décodeur utilisera pour générer une sortie significative pour la tâche donnée.

Au cœur de cet encodeur se trouve une variante bidirectionnelle d'une unité récurrente **Gated Recurrent Unit** (Cho et al en 2014) à couches multiples, ce qui signifie qu'il existe essentiellement deux RNN indépendants : l'un qui est alimenté la séquence d'entrée dans un ordre séquentiel normal, et l'autre qui est alimenté la séquence d'entrée dans l'ordre inverse. Les sorties de chaque réseau sont additionnées à chaque timestep.

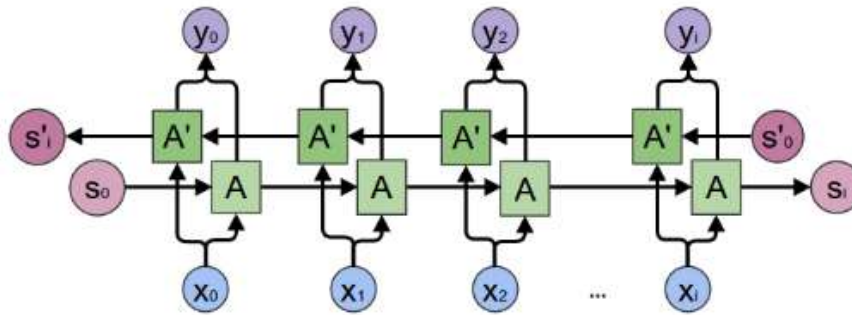


Figure 20: Bidirectionnel RNN

c. Décodeur

Le décodeur implémente un RNN avec le mécanisme d'attention en définissant un sous-module d'attention pour calculer les poids d'attention, ou énergies avec les fonctions de score :

$$\text{score}(\mathbf{h}_t, \bar{\mathbf{h}}_s) = \begin{cases} \mathbf{h}_t^\top \bar{\mathbf{h}}_s & \text{dot} \\ \mathbf{h}_t^\top \mathbf{W}_a \bar{\mathbf{h}}_s & \text{general} \\ \mathbf{v}_a^\top \tanh(\mathbf{W}_a [\mathbf{h}_t; \bar{\mathbf{h}}_s]) & \text{concat} \end{cases}$$

Figure 21: Fonctions de score

d. Phase d'apprentissage

L'apprentissage dans le contexte du notre modèle peut être décrite en 3 étapes :

- Etape #1 - Forward propagation : Cette étape permet principalement de calculer les activations de chaque neurone dans chacune des couches cachées ainsi que la valeur de la fonction coût qui mesure la différence entre cible et estimation, en apprentissage supervisé.

- Etape #2 - Backward propagation : Cette étape permet de calculer les gradients de la fonction coût par rapport aux paramètres du réseau de neurones (W et b , respectivement poids des connexions et biais). Rétropropagation du gradient (**back-propagation**) est basé sur le calcul des dérivées de fonctions composées.
- Etape #3 - Mis à jour des paramètres : cette étape permet de mettre à jour les paramètres (W , b) en fonction de leur gradient respectif calculé à l'étape précédente, et du paramètre de descente de gradient.

i. Fonction coût

On calcule la perte en fonction du tenseur de sortie de notre décodeur, du tenseur cible et d'un tenseur de masque binaire décrivant le remplissage du tenseur cible. Cette fonction de perte calcule la probabilité log moyenne négative des éléments qui correspondent à un 1 dans le tenseur de masque.

ii. Rétropropagation du gradient

Les tenseurs PyTorch peuvent se rappeler d'où ils viennent en termes d'opérations et de tenseurs parents qui les ont initiés, et ils peuvent fournir automatiquement la chaîne de dérivés de ces opérations en fonction de leurs entrées. PyTorch fournit automatiquement le gradient de n'importe quelle expression par rapport à ses paramètres d'entrée.

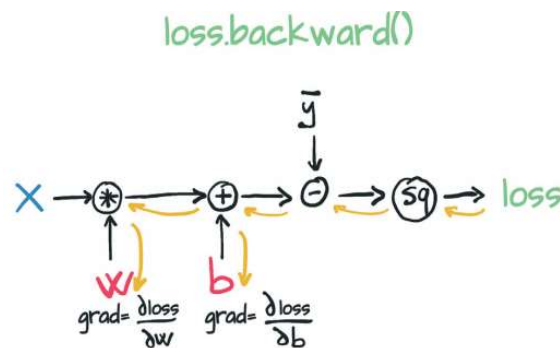


Figure 22: Rétropropagation du gradient

L'utilisation de la rétropropagation du gradient pose un problème nommé l'explosion du gradient, par l'augmentation très rapide des valeurs des gradients pendant la rétropropagation entraînant un dépassement de la capacité de la représentation interne des nombres et l'arrêt de l'apprentissage. Pour atténuer ce problème on utilise le mécanisme d'écrouissage de gradient en limitant artificiellement la valeur maximale des gradients.

e. Phase d'évaluation

Un problème courant, illustré dans la figure 22 est que le système se retrouve coincé dans une boucle infinie de réponses répétitives. C'est un exemple qui montre que le système avec l'apprentissage supervisé ne peut pas toujours générer les réponses correctement

```
> hello
Bot: hello . . . . .
> how are you
Bot: i m fine . . .
> what's your name
Bot: my name is edward . . .
> nice to meet you
Bot: i m sorry . . . .
> how old are you
Bot: i m a bit old man . .
> where are you from
Bot: south kent . the window . .
> do you have a family
Bot: no . . . . .
> ooh sorry
Bot: i m sorry . . . .
> it's nice to talk to you
Bot: i m sorry . . . .
> 
```

Figure 23: Conversation avec un agent avec un apprentissage supervisé

2. Amélioration avec l'apprentissage par renforcement

L'apprentissage par renforcement, c'est apprendre à agir par essai et erreur. Dans ce paradigme, un agent peut percevoir son état et effectuer des actions. Après chaque action, une récompense numérique est donnée. Le but de l'agent est de maximiser la récompense totale qu'il reçoit au cours du temps. Une description visuelle générale du cadre d'apprentissage par renforcement peut être vue dans la figure suivante.

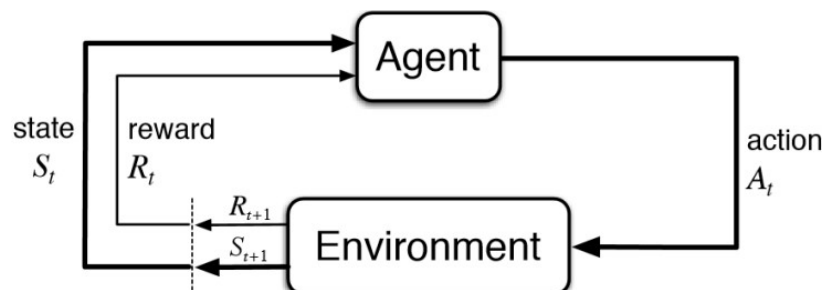


Figure 24: cycle d'apprentissage par renforcement

En RL, l'objectif assigné à un agent est de trouver une stratégie, l'association d'une action à chaque état, qui maximise la somme cumulée sur le long terme de toutes ces récompenses. Des comportements sophistiqués peuvent émerger d'une telle formulation. D'une part l'agent devra parfois sacrifier des gains à court terme au profit de ceux obtenus de manière différée. D'autre part, comme l'agent ignore généralement l'environnement dans lequel il évolue, il aura, pour agir de manière optimale, à concilier l'exploration de cet environnement et l'exploitation des connaissances qu'il y a déjà acquises. Le RL permet à un agent d'apprendre à partir de sa propre expérience plutôt que d'exploiter une expérience passée comme le fait l'apprentissage supervisé.

L'idée centrale ici est de simuler le processus de deux agents virtuels parlant à tour de rôle, à travers lequel nous pouvons explorer l'espace état-action et apprendre une politique qui mène à la récompense optimale attendue.

a. Simulation de dialogue entre deux agents

On simule le processus de deux agents virtuels (Backward Agent et Forward Agent qui est initialisé avec un apprentissage supervisé basé sur un corpus conversation) parlant à tour de rôle. La simulation se déroule comme suit : à l'étape initiale, un message de l'ensemble d'apprentissage est transmis au premier agent. L'agent code le message d'entrée en une représentation vectorielle et commence le décodage pour générer une sortie de réponse. En combinant la sortie immédiate du premier agent avec l'historique des dialogues, le deuxième agent met à jour l'état en codant l'historique de dialogue dans une représentation et utilise le décodeur RNN pour générer des réponses, qui sont ensuite renvoyées au premier agent, et le processus est répété.

b. Les fonctions de récompense

Les fonctions de récompense sont souvent conçues à la main, ce qui signifie que la formulation mathématique des propriétés importantes des conversations est requise au début. On va utiliser la somme pondérée de trois fonctions de récompense différentes :

La première tente de rendre les réponses faciles à répondre. Un ensemble de réponses ennuyeuses S est construit, et si la réponse à l'action en cours a est similaire à celles-ci, alors a reçoit une récompense négative

$$r_1 = -\frac{1}{N_S} \sum_{s \in S} \frac{1}{N_s} \log p_{seq2seq}(s|a)$$

Où N_S dénote la cardinalité de S et N_s dénote le nombre de jetons en s . $p_{seq2seq}$ représente la sortie de probabilité d'un modèle $seq2seq$ standard.

La deuxième récompense tente de capturer le flux d'informations dans une conversation. Plus précisément, chaque énoncé devrait contribuer à de nouvelles informations et elles ne devraient pas être répétitives par rapport aux précédentes. Afin d'obtenir une telle récompense, la similitude sémantique entre les phrases peut être pénalisée par :

$$r_2 = -\log \frac{h_{p_i} h_{p_{i+1}}}{||h_{p_i}|| ||h_{p_{i+1}}||}$$

Où h_{p_i} et $h_{p_{i+1}}$ désignent des représentations cachées du codeur RNN pour deux tours consécutifs p_i et $p_i + 1$. La fonction de récompense finale se concentre sur la cohérence sémantique. Cela récompense les réponses générées qui sont cohérentes et grammaticales en prenant en compte les informations mutuelles entre l'action a et les tours de dialogue précédents :

$$r_3 = \frac{1}{N_a} \log p_{seq2seq}(a|q_i, p_i) + \frac{1}{N_{q_i}} \log p_{seq2seq}^{backward}(q_i|a)$$

Où $P_{seq2seq}(a | q_i, p_i)$ représente la probabilité calculée par un modèle $seq2seq$ de générer une réponse à des énoncés de dialogue précédents donnés $[p_i, q_i]$. $p_{backward} seq2seq$ est un modèle $seq2seq$ formé avec des cibles et des entrées échangées. N_a et N_{q_i} sont le nombre de jetons dans les énoncés a et q_i respectivement.

c. Phase de test

Après l'entraînement de notre agent conversationnel on peut maintenant le tester

```
> hello
Bot: hello . . . .
> how are you
Bot: fine . . !
> what's your name
Bot: vincent . . . . .
> nice to meet you
Bot: the pleasure is waiting for you to say hello
> how old are you
Bot: thirty five . . .
> where are you from
Bot: san francisco . .
> do you have a family
Bot: no . ? ? ? ?

> 
```

Figure 25: Conversation avec agent avec apprentissage par renforcement

Conclusion

Ce projet décrit d'une façon détaillée la conception et la réalisation d'un Agent conversationnel par renforcement en se basant sur un modèle existant avec l'apprentissage supervisé.

Un bref aperçu de l'histoire des agents conversationnels a été donné et le modèle codeur-décodeur a été décrit en détail. L'apprentissage supervise et l'apprentissage par renforcement ont été également présentés, qui ont été proposées pour rendre l'agent conversationnel plus naturels et plus humains.

Bibliographie

<https://arxiv.org/pdf/1609.00777.pdf>

<https://medium.com/smileinnovation/lstm-intelligence-artificielle-9d302c723eda>

<http://colah.github.io/posts/2015-08-Understanding-LSTMs/>

<https://openclassrooms.com/en/courses/5801891-initiez-vous-au-deep-learning/5814656-decouvrez-les-cellules-a-memoire-interne-les-lstm>

<https://pathmind.com/wiki/lstm>

<https://newstrotteur.fr/2018/12/16/tackle-bias-et-autres-problemes-solutions-dans-les-modeles-dapprentissage-automatique/>

<https://arxiv.org/pdf/1606.02689.pdf>

<https://arxiv.org/ftp/arxiv/papers/2002/2002.09419.pdf>

<https://weave.eu/mecanisme-dattention-simple-astuce-mecanisme-universel/#sec5>