ÉCOLE
CENTRALELYON

UE PRO tc2 2021-2022- Projet d'Application Recherche

PAr 102: *Chatbot*

Auteurs :

M. Matthias Personnaz
M. Clément Yvernes

Encadrants :

M. Alexandre Saïdi
M. Philippe Michel

Version du
11 septembre 2022

Résumé

Les *chatbots* connaissent un essor considérable depuis les récents développements en traitement du langage naturel, avec à la clé des applications en renseignement, au sein des entreprises par exemple, pour constituer l'interface entre un grand corpus de données et un utilisateur désireux d'obtenir des informations dessus. Nous implémentons ici toutes les étapes qui constituent une méthode récupérative, en nous basant d'abord sur l'attribution à chaque mot d'un vecteur unique (algorithme Word2Vec) sur un corpus de faible taille du domaine universitaire. Puis, à l'aide de méthodes classiques de Machine Learning (algorithme $k-$NN) et d'une implémentation simplifiée de l'architecture QAnet sous Tensorflow, on constitue en chaîne les éléments d'un chatbot élémentaire. Les résultats obtenus sont mitigés ; la sémantique de notre corpus est bien assimilée par la vectorisation, mais le manque de données d'entraînement en nombre empêche de conclure sur la finesse d'extraction des informations que peut nous apporter le réseau conçu sur le corpus.

Abstract

Chatbots are increasingly used since the recent developments in natural language processing over the last years, with intelligence applications, within companies for example, to bridge a large corpus of data and users wishing to obtain information about it. We implement here all the steps comprised in a retrieval method, first by attributing to each word a unique vector (Word2Vec algorithm) on a small corpus about higher education. Then, using classical Machine Learning methods ($k-$NN algorithm) and a simplified implementation of the QAnet architecture under Tensorflow, we chain together the elements to build an elementary chatbot. The results obtained are mixed; the semantics of our corpus are rather well embedded within the vectorization, but the lack of training data prevents us from concluding on the fineness of the retrieved information that this design can truly perform on our corpus.

Remerciements

Nous remercions dans ce projet les tuteurs, Alexandre Saidi et Philippe Michel, ainsi que les professeurs du cours de Machine Learning de l'ENS Lyon pour leurs cours pertinents qui nous ont aidés dans les tâches du PAr, notamment Yohann De Castro pour ses conseils.

Table des matières

Table des figures

Liste des tableaux

Introduction

Ce PAr naît du besoin d'informer les élèves de l'École Centrale de Lyon sur la scolarité. Cela ce fait pendant une période de diversification des bassins de recrutement, des doubles diplômes et des formations proposés. La scolarité met à disposition un grand nombre de documents mais la compréhension de ces derniers n'est pas toujours aisée. L'intérêt de ce projet est alors de créer un chatbot permettant de simplifier leur consultation.

Ce document fait partie des livrables finaux attendus pour la réalisation de notre PAr. Il présente les enjeux, le contexte du projet, une reprise de l'état de l'art réalisé lors des jalons intermédiaires, un développement du travail effectué ainsi que nos résultats intermédiaires, finaux et une conclusion générale. Un autre livrable comprenant les codes auxquels fait référence ce rapport, se trouvent à l'adresse `https://gitlab.ec-lyon.fr/mpersonn/par102-chatbot`.

Dans ce document, les bibliothèques sont notées en rouge (comme `numpy`), les classes en bleu (comme `numpy.random`), et les fonctions en vert (par exemple, `numpy.random.rnd` ou `numpy.cos`).

Chapitre 1

État de l'art sur le traitement du langage naturel et les *chatbots*

1.1 Généralités historiques

Le *Cambridge dictionary* définit un chatbot comme « un programme informatique conçu pour tenir une conversation avec un être humain ». Pour mieux comprendre de quoi il s'agit précisément, nous devons revenir un peu en arrière dans le temps. Nous reviendrons aux *chatbots* bientôt.

On distingue de nombreux événements marquants dans le développements du traitement du langage naturel ou **natural language processing** (NLP), qui a donné les fondations nécessaires au développement de plusieurs problématiques voisines aux agents conversationnels : modèles de traduction, reconnaissance de texte, etc.

En 1950, Turing conceptualise un test – basé sur la conversation – permettant de distinguer un homme d'une machine. Ce sont en large partie les travaux de Chomsky de 1957 faisant état d'une structure innée du langage qui ont initié l'approche rationaliste du NLP, laquelle, s'est basée sur la programmation de règles et des raisonnements dans les systèmes NLP des années 60 jusqu'à la fin des années 80. Ceci constitue la première vague de progrès en NLP décrite par [6], qualifié de *rationaliste*. Cette approche, quand elle fonctionnait, donnait de bons résultats, mais manquait de souplesse et s'adaptait mal à de nouveaux domaines [6], car comprendre le langage nécessite de comprendre le sujet et le contexte, pas uniquement la structure des phrases [16].

C'est dans la seconde vague, l'*empirisme*, des années 1990 à 2000 que le **(shallow) machine learning** est né, en se basant sur l'analyse de larges quantités de données et des statistiques. Malgré tout, cette approche était encore loin de pouvoir absorber toutes données et les traiter, manquant par ailleurs d'un niveau d'abstraction nécessaire, ce qui sera plus tard apporté par des structures plus « profondes », basé sur un mimétisme de modèles de cellules inspirés de systèmes biologiques neuronaux que l'on assemble en cascade [6].

Ainsi le véritable essor fulgurant du NLP n'intervient qu'à partir du XXI$^{\text{ème}}$ siècle avec l'avènement du **(deep) machine learning** avec l'arrivée d'ordinateurs disposant de capacités de calcul suffisantes pour faire face aux complications grammaticales des langues (notamment en utilisant le mécanisme de rétropropagation dans les réseaux neuronaux multicouches). Cela correspond à la troisième vague de progrès décrite par [6], avec des résultats bien plus robustes et performants que les deux précédentes, respectivement le rationalisme et l'empirisme.

À partir des années 2010, on voit ainsi d'énormes progrès se produire, illustrés, par exemple, avec la victoire de Watson, un système conçu par IBM, au jeu télévisé américain de culture générale *Jeopardy !*, qui se basait sur une base de donnée issue d'internet en langage naturel issue d'internet. De nombreuses entreprises des GAFAM[1] ont également développé des assistants

[1] Google, Amazon, Facebook, Apple, Microsoft

personnels. En 2013, une équipe de Google a implémenté plus efficacement que les tentatives précédentes la technique du **word embedding** avec l'outil *Word2vec*, qui consiste à représenter les mots par des vecteurs, le terme *word-embedding* ayant été utilisé pour la première fois dans [2]. Plus récemment encore, en 2020, l'entreprise OpenAI a développé GPT-3, un **modèle de langage** statistique comprenant 175 milliards de paramètres, dépassant de plusieurs ordres de grandeur tous les modèles précédent. Ce modèle de langage permet entre autres d'écrire des textes proches de la rédaction qui serait faite par un humain, de traduire entre plusieurs langues, etc.

Citons, parmi les résultats remarquables effectués en matière de chatbot, le système ALICE (pour *Artificial Linguistic Internet Computer Entity*) qui en 1995, s'imposa en gagnant trois fois le prix Loebner [8, 1]. Depuis, les chatobt on trouvé leur intérêt. En sus des exemples précédemment mentionnés, un fait récent : en février 2020, un étudiant de l'université de Berkeley, Liam Porr, a utilisé les capacités de GPT-3 pour écrire un article entièrement avec cette IA, lequel a dupé nombre d'internautes (il l'explique ici).

Néanmoins, le risque encore trop pregnant des *chatbot* reste le manque de pertinence. En 2020, la société française Nabla a testé les capacités d'un chatbot basé sur GPT-3 appliqué à la prise de contact avec les patients d'une clinique. Lors d'un test, GPT-3 a conseillé à un patient de se suicider. Plus tôt, en 2016, un chatbot lancé par Microsoft, à peine mis en place a commencé à relayer des propos racistes. Si ces exemples sont extrêmes, ils n'en restent pas moins éloquents : ils soulignent que mêmes des outils en apparence évolués, s'ils ne sont pas entraînés correctement, peuvent donner des résultats hors de propos ou se montrer très néfastes.

1.2 Le traitement du langage naturel (NLP)

On peut dès lors formaliser pour ce rapport la notion : d'une manière très générale, le traitement du langage naturel ou en anglais *atural language processing*(NLP) est un domaine d'études à l'interface entre la linguistique et l'informatique et l'intelligence artificielle, qui traite le langage naturel humain et sa gestion par des ordinateurs.

1.2.1 Tâches courantes en Traitement du Langage Naturel

Parmi les tâches qui se réfèrent au traitement informatisé du langage naturel, on peut citer :

- L'analyse de sentiments, classification de textes en fonction du ton ;
- La réponse à des questions par rapport à un corpus ;
- Le remplissage de textes à trous ;
- La génération de textes ;
- Le résumé automatique de texte ;
- et bien d'autres encore.

Certaines de ces tâches sont relativement bien maîtrisées car il suffit en général d'une approche statistique pour les remplir, mais certaines autres relèvent un véritable défi. Par exemple, les deux dernières constituent, pour être acceptables, une réelle *compréhension* de la sémantique du langage. Ces tâches s'imposent comme les plus difficiles et la recherche les concernant s'intensifie.

1.2.2 Les outils linguistiques basiques

Parmi les méthodes fondamentales en linguistique incontournables qui sont communément employées depuis l'apparition du *machine learning*, et implémentées dans les bibliothèques de NLP, on peut citer les suivantes ([7]) :

- **Tokenization** (tokénisation) : consiste à découper un texte en éléments significatifs dénommés **tokens** (ou jetons, en français), qui peuvent être des mots, symboles, phrases. C'est un terme souvent utilisé en cryptographie, qui s'attelle au déchiffrement de textes chiffrés.
- **Part-Of-Speech (POS) tagging** : consiste à parcourir les phrases et à associer à chaque mot, phrase, symbole, une fonction (au sens grammatical), comme verbe, adjectif, adverbe, nom, etc. À un mot dans une phrase est associée une unique fonction, bien que ce mot puisse avoir différentes natures lorsque pris individuellement. Les catégories universelles de POS-tagging sont les suivantes par ordre alphabétique :

```
ADJ: adjective
ADP: adposition
ADV: adverb
AUX: auxiliary
CCONJ: coordinating conjunction
DET: determiner
INTJ: interjection
NOUN: noun
NUM: numeral
PART: particle
PRON: pronoun
PROPN: proper noun
PUNCT: punctuation
SCONJ: subordinating conjunction
SYM: symbol
VERB: verb
X: other
```

- **Stemming** (racinisation) : vise à épurer un mot en lui supprimant ses préfixes et suffixes pour obtenir le cœur du mot, potentiellement sa **racine** (au sens étymologique). Ce n'est pas un procédé fiable, et la racinisation est améliorée par la lemmatisation.
- **Lemmatization** (ou lemmatisation) : consiste à améliorer les techniques de racinisation en en essayant de trouver le *lemme* d'une *forme fléchie*, c'est-à-dire renvoyer sa forme élémentaire, que l'on pourrait trouver dans un dictionnaire. Cela peut être beaucoup moins évident que l'élimination de préfixes ou suffixes ; par exemple pour trouver l'infinitif des verbes irréguliers conjugués,[2] ou encore l'adjectif d'un superlatif ou d'un comparatif.[3] Contrairement à la racinisation, la lemmatisation prend en compte le contexte pour discriminer les différents sens d'un mot.
- **Parsing** : terme général qui englobe la construction de l'**arbre syntaxique** correspondant à une phrase donnée, d'une manière générale.
- **Named-entity recognition** : vise comme son nom l'indique à identifier les noms propres et entités, personnes, fictives ou réelles, du texte.
- **NP-chunking** ou **noun-chunk** : identification des phrases nominales simples (un groupe verbal qui commence par un nom).
- **Stop-words** : petits mots récurrents qui ne comportent pas d'information, comme les auxiliaires conjugués, les conjonctions de coordination, les articles, déterminants, etc.
- **Dependency parsing** : consiste à trouver les relations de dépendance entre les termes d'une phrase et à construire un **arbre syntaxique**, par exemple au sein d'un groupe nominal, puis d'un groupe verbal : relations entre adjectifs, génitifs, verbes, prépositions, déterminants etc. D'une racine découlent des descendants (et réciproquement les descendants ont des ancêtres). Cela donne lieu à un **arbre de dépendance** ou **arbre syntaxique**. En Python, les fonctions des bibliothèques `spacy.displacy.render` et `nltk.corpus.treebank` permettent de générer une telle image.

  La plupart de ces outils sont explicités dans [13]. Toutes ces méthodes sont proposées par la bibliothèque `spacy`. La bibliothèque `NLTK` en fournit aussi.

[2] Penser à « je suis » pour « être » ou « suivre ».
[3] Penser à « meilleur » pour « bon ».

1.2.3 Mise en œuvre et focus sur le langage Python

Les codes modernes permettant les fonctions précédemment citées utilisent des outils statistiques. On apprend dans la documentation de Spacy que l'étiqueteur, l'analyseur syntaxique, et les autres composants de `spacy` sont forgés par des modèles statistiques. Chaque valeur retournée est une prédiction basée sur les valeurs du modèle utilisé, qui sont estimées sur la base des exemples sur lesquels le modèle a été entraîné et qui ont servi à l'apprentissage supervisé. C'est un processus itératif dans lequel les prédictions du modèle sont comparées aux annotations de référence afin d'estimer le gradient de la perte. Ensuite un algorithme de rétropropagation. Par conséquent, comme dans tout modèle d'apprentissage machine, la qualité des résultats en phase de test est entièrement conditionnée par la qualité et le contexte de l'apprentissage.

Les méthodes statistiques modernes : de l'idée d'encapsulation à l'approche statistique pour approcher la sémantique

> For a large class of cases—though not for all—in which we employ the word "meaning" it can be defined thus : the meaning of a word is its use in the language.
>
> Wittgenstein's *Philosophical Investigations reads*, section 43

Calcul de N-grams et matrices de co-occurrence On peut ne plus considérer des mots uniques comme porteurs d'information, mais des $n-$uplets de mots consécutifs, qu'on appelle N-grams. Historiquement, cette idée est reliée aux processus markoviens, donc à mémoire limitée, en visant à essayer de décrire statistiquement les arrangements probables de mots adjacents dans une phrase. Cette approche est légitime car la sémantique des **langues analytiques** repose essentiellement sur leur syntaxe ; pour lesquelles l'information est transmise principalement selon l'ordre dans lequel sont arrangés les mots dans la phrase. Néanmoins, c'est un processus à mémoire limitée, ne serait-ce que parce que chaque phrase est grammaticalement indépendante des autres et véhicule son propre sens, et il est donc correct de limiter la taille de la fenêtre d'observation : les mots n'ont *a priori* de relations qu'avec quelques-uns de leurs voisins et le premier mot d'une phrase n'est vraisemblablement pas corrélé au dernier si la phrase est assez longue. Il est donc nécessaire et pertinent de s'intéresser à la distribution statistique des N-grams. Claude Shannon avait également introduit les N-grams lors de ses travaux en théorie de l'information. En traitement du langage naturel, il est légitime de s'intéresser à de telles constructions. Par exemple, le 2-gram « par exemple » a un sens clairement défini, que les deux mots *par* et *exemple* ne portent pas forcément lorsqu'ils sont pris individuellement. Pour aller plus loin, on peut également générer pour un texte sa **matrice de co-occurrence**, qui donne le probabilités d'apparition de tous les 2-grams.

Word-embedding Le **word embedding**, ou « encapsulation » de mots, est une technique relativement récente, dont le nom est apparu dans l'article « A neural Probabilistic Language Model » [3] par Yoshua Bengio, qui consiste à catégoriser et quantifier les similarités sémantiques parmi les éléments linguistiques [10]. Le *word embedding* est défini par [10] comme la *catégorisation et quantification de similarités sémantiques parmi les éléments linguistiques*. Cette approche a été poursuivie dans l'article de Ronan Collobert et Jason Weston « A Unified Architecture for Natural Language Processing » [5], dans lequel ils montrent comment l'utilisation de techniques d'apprentissage semi-supervisé améliore la généralisation de tâches partagées [7].[4]

[4] Dans [7] on peut lire « . . .in which [Collobert and Jason Weston] demonstrated how the use of multitask learning and semi-supervised learning improve the generalization of shared tasks. », page 76, même s'il n'est pas

One-hot encoding scheme Au début, les premières implémentations de cette technique utilisaient des vecteurs creux, avec une dimension égale à la taille du vocabulaire, aux coordonnées dans $\{0, 1\}$. Cela ne permet que la comparaison entre mots et ne donne évidemment pas d'informations sur les relations existantes entre les mots. On peut faire évoluer cette méthode vers ce qu'on appelle **count-vectorization**, qui consiste à simplement assigner à l'unique coordonnée non nulle du mot du corpus son nombre d'occurrence dans le texte.

Word2vec En 2013, un chercheur, Thomas Mikolov a proposé avec son équipe l'implémentation **Word2vec** de l'article intitulé « A neural probabilistic language model » [12], un algorithme d'apprentissage non supervisé qui prend en entrée un corpus et donne pour chaque mot en sortie un vecteur dans un espace continu (aux coordonnées comprises dans $\mathbb{R}$, par exemple) pour combattre le **fléau de la dimension** auquel les précédentes tentatives s'étaient heurtées. L'idée au cœur de l'article est de trouver un bon modèle pour l'approximation

$$f(i, m_{t-1}, \ldots, m_{t-n+1}) \approx P(w_t = i | w_{t-1} = m_{t-1}, \ldots, w_1 = m_1) \tag{1.1}$$

i.e. la probabilité de présence du mot i à l'emplacement t sachant les mots des emplacements $t-1$ à 1 donnés.

L'article récent [9], dans son état de l'art, donne encore l'analyse suivante : Word2vec *comprend et vectorise la signification des mots d'un document d'après l'hypothèse que des mots à la signification similaire dans ce contexte finissent proches les uns des autres dans l'espace vectoriel*. On peut légitimement se demander en quoi cela est une bonne idée. Pour certains contextes très spécifiques, il est raisonnable de vouloir tenter d'interpréter chaque mot (par exemple, des capitale) comme un vecteur dont chaque dimension représenterait un attribut (taille, population, etc.). Word2vec est en fait un **modèle de langage**, c'est-à-dire un modèle statistique qui rend compte de la distribution des arrangements entre les mots, ; selon leur usage. À ce propos, Wittgenstein affirmait que « la signification d'un mot est son usage dans le langage ». Plus spécifiquement, word2vec cherche à construire pour chaque mot un vecteur de telle sorte que les mots « proches » les uns des autres au sens sémantique, le restent au sens de la norme.

La dimension choisie varie en fonction de la taille du sous-ensemble sémantique auquel on se restreint, mais est usuellement comprise entre quelques dizaines (par exemple pour des sous-ensembles restreints comme le vocabulaire technique d'un domaine) et quelques centaines. Réduire la dimension permet souvent de diminuer le temps de calcul et peut parfois donner de meilleurs résultats.

Cette nouvelle technique a pu donner des résultats remarquables et bien plus utiles, rendant compte des relations **sémantique** entre les mots (leurs sens et les similarités entre eux). À titre d'exemple, après entraînement sur un ensemble de vocabulaire, la vectorisation peut donner lieu à des relations sémantiques comme Paris $-$ France $+$ Italie $\approx$ Rome. Cela sans intervention humaine ! Sur le site du dépôt du projet original https://code.google.com/archive/p/word2vec/, on apprend que la qualité des relations dépend fortement de la taille des données en entrée. Les quelques ensembles de la recherche évoqués comprennent plusieurs milliards de mots.

En python, les biblithèques `gensim` et `NLTK` permettent de construire des *word-embedding* Word2Vec. TensorFlow dispose également d'outils pour en créer.

La **vectorisation** a donc pris une place considérable parmis les approches par apprentissage non supervisé ces dernières années dû à ses performances et à l'approche sémantique qu'elle apporte dans de nombreuses applications, telles que la traduction, la recherche d'informations, et la synthèse de questions et réponses [7].

très clair à quoi cette phrase fait référence.

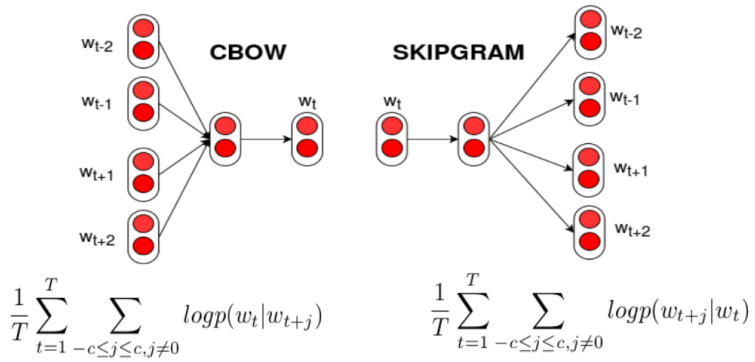


Figure 1.2 : Schémas simplifiés des deux architectures de réseaux neuronaux pour les approches CBOW et Skip-gram pour l'algorithme Word2vec, depuis https://fr.wikipedia.org/wiki/Word2vec

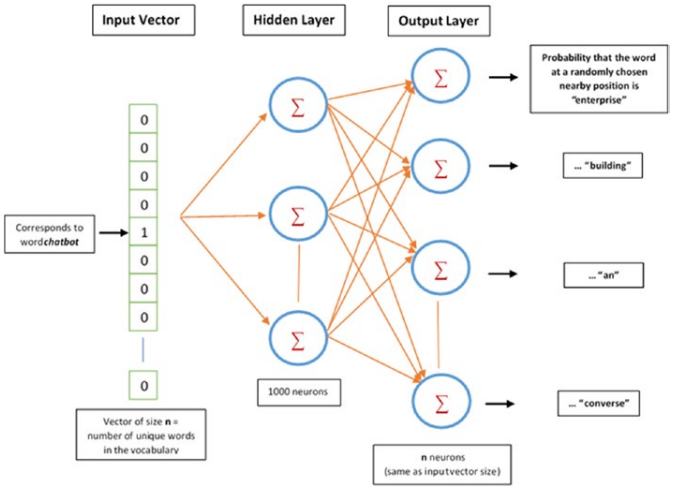


Figure 1.1 : Architecture par skip-gram d'un réseau neuronal pour l'entraînement d'un modèle de langage de type word2vec prescrite dans l'ouvrage [17], page 122.

Continuous bag of words (CBOW) et Skip-grams Il y a deux approches pour construire les représentations vectorielles de word2vec.

- Dans la même idée que l'analyse des N-grams, on peut tenter de prédire les mots environnants sur une fenêtre fixe en fonction du centre : c'est la méthode **Skip-gram**.
- L'approche **CBOW**, elle, utilise l'entourage pour prédire le centre (pour *Continuous Bag Of Words*). C'est le modèle utilisé par [12], qui, rappelons-le, est la publication qui a initié word2vec.

Niveau performances, on ne peut pas dire qu'il y ait réellement un consensus sur la question de la meilleure approche parmi ces deux-là. S'il est clair que Word2vec montre des performances meilleures que d'autres algorithmes de *word embedding*, il semblerait que l'approche par skip-gram soit meilleure que CBOW pour le rapprochement sémantique de paires de mots sur un certain corpus du milieu biomédical [9]. Les résultats dépendent beaucoup des métriques utilisées et des ensembles de données pris en entrée pour l'entraînement [9].

Principe d'entraînement et limites des *word embedding*

On commence généralement par des vecteurs aléatoires ou un *one-hot encoding* avec des vecteurs aléatoires, puis on itère en ajustant les poids de la couche visible d'un réseau de neurones afin de coller à la réalité du corpus, qui est l'ensemble des probabilités mesurées (de présence de mots les uns près des autres, selon les N-grams décrits précédemment). Voir figure 1.1. L'algorithme effectue une **descente de gradient stochastique** par **rétropropagation d'erreur**. Il s'agit précisément d'une technique d'apprentissage **auto-supervisée**. En Python, la bibliothèque `gensim` permet de construire de telles représentations et d'entraîner un modèle word2vec, mais c'est une étape assez coûteuse en puissance de calcul et il vaut donc mieux utiliser des modèles pré-entraînés sur 100 milliards de mots, fournis par Google, pour la langue anglaise du moins ([10], page 91).

Il faut néanmoins nuancer les apports d'une représentation de type Word2Vec. Il est remarquable qu'un tel apprentissage non-supervisé fasse émerger des relations sémantiques uniquement à partir de l'observation des distributions fréquentielles dans l'usage des mots. Il semble aussi qu'aucun résultat théorique n'ait pu prédire ces résultats, ce qui a fait le succès de Word2Vec lors de son invention. si l'on a vu que certaines relations élémentaires comme le genre semblent sensiblement constantes à travers le corpus de Google, il est malaisé de donner un sens à toutes les opérations possibles entre vecteurs.

C'est pourquoi cette approche seule n'est pas suffisante pour quelques des tâches les plus complexes en traitement du langage naturel comme le résumé automatique qui requiert une réelle compréhension des phrases, au sens linguistique.

1.3 *Chatbots*

Un chatbot est un programme d'intelligence artificiel qui dialogue avec l'utilisateur. Dans les faits, les chatbots peuvent remplir plusieurs fonctions :

- **Répondre à des questions**. Le chatbot répond aux questions de l'utilisateur en fonction des données présente dans un document ou dans un ensemble de documents.
- **Complétion de texte.** Le chatbot complète un texte en fonction des parties connues de ce dernier. C'est par exemple ce que fais GPT-3.
- **Dialogue à objectif.** Le chatbot discute avec l'utilisateur dans le but d'atteindre un objectif. Par exemple, il peut négocier un prix.
- **Conversation.** Le chatbot cherche à discuter avec l'utilisateur sans rechercher un quelconque objectif.

Un chatbot fonctionne généralement avec trois systèmes et des bases de données :

- **Un système d'écoute de l'utilisateur.** Ce système récupère la question de l'utilisateur et la transforme en querry. Une querry est une reformulation d'une question qui est comprise et prise comme argument par les systèmes suivants. Ce système peut comporter des éléments d'apprentissages et donc nécessiter une base de donnée d'entraînement.
- **Un système de calcul de la réponse.** Ce système cherche les éléments de réponses à la querry dans les documents prévus à cet effet. Ces documents forme donc une base de données du chatbot.
- **Un système d'expression** Ce système consiste à transformer la réponse calculée par le système précédent en un texte compréhensible par l'utilisateur.

On a représenté schématiquement le structure générale d'un chatbot dans la figure 1.3.

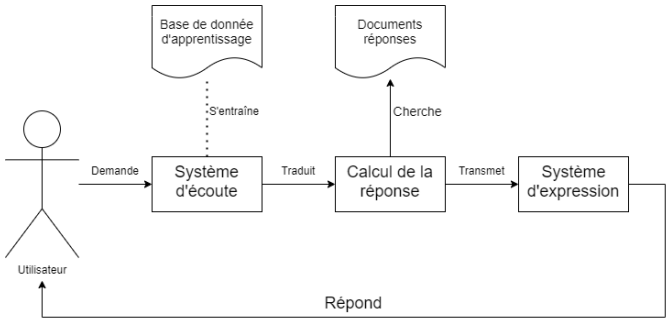


Figure 1.3 : Structure globale d'un chatbot

1.3.1 Modèles de chatbot question réponse

Il existe principalement deux approches pour générer des réponses aux questions de l'utilisateur. Un approche récupérative qui va chercher la réponse dans un ensemble conçu pour cela. Une méthode générative qui transforme directement les questions de l'utilisateur en réponses.

Les approches récupératives

Les approches récupératives répondent à l'utilisateur en cherchant l'information dans un ensemble donné. Cet ensemble est souvent composé d'un grand nombre de couples questions-réponses.

Les questions de l'utilisateurs sont alors transformées selon une méthode de NLP choisie. On fait de même avec les questions des couples questions-réponses. La réponse donnée à l'utilisateur est celle associée à la question la plus similaire de la question de l'utilisateur.

La similarité entre deux phrases dépend de la méthode choisie pour l'embedding. Si les phrases sont représentées vectoriellement alors toutes les normes sont envisageables. D'autres notions de similarité sont possibles comme celle de Jaccard ou celle implémentée dans Spacy.

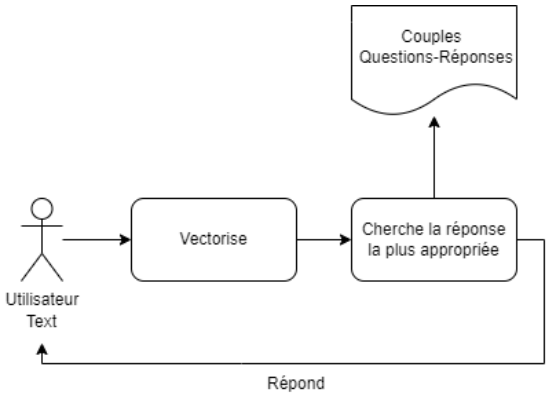


Figure 1.4 : L'approche récupérative

Les approches génératives

Les approches génératives quant à elles ne récupèrent pas les réponses dans un ensemble pré-établi, elles génèrent les réponses en lisant la question de l'utilisateur. Ce sont donc des approches très différentes car elles n'ont plus besoin de documents sources pour être déployées une fois l'apprentissage effectué.

Seq2Seq Ces approches transforment les phrases de l'utilisateur en réponses. On peut donc parler de modèle **sequence-to-sequence**. Ces approches étaient avant tout développées pour la traduction automatique de texte mais il est possible de les adapter aux problèmes de questions-réponses. Les tailles des phrases n'étant pas fixe, on utilise alors des réseaux récurrents (RNN). Ces approches sont composées de 3 parties.

- **Encodeur** : L'encodeur est partie du réseau qui va lire la phrase d'entrée. Il est composé de cellules récurrentes Long short-term memory (LSTM) ou Gated Recurrent Units (GRU) [4] mises en série. Chaque cellule va lire un mot et modifier l'état caché reçu par la cellule précédente. A la fin de la chaîne on obtient ce qu'on appelle un vecteur de pensée. Le même réseau peut être utilisé plusieurs fois dans la chaîne.
- **Vecteur de pensée (thought vector)** : Les vecteurs de pensée sont dans les faits les états internes des unité récurrentes. Bien que son interprétation est impossible pour un humain, ce vecteur représente d'une certaine manière le contexte de la phrase. Sa taille est donc plus grande que celle d'un vecteur de mot car un mot contient moins d'information qu'une phrase.
- **Décodeur** : Le décodeur est la partie du réseau qui à un vecteur de pensée génère une réponse. Comme l'encodeur, il est composé d'une chaîne d'unité récurrentes qui génèrent la phrase mot par mot. Les unités récurrentes génèrent des vecteurs de mots, il faut alors trouver le mot le plus proche pour formuler la phrase. Aussi le décodeur doit être capable de générer un mot de fin de phrase.

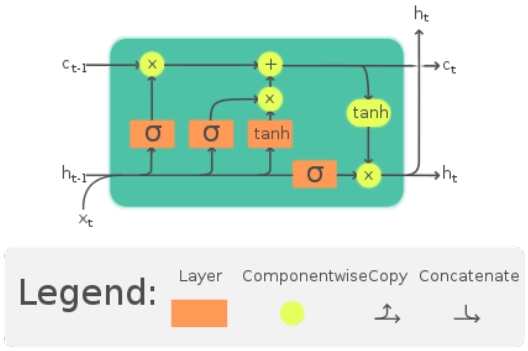


Figure 1.5 : Une unité récurente (LSTM) (Wikipédia)

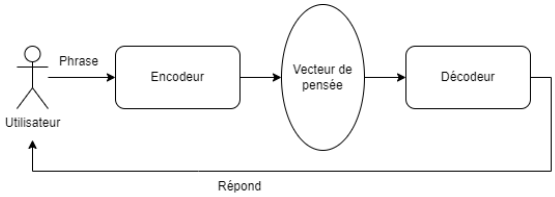


Figure 1.6 : Le modèle Seq2Seq

La prise en compte de l'attention Lorsque la phrase d'entrée devient trop longue, l'approche précédente perd un trop grand nombre d'information. D'autre part, les RNN standard (réseaux de neurones récurrents) souffrent de problèmes de gradient qui disparaissent et explosent. En particulier, les premiers mots ont une grande chance d'être oubliés. Les unités LSTM (Long Short Term Memory) et GRU (Gated Recurrent Unit) essayent de résoudre ce problème en conservant un vecteur de mémoire qui se dilue moins. Les LSTM traitent aussi ces problèmes en introduisant de nouvelles portes, telles que les portes d'entrée et d'oubli, qui permettent un meilleur contrôle sur le flux de gradient et permettent une meilleure préservation des «dépendances à longue portée». L'article [2] (« Neural Machine Translationby Jointly Learning to Align and Translate ») propose alors de permettre au réseau de ne porter son attention que sur les parties importantes de la phrase. Chaque cellule récurrente du décodeur reçoit une combinaison linéaires des états internes précédents. Les coefficients de cette combinaison sont déterminés par un autre réseau de neurones entraîné conjointement avec le modèle.

Les transformers L'article [18] (« Attention is all you need ») propose de conserver uniquement le travail sur l'attention en supprimant les aspects récurrents (donc, les couches récurrentes). Lors de sa parution la méthode proposée par l'article a résulté en un meilleur score jamais atteint sur un ensemble de données pour la traduction de l'anglais au français, selon sa propre description. Le réseau reçoit alors la phrase complète en encodant la position des mots. Cette nouvelle structure s'appelle un **Transformer**. La méthode est depuis assez utilisée et proposée par exemple dans la documentation de TensorFlow, dont voici une implémentation sur `tensorflow.org` appliquée à la traduction, ou encore une autre avec la bibliothèque `keras`. Le but des transformers et de prendre en compte le **contexte**. On crée alors réellement un **modèle de langage** et non seulement un word embedding. OpenAI s'est servi de cette approche Transofmers pour créer le modèle de language **GPT-3**. C'est aussi ce modèle qui a inspiré les modèles **BERT** et **CamenBERT** [11].

Combiner les deux approches générative et récupérative : Le Multi Seq-Seq Ces différentes approches on des avantages et des inconvénients [insérer tableau comparatif]. L'article propose alors de combiner ces deux approches afin d'obtenir un modèle plus robuste. L'idée est de concatener les vecteurs de pensées fournies par la question de l'utilisateur et par les réponses données par la méthode récupérative. Ce nouveau vecteur est alors utilisé par un décodeur pour générer la réponse.

Chapitre 2

Principaux résultats du projet

Dans ce chapitre, nous détaillons pas à pas les réflexions, choix de conception et approches employées en détaillant les protocoles et résultats intermédiaires et finaux, que nous avons été amenés à réaliser dans ce PAr pour répondre aux objectifs.

2.1 Sélection, acquisition, appropriation et traitement des données d'entraînement issues de la scolarité de l'ECL

2.1.1 Acquisition des données

La première étape a été de rassembler les documents pertinents publics. Nous nous basons essentiellement sur le règlement de scolarité du 6 juin 2021, ainsi que quelques documents explicatifs à propos des mobilités, césures, séjours internationaux, TFE et impacts du Covid-19 disponibles sur l'intranet de l'école. On désire une qualité du texte proche de la perfection pour biaiser l'analyse en NLP le moins possible. Cette tâche a mobilisé une vingtaine d'heures.

À l'aide de la bibliothèque `pdfminer` on peut extraire le texte. Les méthodes utilisées détaillées dans le code ci-dessous, repris à partir d'un modèle, est cependant un algorithme « glouton » (*greedy*), il se contente de balayer et détecter les zones de texte, et de recopier le texte linéairement. Cet outil donne donc des résultats inexploitables, par exemple, pour les documents techniques, ou faisant état d'un catalogue comme celui des cours du parcours électif avec de nombreux tableaux et données textuelles non-phrasées. Il donne en revanche une base de travail suffisamment satisfaisante pour des blocs importants de texte, et des résultats meilleurs que la plupart des bibliothèques faisant la même tâche.

2.1.2 Nettoyage des données

Grâce à la bibliothèque `re` (pour *regular expression* c'est-à-dire les expressions rationnelles qui constituent les grammaires de type 3 dans la hiérarchie de Chomsky), on peut facilement construire des automates pour la reconnaissance de motifs et leur remplacement ou leur élimination dans le texte brut. Ceci inclue notamment la suppression du texte présent en en-tête et pied-de-page, la numérotation des pages, la suppression des titres et numérotations d'annexes, la correction de mauvaise ponctuation, la suppression de caractères répétés à mauvais escient, le recollement de mots coupés, le remplacement des ligatures françaises (voire figure 2.1), le filtrage de caractères autorisés, l'inversion de lignes acquises dans le mauvais ordre du fait de l'indentation, la suppression de contenu inexploitable, non-phrasé, des réminiscences des tableaux, etc. On exploite la fonction `re.sub` (pour *substitution*).

Voici quelques exemples qui ont été utilisés pendant cette phase du travail :

$AE \rightarrow Æ$	$ij \rightarrow ij$
$ae \rightarrow æ$	$st \rightarrow ﬅ$
$OE \rightarrow Œ$	$ft \rightarrow ft$
$oe \rightarrow œ$	$et \rightarrow \&$
$ff \rightarrow ff$	$fs \rightarrow ß$
$fi \rightarrow fi$	$ffi \rightarrow ffi$

Figure 2.1 : Ligatures communes (source Wikipedia)

```
texte = regex.sub(r"\([^()]*\)",'',texte)     # enlever les parenthèses et leur contenu
texte = regex.sub(r"' +", "'", texte)     # enlever les espaces après les apostrophes
texte = regex.sub(r" +([,\.])",r'\1',texte)  # supprimer les espaces imprévu avant une
↪  ponctuation
texte = texte.replace(r"-\n", "")         # raccorder les mots coupés dans un paragraphe
texte = regex.sub(r" +",' ',texte)   # supprimer les répétitions d'espaces
```

Des interventions à la main sont souvent indispensables, pour inverser des lignes incorrectement acquises, corriger des anomalies uniques, etc. L'ordre dans lequel on procède pour cette opération de nettoyage est également primordial. Par exemple il faut évidemment remplacer les ligatures par le couple de caractères correspondants avant le filtrage des caractères autorisés communs. Compte tenu de la faible quantité de documents dont on dispose et de leur faible qualité, on procède par essai-erreur en ajoutant et ordonnant les opérations de nettoyage au fur et à mesure tout en relisant attentivement le texte obtenu.

2.1.3 Préparation des données

Cela inclue non exhaustivement la reconstruction de phrases héritées des listes à puces dans le fichier PDF, reconstitution d'ellipses (mots manquants ou implicites du fait de l'utilisation des listes à puces), remplacements de pronoms éventuellement en début de phrase faisant référence à une phrase précédente. Cette étape prend du temps, mais s'avère incontournable pour obtenir des phrases irréprochables pour un système de questions-réponses avec si peu de données.

2.1.4 Tokénisation et lemmatisation

On part des données textuelles prétraitées et nettoyées. Voici par exemple la première phrase du règlement de scolarité que nous avons extraite.

```
    Le Règlement de scolarité présente les modalités d'admission à l'École Centrale
de Lyon, les objectifs et les modalités de l'évaluation des connaissances et
des compétences de la formation ingénieur, les modalités de diversification
de cette formation et les conditions d'obtention des diplômes de l'École Centrale
de Lyon, hors diplômes de Master co-accrédités et diplôme d'Ingénieur Energie
en Alternance.
```

On utilise ensuite le parseur de `NLTK` afin de découper les phrases, puis un modèle de `spacy` pour tokéniser chaque phrase. Le code très réduit ci-dessous donne une idée des points exssentiels.

```
import nltk
from nltk.tokenize import sent_tokenize
sentences = sent_tokenize(texte, language='french')
```

```
import spacy
nlp = spacy.load('fr_core_news_sm')
phrasesSpacy = [nlp(sequence) for sequence in sentences]
```

Il s'agit ensuite de *lemmatiser* conditionnellement chaque *token* de chaque phrase. Le résultat de la commande `nlp(sequence)` est un **itérable** que l'on peut donc parcourir mot par mot pour avoir accès à chaque token ainsi qu'à ses attributs et propriétés. Les ponctuations et les **stop-word** sont supprimées. S'il fait partie d'un cluster de token constituant une **entité**, le mot est conservé, sinon il est lemmatisé à l'aide du modèle fourni. Le résultat fourni pour la phrase-exemple est donné ci-dessous. Notez les mots des expressions « École Centrale de Lyon » ou « Ingénieur Énergie en Alternance co-accrédités » n'ont pas été lemmatisés.

```
['règlement', 'scolarité', 'présente', 'modalité', 'admission', 'école', 'centrale',
'lyon', 'objectif', 'modalité', 'évaluation', 'connaissance', 'compétence', 'formation',
'ingénieur', 'modalité', 'diversification', 'formation', 'condition', 'obtention',
'diplôme', 'école', 'centrale', 'lyon', 'diplôme', 'master', 'co-accrédités',
'diplôme', 'ingénieur', 'energie', 'alternance']
```

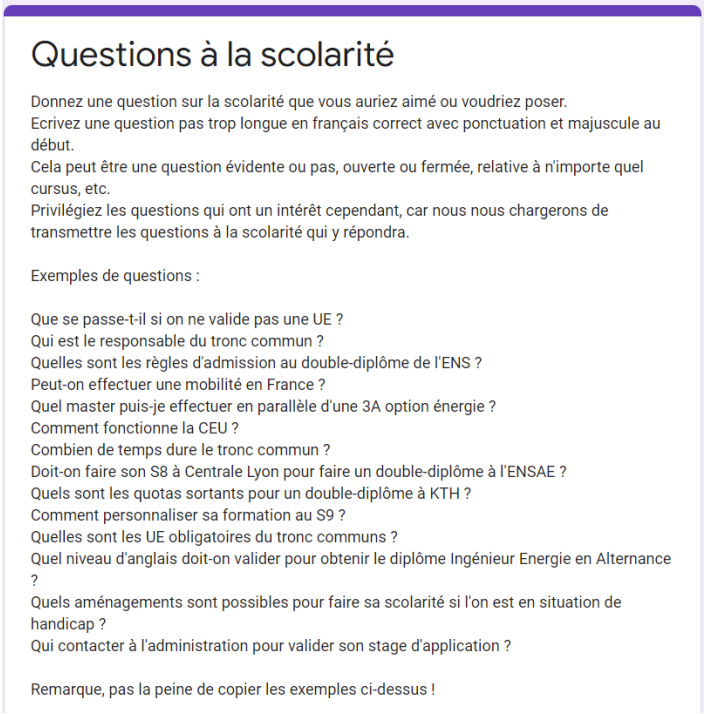
Il reste ensuite à construire le Word2Vec, par exemple avec 300 dimensions :

```
from gensim.models import Word2Vec
dimension = 300
skipgram = Word2Vec(sentences=tokenizedSentencesSpacy, window=6,
↪  min_count=1, sg=1, vector_size=dimension) # 1 pour skip gram, 0 pour
↪  cbow
word_embedding = skip_gram[skip_gram.wv.vocab]
```

2.1.5 Récupération et génération des questions et des réponses pour l'entraînement

Récupération directe des questions auprès des étudiants

Afin d'obtenir un ensemble de questions naturelles et suffisamment nombreuses sur la scolarité, on a décidé de les récolter directement auprès des étudiants. Pour cela, un formulaire (figure 2.2) a été publié sur les différents groupes de communication de l'école.



Questions à la scolarité

Donnez une question sur la scolarité que vous auriez aimé ou voudriez poser.
Ecrivez une question pas trop longue en français correct avec ponctuation et majuscule au début.
Cela peut être une question évidente ou pas, ouverte ou fermée, relative à n'importe quel cursus, etc.
Privilégiez les questions qui ont un intérêt cependant, car nous nous chargerons de transmettre les questions à la scolarité qui y répondra.

Exemples de questions :

Que se passe-t-il si on ne valide pas une UE ?
Qui est le responsable du tronc commun ?
Quelles sont les règles d'admission au double-diplôme de l'ENS ?
Peut-on effectuer une mobilité en France ?
Quel master puis-je effectuer en parallèle d'une 3A option énergie ?
Comment fonctionne la CEU ?
Combien de temps dure le tronc commun ?
Doit-on faire son S8 à Centrale Lyon pour faire un double-diplôme à l'ENSAE ?
Quels sont les quotas sortants pour un double-diplôme à KTH ?
Comment personnaliser sa formation au S9 ?
Quelles sont les UE obligatoires du tronc communs ?
Quel niveau d'anglais doit-on valider pour obtenir le diplôme Ingénieur Energie en Alternance ?
Quels aménagements sont possibles pour faire sa scolarité si l'on est en situation de handicap ?
Qui contacter à l'administration pour valider son stage d'application ?

Remarque, pas la peine de copier les exemples ci-dessus !

Figure 2.2 : Formulaire envoyé aux étudiants pour récolter leurs questions sur la scolarité

Cette approche a permis de récolter une cinquantaine de questions naturelles. Bien que certaines questions n'étant pas appropriées, il était donc nécessaire d'en supprimer quelques-unes.
Cela a rapidement montré ses limites. En effet, 50 questions ne sont pas suffisantes pour créer un chatbot fonctionnel. En parcourant les documents de la scolarité, nous avons généré à la main des questions et leur réponse pour chaque paragraphes. Finalement, cela a permis d'obtenir un total de 157 questions dont une centaine possèdent une réponse pour l'apprentissage avec QAnet (section suivantes). Ce travail est long et fastidieux, aussi a-t-on décidé de se limiter à un nombre restreint (moins de 200, donc) puisque nous n'aurions jamais atteint des ordres de grandeurs supérieurs[1].

Reformuler les questions pour augmenter la taille des données

Les questions récoltées et générées à la main n'étaient pas suffisamment nombreuses pour espérer créer un chatbot de bonne qualité. Il fallait donc augmenter le nombre de questions. Cela ne pouvait pas se faire à la main, il fallait donc utiliser un outil de reformulation automatique. Pour ce faire il a été décidé de reformuler par double traduction. A l'aide d'un logiciel les questions sont traduites dans une langues puis retraduites en français. Les phrases obtenues sont généralement des paraphrases des phrases d'origine.

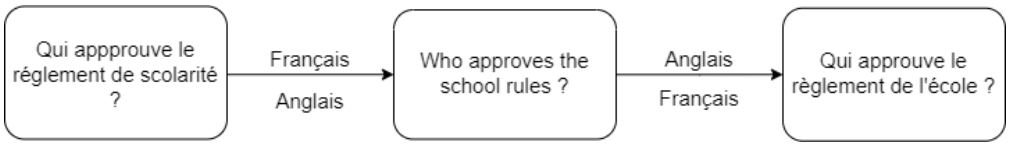


Figure 2.3 : Exemple de reformulation par double traduction

[1] Ce qui serait pourtant nécessaire !

Le temps d'entraînement d'un réseau de traduction étant très long, les traductions ont été réalisées sur un traducteur en ligne. Les reformulations obtenues sont très satisfaisante. En appliquant plusieurs fois ce processus en changeant la langue intermédiaire. Il a été possible d'obtenir un ensemble de 1570 questions.

2.2 Réalisation d'un *Word Embedding* de type Word2Vec sur les données de la scolarité

La première étape de NLP à proprement parler que nous avons pris le parti de faire, est évidemment un *Word Embedding*. Comme déjà évoqué dans l'état de l'art, il permet de capturer les relations sémantiques des termes d'un corpus en se basant sur leur contexte.

Puisqu'il s'agit d'une étape fondatrice, on détaille la création du Word2Vec à partir des données brutes. On ne s'attarde pas sur les subtilités du code et du traitement de texte, qui représentent néanmoins la majeure partie du travail fourni pour arriver au résultat. Ce résultat peut être ensuite réutilisé dans d'autres bibliothèques de *Machine Learning* comme `tensorflow.keras` afin d'initialiser les poids des couches d'*embedding*.

On a exploité les bibliothèques `gensim`, `NLTK` et `spacy`.

Ce travail a donné lieu à une bibliothèque et à un travail de POO dont voici le diagramme des classes est donné en figure 2.4.

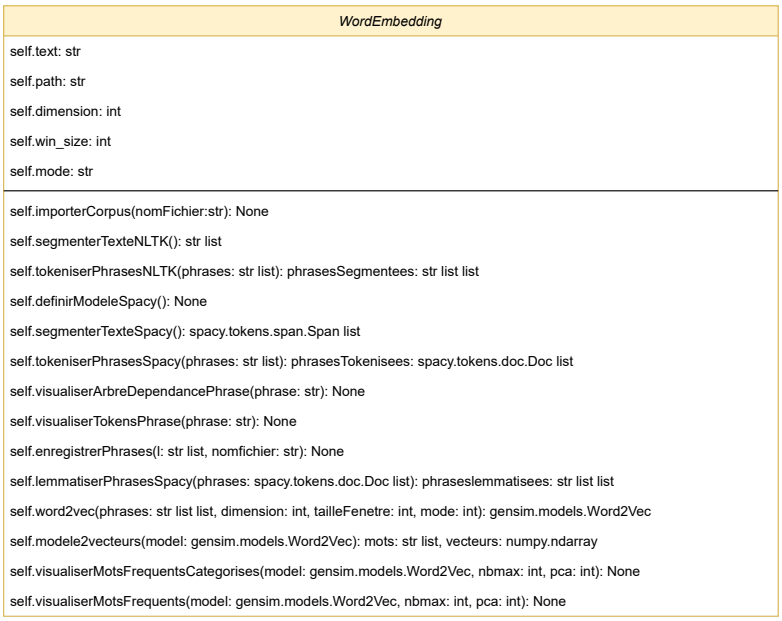


Figure 2.4 : Classe `Word2Vec`

Les vecteurs sont évidemment en trop grande dimension pour être visualisés tels quels. Pour les projeter en deux dimensions, des possibilités s'offrent à nous avec `sklearn`. L'usage est d'abord de faire une analyse en composantes principales `decomposition.PCA`, algorithme déterministe linéaire qui préserve la variance des données et leur structure globale, et/ou `manifold.TSNE`, algorithme stochastique non-linéaire qui préserve la structure locale des données (les distances).

En figure 2.5, nous avons une visualisation des 50 mots les plus fréquents dans le corpus. La figure est une représentation en 2D des points de l'espace de départ de dimension 250 dans l'*embedding*.

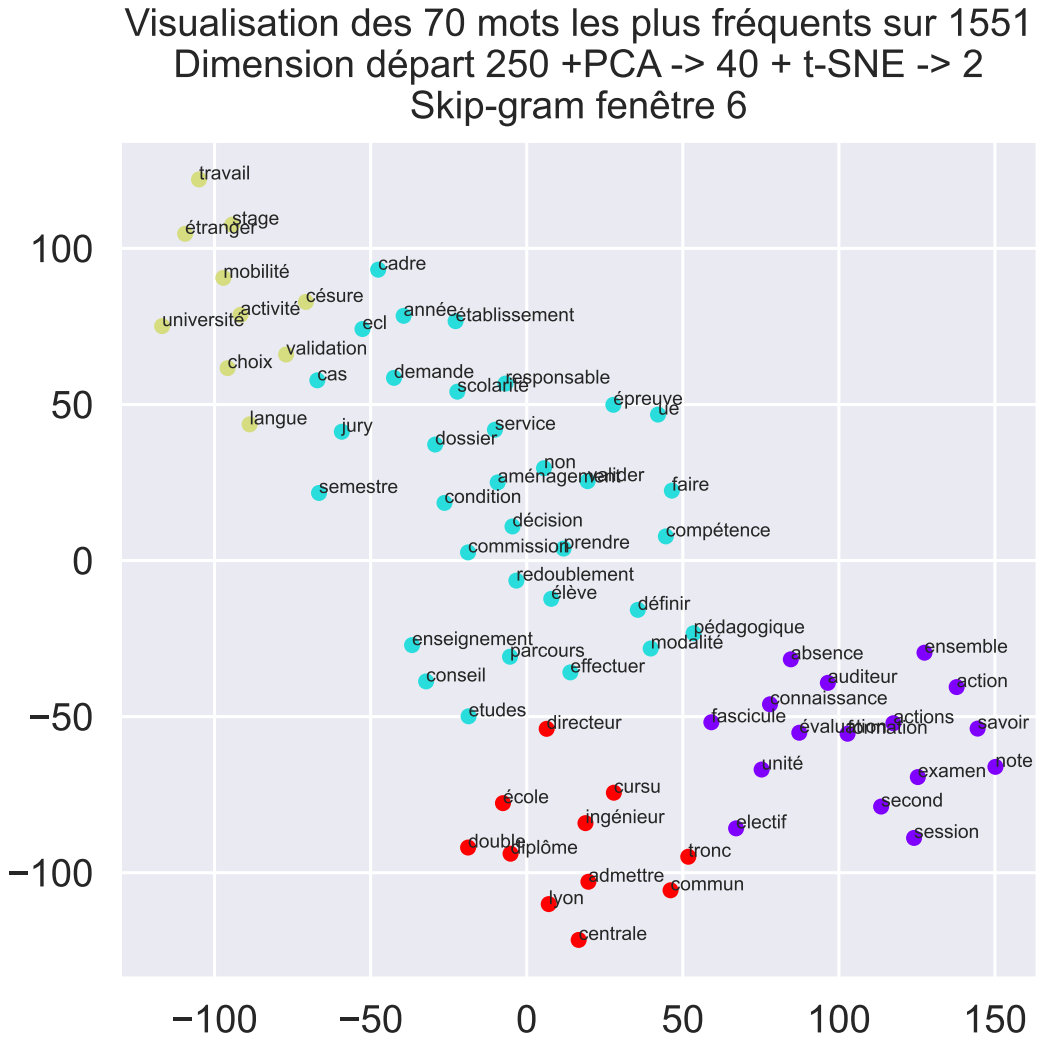


Figure 2.5 : Visualisation des 70 mots les plus fréquents du corpus par Word2Vec entraîné avec Gensim avec les documents de scolarité et les paramètres indiqués. Avant la projection dans le plan 2D, nous avons catégorisé les 70 vecteurs affichés à l'aide de `sklearn.cluster.KMeans` (algorithme du plus proche voisin) en 4 catégories. On voit que `gensim` a réussi à capturer la sémantique des mots à travers la vectorisation.

On distingue très clairement dans l'exemple figure 2.5 des « pôles » associés aux différentes activités de l'école Centrale. Un axe semble représenter le temps qui s'écoule diagonalement vers le coin supérieur gauche, aboutissant plutôt les mobilités, césures et formations en double-cursus, tandis qu'en bas à droite, les activités académiques soumises à validation. En bas à gauche, les termes essentiels qui caractérisent l'école.

2.2.1 Étude de l'espace sémantique des documents de la scolarité

Nous avons ensuite étudié l'espace sémantique des mots ainsi obtenus avec les documents de la scolarité. Un Word2Vec donne une distribution de vecteurs, que l'on suppose suivre une loi statistique reflétant les relations sémantiques qui existent entre les mots.

Similarité cosinus Pour deux vecteurs réels x et y en dimension quelconque (finie), leur similarité cosinus vaut

$$d(x,y) = \frac{\langle x, y\rangle}{\|x\|_2 \|y\|_2} \tag{2.1}$$

On peut par exemple tester avec « scolarité », « diplôme », etc.

```
skipgram.wv.most_similar(['Centrale'])

[('Lyon', 0.9979485273361206),
 ('ingénieur', 0.997929573059082),
 ('École', 0.9978417158126831),
 ('obtention', 0.9974856972694397),
 ('Directeur', 0.9974188804626465),
 ('année', 0.9973104000091553),
 ('partenair', 0.997223436832428),
 ('diplôme', 0.9971845746040344),
 ('formation', 0.997157633304596),
 ('titre', 0.9971100091934204)]
```

Optimisation de l'entropie selon chaque axe au sens de la théorie de l'information

Notre espace de mots comporte autant de dimensions que l'on veut : c'est un paramètre. Il est encore difficile à l'heure actuelle d'expliquer comment les algorithmes de *machine learning* parviennent à leurs solutions. Word2Vec tente de minimiser une fonction de *loss* et ne fait pas exception. L'analyse en composante principale tente par exemple de maximiser la variance par changement de repère. Les recherches antérieures comme nous l'avons vu, peuvent mettre en évidence le sens de directions privilégiées (comme le genre). Nous faisons l'hypothèse que chaque axe brut des données (dimension) est porteuse d'un sens que la machine détermine en fonction des textes. Ainsi, nous allons tenter de choisir un nombre de dimensions qui maximise l'entropie de la distribution selon chaque axe pris indépendamment. Selon cette hypothèse, l'information partielle selon l'axe (donc la loi marginale) aide à discriminer les données.

Nous procédons ainsi : soit $(x_1, \dots, x_N)$ l'ensemble des vecteurs des mots dans $\mathbb{R}^d$: $x_i = (x_i^{(1)}, \dots, x_i^{(d)})^\intercal$. Pour tout j dans $[\![1, d]\!]$, nous considérons l'ensemble des $j-$ièmes composantes $E_j = \{x_i^{(j)}, \quad 1 \leq j \leq d\}$ que l'on classe en C classes (on fait un histogramme). À chaque classe $c \in \{1, \dots, C\}$ on associe la proportion $p_c^{(j)}$ de vecteurs dont la $j-$ième

	$d = 5$	$d = 10$	$d = 20$	$d = 40$	$d = 60$
CBOW	$1,256$	$0,879$	$0,591$	$0,418$	$0,327$
Skip-Gram	$1,333$	$1,140$	$0,845$	$0,591$	$0,476$

Table 2.1 : Moyenne des entropies de chaque axe pondérées par l'écart-type de la distribution selon l'axe. Fenêtre de taille 5, 25 itérations lors de l'entraînement.

composante est dans l'intervalle I_c (ainsi $E_j \subset \bigcup_{i\in\{1,\dots,C\}} I_c$). Nous calculons ensuite la quantité à maximiser suivante :

$$\frac{\sum_{1\leq j\leq d} H(p_1^{(j)},\dots,p_c^{(j)})\mathrm{STD}(E_j)}{\sum_{1\leq j\leq d}\mathrm{STD}(E_j)} \tag{2.2}$$

où STD signifie *standard deviation*, i.e. écart-type, et H l'entropie de Shannon de la probabilité $(p_1^{(j)},\dots,p_c^{(j)})$:

$$H : p \in \{p \in [0,1]^{\mathbb{N}}, p \text{ probabilité}\} \mapsto \sum_{g\in\mathbb{N}} -p_g \log_2(p_g) \tag{2.3}$$

C'est-à-dire que l'on calcule la moyenne des entropies des distributions approchées en classes selon chaque axe sur les C classes, pondérées par l'étalement des valeurs des composantes selon chaque axe (voir figure 2.6). L'intuition derrière ce calcul, est que plus l'entropie de la distribution selon un axe est élevée, plus elle permet de discriminer les vecteurs, mais on doit prendre en compte l'étalement également car des valeurs peu étalées (proches) permettent moins de discriminer les vecteurs. Cette métrique permet ainsi de comparer les Word2Vec entre eux et les dimensions. Les résultats sont consignés dans la table 2.1. Cette mesure semble très clairement privilégier les petites dimensions. Le résultat était un peu prévisible au sens où notre dataset de phrases comprend peu de mots (vocabulaire de taille inférieure à 2000) et très proches les uns les autres, sur lequel un traitement de simplification (lemmatisation, etc) a été appliqué. Pour comparaison, Google avait utilisé 300 dimensions pour la langue anglaise (avec 3 millions de mots). Skip-gram ressort également meilleur : c'est « normal » puisque skipgram prend plusieurs mots pour deviner le centre, il y a donc moins de variabilité (c'est un texte à trou, pas une prédiction de champ lexical comme CBOW).

Critique de la métrique La faiblesse de cette approche est évidemment de regarder les lois marginales qui ne permettent pas de reconstruire les distributions. À unique distribution typique discrète donnée de valeurs pour chaque axe (disons n valeurs possibles sur l'axe), des permutations dans les valeurs prises par tous les vecteurs sur chaque axe permettent de construire un espace sémantique de taille $\mathcal{O}(n^d)$. Néanmoins, cette mesure permet bien de sanctionner les axes peu porteurs de sens.

Nous avons mené plusieurs tests empiriques et informels avec des dimensions entre 200 et 300, différentes tailles de fenêtre autour de 3 à 6, et il semble également que Skip-gram donne de bien meilleurs résultats que CBOW.

Optimisation des corrélations statistiques

Prenons un exemple et comparons un Skip-gram de dimension 16 et un de 32 sur le même corpus avec les mêmes paramètres d'apprentissage. Nous verrons dans les deux paragraphes suivants, que l'étude statistique des deux cas présentés conduit à privilégier une plus petite dimension (donc 16).

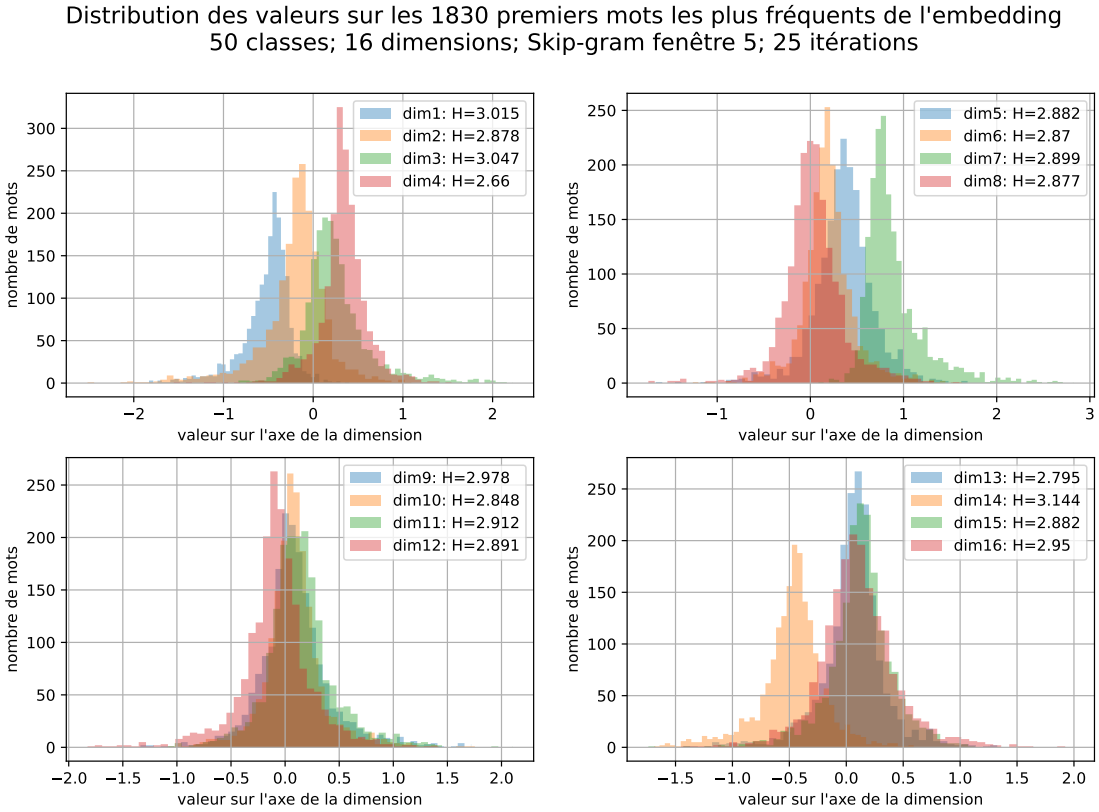


Figure 2.6 : Distributions des valeurs et entropies associées pour un Skip-gram en dimension 16

Matrices de similitude empirique des mots présents En dehors de la théorie de l'information on peut aussi simplement observer les corrélations statistiques, soit entre les mots, soit entre les dimensions. Si deux dimensions sont très corrélées, l'une est peut-être redondante. Si beaucoup de mots sont corrélés entre eux, la dimension de l'espace sémantique est probablement trop faible.

En calculant les matrice de similitude cosinus non normée des 30 mots les plus présents dans le corpus

$$(\langle x_i, x_j \rangle)_{1 \leq i,j \leq 30} \in \mathscr{M}_{30}(\mathbb{R}) \tag{2.4}$$

qui est donnée sous forme d'image en figure 2.7, il est clair qu'une dimension plus grande distingue mieux les termes au sens différent et conserve les similarités cosinus des termes similaires. Or, à terme, dans les modèles d'apprentissage, c'est la similarité cosinus qui permet de recréer un mot. Il est donc plus judicieux d'augmenter la dimension lorsque le sens sémantique est mieux capturé, à plus forte raison pour les mots les plus fréquents qui ont une importance vis-à-vis du sujet.

Matrices de covariance empirique des dimensions et données statistiques On peut aussi calculer les matrices de corrélation des dimensions au travers de *tous* les mots, ce qui est la première étape d'une décomposition en composantes principales : le calcul de

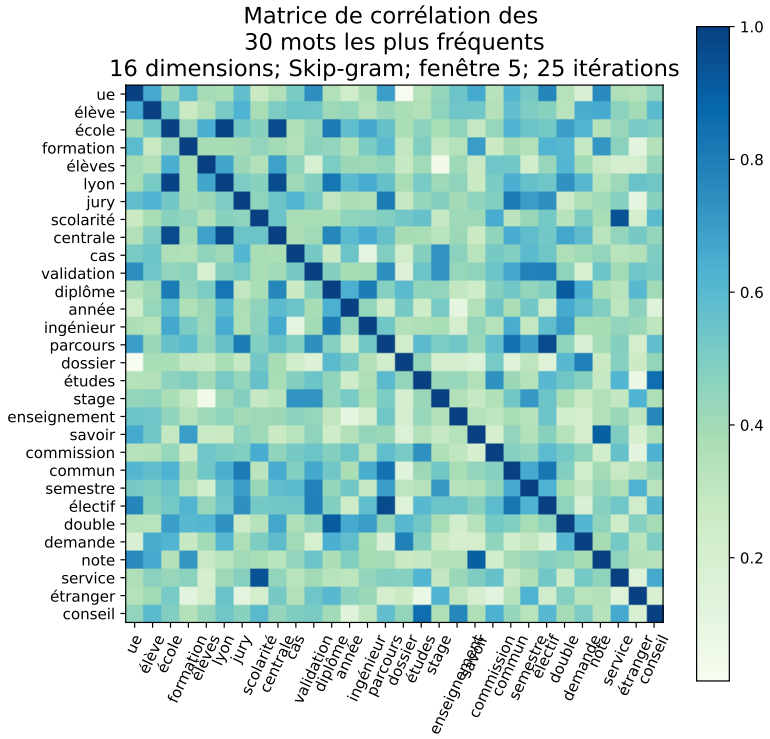
$$\sum_{i=1}^{N} x_i x_i^\intercal \in \mathscr{M}_d(\mathbb{R}) \tag{2.5}$$

qui est donné sous forme d'image dans la figure 2.8

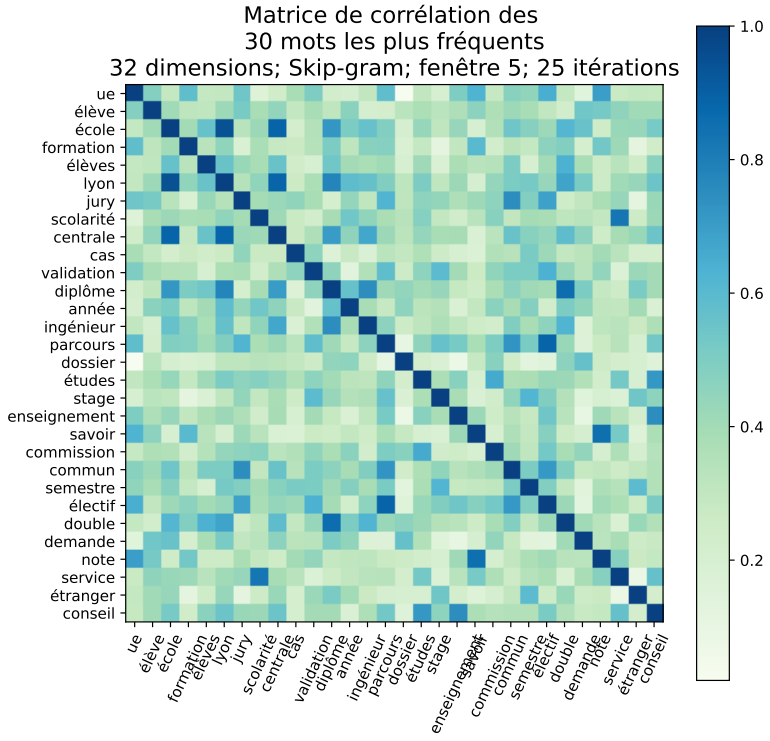
2.2.2 Conclusion intermédiaire : Word2Vec

À cause du fléau de la dimension, l'information marginale apportée par l'ajout d'un axe sur lequel représenter les mots augmente avec d, i.e. à N fixé la suite $\left(\frac{N^{d+1}-N^d}{N^d}\right)_d$ est croissante. Cette simple observation permet d'utiliser 16 ou 32 dimensions pour notre vocabulaire (1500 mots environ suivant le comptage et le pré-traitement de lemmatisation) et à Google de représenter la langue anglaise sur seulement 300 dimensions (3 millions de mots). Autrement dit un rapport de 10^3 en mots et seulement 10^1 en dimensions.

Conformément à la littérature déjà existante, on ne *connaît pas* de critère décisif sur la bonne dimension à adopter en fonction du corpus. Mais nous avons pu voir au travers de l'étude de quelques cas raisonnables, qu'une petite dimension est suffisante pour notre corpus, qui est de plus un vocabulaire très spécialisé (enseignement supérieure, doublé de vocabulaire propre et redondant à l'école). Dans une optique de réduction des temps de calcul et complexité en temps/mémoire, une dimension plus faible reste évidemment privilégiable. La corrélation entre les axes et la corrélation entre les mots préconise d'augmenter la dimension, tandis que l'entropie et les distributions suggèrent qu'une plus petite dimension suffit au voisinage des valeurs qui nous intéressent. En pratique on choisira donc arbitrairement un entier dans la fourchette entre 16 et 32, qui respecte le besoin vis-à-vis de l'ordre de grandeur du vocabulaire. Il est inutile, et inenvisageable d'un point de vue des puissances de calcul disponibles, de choisir $d > 32$.

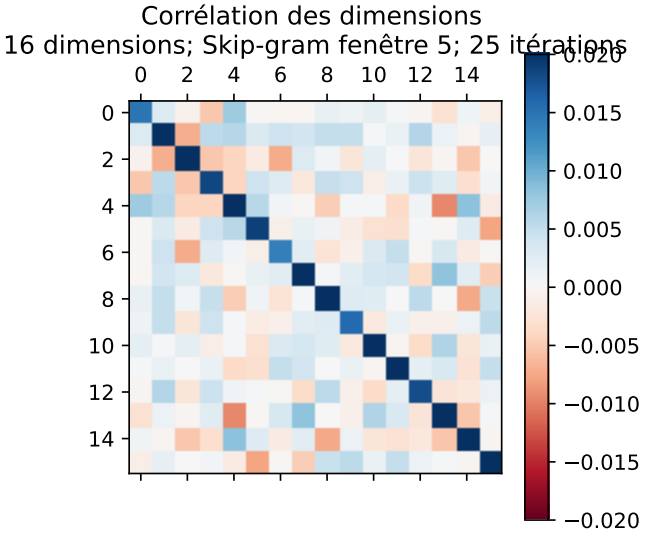


(a) dimension 16

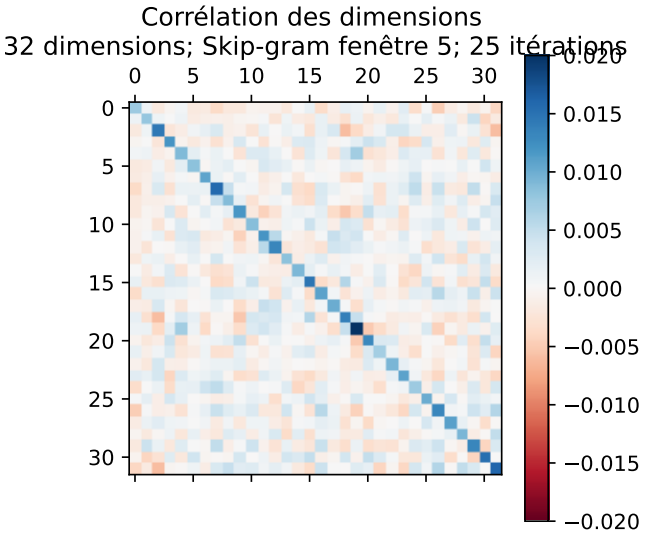


(b) dimension 32

Figure 2.7 : Matrices de corrélation des 30 mots les plus fréquents du corpus. Une dimension plus grande permet de distinguer mieux les termes (les valeurs sont dans l'ensemble plus proches de 0). On constate que les produits scalaires sont presque toujours positifs (certaines négatifs existent au travers de l'ensemble complet), cela s'explique par la forte concentration de beaucoup de vecteurs dans un même cadran, comme montré par la figure 2.9

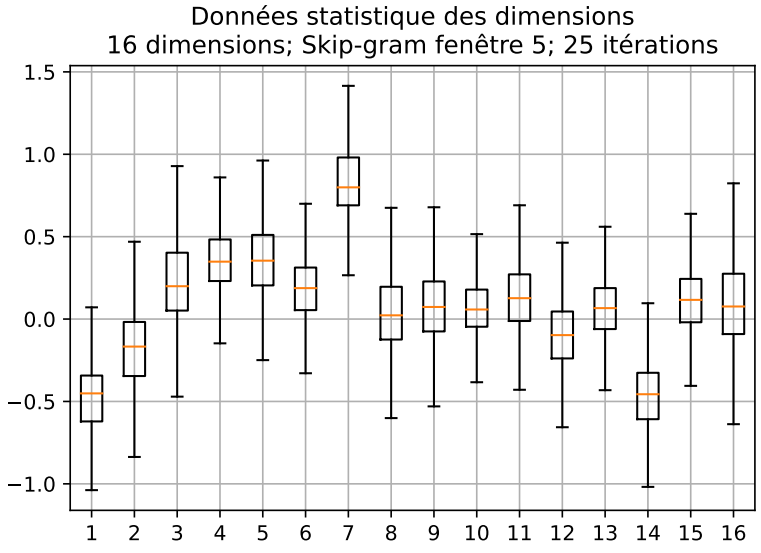


(a) dimension 16

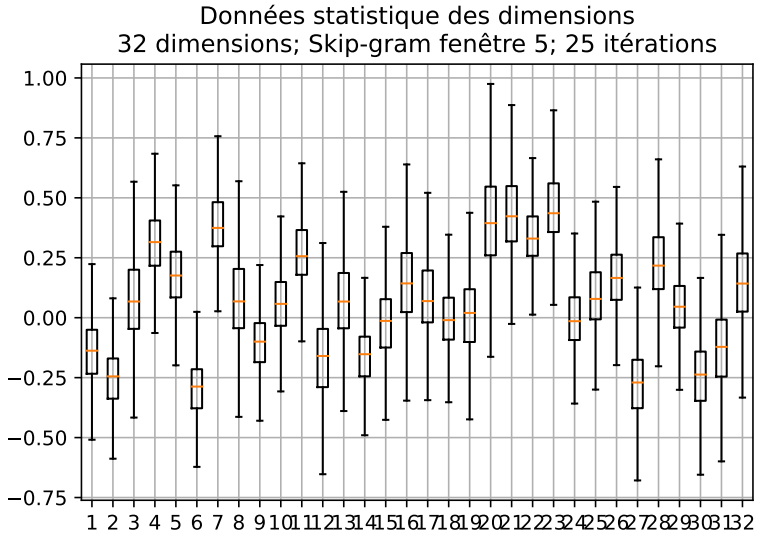


(b) dimension 32

Figure 2.8 : Matrices de corrélation des dimensions. Les valeurs saturent au delà de l'échelle (sur la diagonale). Les vecteurs n'ont pas été normés, et la diagonale n'est donc pas constante. Les échelles sont identiques sur les deux images pour la comparaison. Là encore, les couleurs plus proches du blanc (ce qui indique des valeurs nulles) globalement pour $d = 32$, indiquent que les dimensions sont plus indépendantes.



(a) dimension 16



(b) dimension 32

Figure 2.9 : Distributions des valeurs prises par les coordonnées des vecteurs des mots du corpus. Des distributions très peu étalées sont moins porteuses de sens. Ici, la comparaison entre 16 et 32 dimensions d'*embedding* ne permet pas vraiment de conclure à l'utilité ou à l'inutilité de certaines dimensions. Néanmoins, la concentration de termes techniques de notre corpus à vocabulaire limité entraîne l'occupation privilégiée d'un cadran de l'espace $\mathbb{R}^d$: on le voit par les décalages forts de la majorité du poids des distributions au dessus, ou au dessous de 0. On peut ainsi voir, comme l'a montré la section calculant l'entropie, que la dimension 32 est moins hétéroclite, donc plus redondante.

2.3 Utilisation des capacités d'analyse grammaticale de spacy pour l'extraction d'informations utiles dans le texte

2.3.1 Détection d'entités avec spacy

D'autre part, on a vu les différents outils de spacy dans l'état de l'art, l'une des fonctions proposées par les modèles de langages est la détection d'entités. Pour certaines questions précises, portant sur ces entités, on peut alors coupler spacy avec CamemBERT de la sorte. Pour l'exemple, reprenons toujours le même exemple, i.e. la première phrase du règlement de scolarité. Nous utiliserons le modèle `fr_core_news_md` qui est entraîné sur des corpus d'articles de presse principalement, selon la documentation de spacy. Cela a son importance, puisque les sujets et le vocabulaire spécifique du milieu de l'enseignement supérieur et plus particulièrement de l'école Centrale risquent de n'être pas bien capturés par les modèles de langages disponibles.

L'analyse des entités dans la phrase traitée

```
for ent in doc.ents:
    print(ent.text, ent.start_char, ent.end_char, ent.label_)
```

donne (et vous constatez quelques erreurs)

```
Règlement de scolarité 3 25 ORG
École Centrale de Lyon 65 87 ORG
École Centrale de Lyon 300 322 ORG
Master 341 347 MISC
Ingénieur Energie 375 392 PER
```

La détection de groupes nominaux (*chunking* et leur organisation avec spacy

spacy est capable de donner les groupes nominaux dans une phrase. C'est utile car cela simplifie l'extraction d'information. À chaque groupe nominal est associé une racine, de laquelle on peut partir en remontant ou descendant l'arbre syntaxique pour trouver des informations utiles sur les dépendances et chercher l'information qui nous plaît (voir suite).

```
for chunk in doc.noun_chunks:
    print('CHUNK:', chunk.text,
    'RACINE:', chunk.root.text,
    'HEAD:', chunk.root.head.text,
    'DEP:',  chunk.root.dep_)
```

donne

```
CHUNK: Le Règlement de scolarité RACINE: Règlement HEAD: présente DEP: nsubj
CHUNK: les modalités d'admission à l'École Centrale de Lyon RACINE: modalités
HEAD: présente DEP: obj
CHUNK: de l'évaluation des connaissances et des compétences de la formation
ingénieur RACINE: évaluation HEAD: modalités DEP: nmod
CHUNK: , les modalités de diversification de cette formation RACINE: modalités
HEAD: présente DEP: conj
CHUNK: et les conditions d'obtention des diplômes de l'École Centrale de Lyon,
hors diplômes de Master co-accrédités et diplôme d'Ingénieur Energie en Alternance
RACINE: conditions HEAD: présente DEP: conj
```

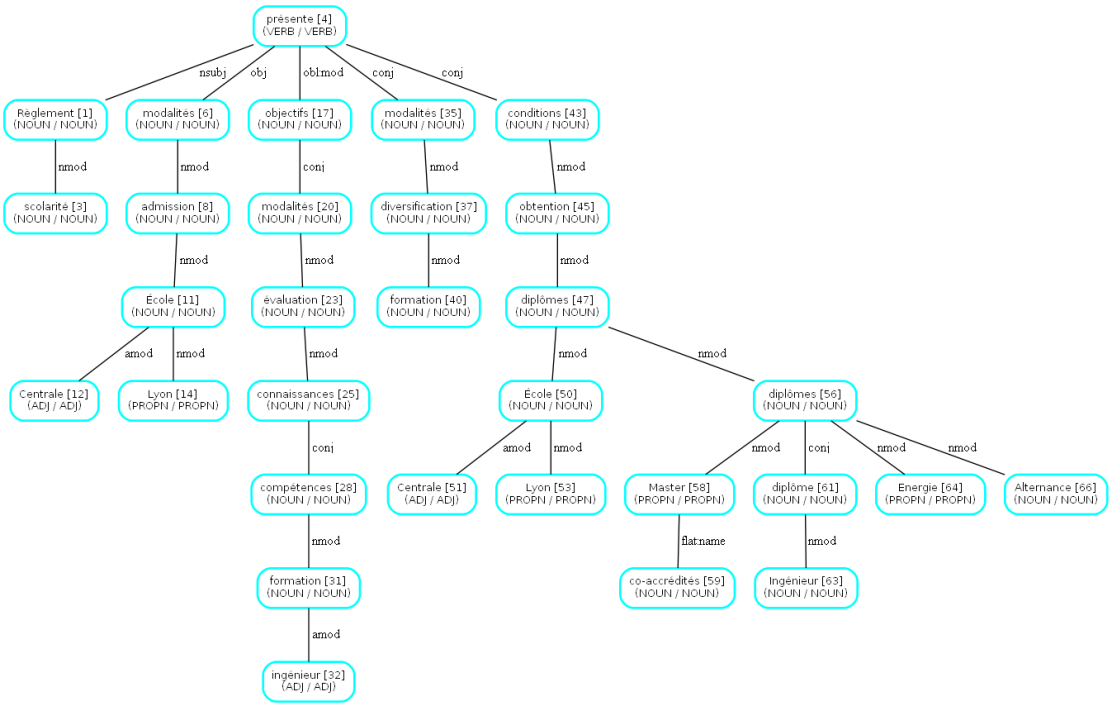


Figure 2.10 : Arbre syntaxique de la phrase épurée dont on a éliminé les stopwords et la ponctuation

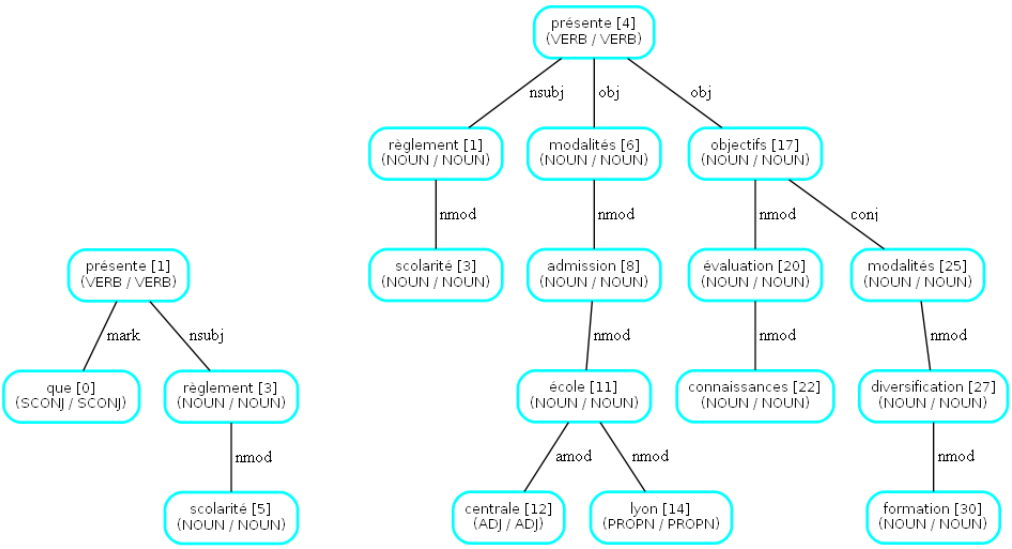
2.3.2 Construction d'arbres syntaxiques avec spacy

On a vu également que, en se donnant une phrase, spacy était capable d'analyser la syntaxe de la phrase grâce au modèle de langage défini. On peut en dresser l'arbre, mais il serait quasiment illisible pour une phrase complexe du règlement de scolarité, en raison de tous les déterminants, articles, stopwords, etc.

Après filtrage des mots de la phrase qui élimine les *stop-words* et la ponctuation (test `token.is_stop or token.is_punct`), on peut donc dresser l'arbre syntaxique de cette même phrase (voir figure 2.10) et remonter ainsi l'arbre syntaxique simplifié *via* les relations de parentalité entre les tokens. Vous remarquez que les tokens s'affichent avec leur numérotation d'origine. Néanmoins, il est préférable d'éliminer les tokens de *POS-part* de type `DET`, `ADP`, `PUNCT`, `CCONJ` dans un premier temps afin de ne pas éliminer les pronoms interrogatifs des questions comme nous allons le voir plus tard, par exemple. On fera simplement le test `token.pos_ not in ['ADP', 'DET', 'PUNCT', 'CCONJ']`.

Exemple de récupération sémantique sur une phrase du règlement de scolarité Prenons un exemple pour illustrer la démarche avec les points précédents. On va simplifier le problème et supposer un monde idéal où l'on dispose de la question « Que présente le règlement de scolarité ? » et la phrase contenant la réponse simplifiée « Le Règlement de scolarité présente les modalités d'admission à l'École Centrale de Lyon, les objectifs de l'évaluation des connaissances et les modalités de diversification de cette formation. ». Si, comme on le suppose, la réponse est formulée comme la question, on a bien une équivalence entre les relations de parentalité des tokens communs de la phrase (voir figure 2.11. Il s'agit alors de récupérer l'*objet* au sens de spacy du sujet.

On pourrait alors imaginer une approche de question-réponse en extrayant l'information à partir des caractéristiques linguistiques. En supposant que la question ne comporte qu'un



(a) question **(b)** réponse

Figure 2.11 : Arbres simplifiés de la question et de la réponse

groupe nominale sujet :

```
question = nlp("Que présente le règlement de scolarité ?".lower())

reponse = nlp("Le Règlement de scolarité présente les modalités
d'admission à l'École Centrale de Lyon, les objectifs de
l'évaluation des connaissances et les modalités de diversification
de cette formation.".lower())


chk_question = list(question.noun_chunks)[0]


# Afficher les résultats de la question
[
chunk.text for chunk in reponse.noun_chunks
if chunk.root.head.lemma_ == chk_question.root.head.lemma_
and chunk.root.dep_ in
['obj',
 'obl:agent',
 'obl:arg',
 'obl:mod']
 ]
```

qui donne la liste des objectifs suivante comme on l'attendrait

```
["les modalités d'admission à l'école centrale de lyon,",
 "les objectifs de l'évaluation des connaissances et les modalités de diversification
de cette formation"]
```

L'idée serait alors, pour chaque question, d'isoler les mots pertinents, grâce notamment à la reconnaissance par entités et la reconnaissance des groupes nominaux récurrents, de

dresser une liste de phrases donnant probablement l'information nécessaires dans le corpus et d'appliquer ce codes et des variations pour des types de questions récurrents (« qui fait ? », « que .... », etc).

2.3.3 Conclusion intermédiaire : l'approche sémantique

Cette approche possède plusieurs problèmes très bloquants :

- elle est entièrement **rationaliste** et très difficilement généralisable ;
- elle repose sur des hypothèses extrêmement fortes qui sont que la question et la réponse ne sont en somme que deux versions de la même phrase, l'une à l'interrogatif éventuellement sans complément et l'autre à l'indicatif. Elle est donc peu résiliente à des données incomplètes ou mal formulées ;
- elle repose sur la bonne syntaxe du langage, elle est donc peu robuste à des défauts dans le langage naturel côté question ou côté modèle de langage sur la réponse du corpus.

Nous ne l'avons donc pas retenue, ni en tant que telle pour concevoir un chatbot, ni comme élément intermédiaire de la chaîne du chatbot.

2.4 Conception d'un autoencodeur pour l'approche récupérative

Les approches principales permettant de coder un chatbot se font à l'aide de modèle *sequence-to-sequence* (Seq2Seq). Pour une approche générative, on apprend le modèle à « traduire » une question en une réponse à l'aide d'une base de données. Pour une méthode récupérative, le modèle cherche dans une base de données la réponse la plus adaptée à la question de l'utilisateur. Il répond donc avec les réponse prévue dans la base de données. Plusieurs approches ont été codées.

2.4.1 Le Seq2Seq auto-encodeur

Dans l'approche récupérative, le modèle doit chercher dans une base de donnée la question la plus proche de celle posée par l'utilisateur. Il faut donc se doter d'une métrique permettant de quantifier la proximité de deux phrases. Comme pour Word2Vec, on cherche à vectoriser des phrases. On le fait donc à l'aide du modèle Seq2Seq auto-encodeur. Un tel modèle est un modèle séquence a séquence entraîné à recopier la phrase d'entrée. On prend alors le vecteur de contexte comme vectorisation de la phrase d'entrée.

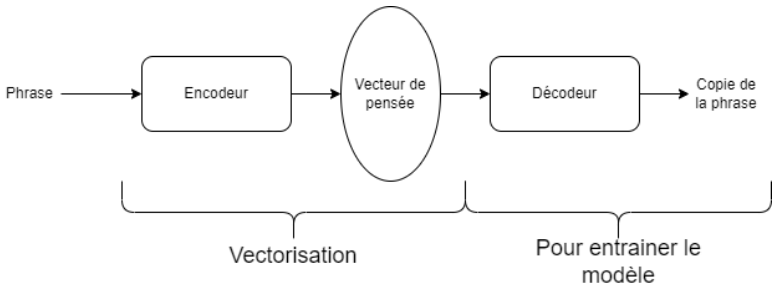


Figure 2.12 : Structure d'un modèle Seq2Seq auto-encodeur

2.4.2 Seq2seq adapté de la documentation Keras

Ce modèle a été implémenté en adaptant la structure générale d'un Seq2Seq sous Keras proposé dans la documentation (https://keras.io/examples/nlp/lstm_seq2seq/). Le modèle présenté permet de faire de la traduction anglais-français caractère par caractère. Il est aisé de transformer ce modèle en auto-encodeur en changeant les données d'entraînement. Il suffit de mettre la même phrase au lieu de la phrase traduite. Cependant, il est a priori plus judicieux de créer un modèle travaillant au niveau des mots et non au niveau des caractères. Un tel modèle a été créé en vectorisant au préalable les données d'entraînement.

Le modèle a été entraîné sur 1000 phrases, la taille du vecteur de pensée a été fixée à 512 et celle des vecteurs de mot à 126. La grandeur optimisée est la moyenne quadratique des erreurs.

Le modèle obtenu donne des résultats peu satisfaisants. Il n'est pas capable de copier de façon convenable une phrase donnée. La précision en fin d'apprentissage est très faible (moins de 10 %) est n'augmente pas significativement en entraînant le modèle sur 2000 cycles. De même, l'erreur quadratique moyenne n'est pas suffisamment optimisée pour que les phrases copiées puisse correspondre aux phrases initiales.

Il est difficile de savoir si ces résultats sont dus au modèle ou à d'autres paramètres. Néanmoins, une autre approche donne des résultats intéressants.

2.4.3 Seq2seq avec mécanisme d'attention pour le NMT adapté de la documentation TensorFlow

Nous avons repris l'exemple proposé par la documentation de TensorFlow (https://www.tensorflow.org/addons/tutorials/networks_seq2seq_nmt), qui crée un code en surchargeant `tf.keras.Model`. On s'applique à auto-encoder des phrases de français à français, nous avons pris pour la suite les 500 premières entrées de la liste des phrases en français données ici : http://www.manythings.org/anki/. Voici quelques-unes des phrases utilisées : (des numéros 440 à 462). Ce sont donc des phrases très courtes.

```
Demandez-leur.
Recule !
Reculez.
Cassez-vous.
Recule !
Reculez.
Retire-toi !
Cassez-vous.
Sois un homme !
Soyez un homme !
Soyez courageux !
Soyez bref.
Sois bref.
Sois brève.
Soyez brève.
Soyez brefs.
Soyez brèves.
Sois calme !
Soyez calme !
Soyez calmes !
Aucune idée.
```

L'intérêt est de pouvoir entraîner ce modèle sur des phrases issues du corpus de phrases du règlement de scolarité pour obtenir le **vecteur de contexte**. Le code a de particulier qu'il nécessite la librairie `tensorflow-addons` pour son mécanisme d'attention. Cela a mobilisé une partie du temps de travail pour installer cette bibliothèque en raison des nombreuses incompatibilités entre versions (Python, TensorFlow).

Description du modèle :

- deux classes `Encoder` et `Decoder` héritant de `tf.keras.Model` sont créées. Le décodeur comprend les têtes d'attention et l'encodeur fait un *one-hot-encoding*.

- des paramètres d'apprentissage sont créés tels que taille du *batch*, taille du *buffer*, taille des données en nombre de phrases, dimension pour l'*embedding*
- la fonction de perte à minimiser est prise de `tf.keras.losses.SparseCategoricalCrossentropy`
- une fonction d'encodage et de décodage de bout en bout est créée pour l'évaluation

Le modèle parvient à encoder correctement comme le montrent l'expérience et l'entropie croisée. Après 100 étapes on obtient une précision de 87% pour les paramètres suivants : dimension : 128, taille batch : 128, taille buffer : 32000. On considère qu'un encodage est réussi si la chaîne de caractères en sortie est la même que celle d'entrée (à la casse près).

Limitations Les limitations sont nombreuses avant de pouvoir adapter cela à notre phrases. Premièrement, le temps de calcul nécessaire, déjà long pour avoir ces scores avec 500 phrases très courtes. Secondement, la mémoire graphique nécessaire augmente avec la taille du *batch* et oblige à un plus grand temps de calcul (plusieurs étapes de batch par époque, qui peuvent dépasser plusieurs centaines). Enfin, nous n'avons de l'ordre de 600 phrases issues du règlement de scolarité qui sont de surcroît très longues.

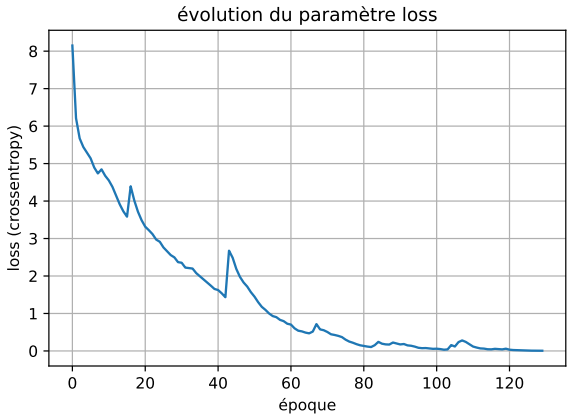


Figure 2.13 : Evolution du paramètre « loss crossentropy » pour le modèle de code TensorFlow utilisant l'attention

2.4.4 Seq2seq avec mécanisme d'attention et GRU

Le modèle précédent a été repris pour générer un auto-endodeur sur les questions récoltées. L'expérience a montré que les couches récurrentes GRU étaient plus performantes que le couches LSTM. Ce modèle a donc été muni de couches GRU. La taille du vecteur de pensée a été fixée à 250 et la taille des vecteurs de mot a été fixée à 50. Les figures 2.14 et 2.15 montrent les résultats de son apprentissage. La fonction de loss optimisée est la moyenne quadratique des erreurs.

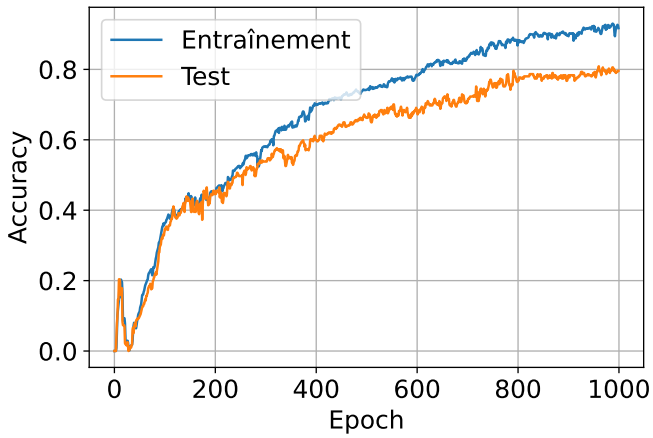


Figure 2.14 : Évolution de la précision du modèle autoencodeur avec *Gated Recurrent Units* (GRU) pendant l'apprentissage

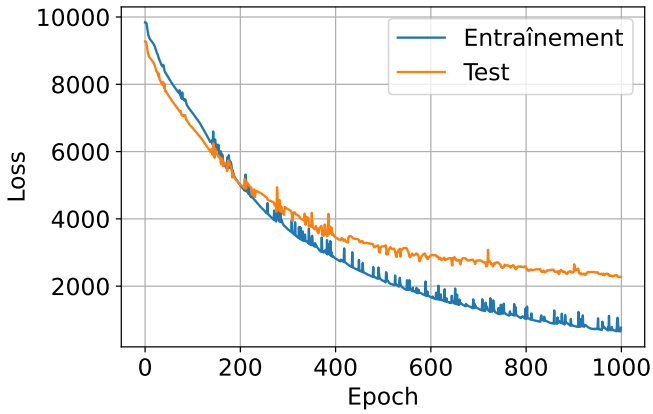


Figure 2.15 : Évolution de la fonction loss du modèle autoencodeur avec *Gated Recurrent Units* (GRU) pendant l'apprentissage

Ce modèle permet d'obtenir des résultats plutôt satisfaisants. La précision atteint 90% sur les données d'entraînement et 80% sur les données de tests. Aussi, la courbe sur les données de tests semble montrer que le réseau n'est pas dans un cas de sur-apprentissage. On s'attend donc à ce qu'il possède bien un caractère général. Nous avons donc décider de l'utiliser comme algorithme de Seq2Vec en ajoutant une fonction `vectoriser()` à la classe `Encoder`. Celle ci prend en entrée une question en langages naturelle, effectue toutes les transformations nécessaires (lemmatisation,...) pour permettre à l'encodeur de la lire. L'état interne de la couche récurrente est alors renvoyé, ce qui forme un vecteur de pensée.

2.4.5 Récupération de la réponse la plus adaptée dans la base de donnée

Une fois que l'on dispose d'un outil de vectorisation de phrases il faut être en mesure de trouver la réponse la plus adaptée à celle de l'utilisateur. Lorsque l'on dispose d'un ensemble de couples questions-réponses. Le problème se traduit alors comme un problème de classification. En effet, en indiçant les réponses par des entiers on peut voir le problème

sous la forme suivante : On connait $(Q, R) \in \mathbb{R}^N \times \mathbb{N}$ des couples questions-réponses. Pour tout $\mathscr{X} \in \mathbb{R}^N$, la question de l'utilisateur, on veut donner $\mathscr{Y}$, l'indice de la réponse la plus adaptée. Cela est le formalisme abstrait de tout problème de classification. Il existe plusieurs façons de le résoudre. Une approche classique pour résoudre ce type de problèmes est l'algorithme des k plus proches voisins.

L'algorithme des k plus proches voisins (k-nearest neighbours : k-NN)

Cette solution consiste à associer à $\mathscr{X}$, le choix majoritaire de ses k-plus proches voisins dans l'ensemble des questions connues. Le nombre k est un paramètre à optimiser pour chaque problème. En effet, si k= 1, on classifie par le choix du plus proche voisin et si k est plus grand que le nombre total de point alors tout nouveau point sera classifié comme faisant partie de la classe majoritaire. La figure 2.16 montre bien l'importance de k pour ce genre d'approche.

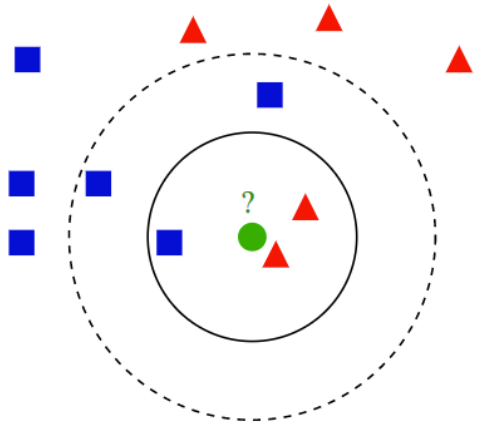


Figure 2.16 : Un exemple de classification binaire avec k-NN (Wikipédia)

Pour trouver le meilleur k à notre problème, nous avons utilisé une technique de cross-validation. La *cross-validation*, ou validation croisé, en apprentissage automatique, est une méthode qui consiste à séparer le jeu de données en deux ensembles (train et *test*), à ratio fixé. On sélectionne une partie (en général, autour de 80%) qui sert à l'entraînement, et on teste les résultats sur l'autre partie (les 20% restants). La cross-validation effectue une rotation des données données d'entraînement et validation à chaque étape d'évolution des gradients.

La figure 2.17 montre les résultats de cette méthode sur l'hyperparamètre k et nos données obtenues par le modèle d'autoencodeur avec GRU. La conclusion de ce test est que le meilleur score est obtenu pour $k = 1$.

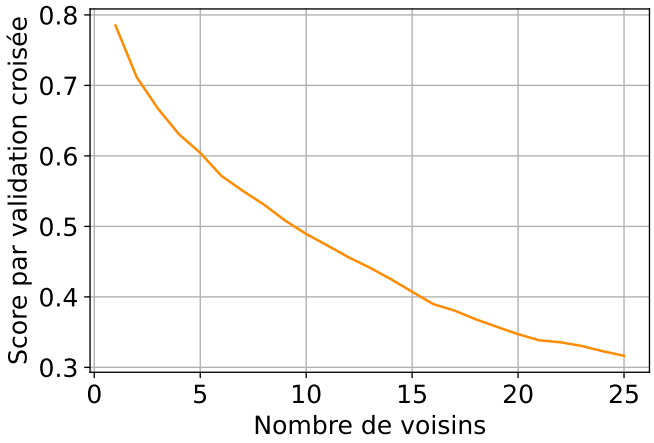


Figure 2.17 : Optimisation du paramètre k pour l'algorithme k-NN

2.4.6 Conclusion intermédiaire : algorithme k-NN

En testant cette approche sur les 1570 questions d'entraînement, le chatbot ainsi formé donne la bonne réponse dans seulement 98% des cas. Cependant, pour des questions non connues, le chatbot n'est pas performant. Par exemple, la question : « Qui approuve le règlement de scolarité ? » fait partie des données d'entraînement mais la question « Qui approuve le règlement intérieur ? » n'en fait pas partie. Le chatbot répond à cette dernière par « Une UE déjà validée ne peut pas être refaite. ». La réponse renvoyée n'est pas adaptée à la question de l'utilisateur.

2.5 Conception d'un *Retriever* basé sur l'architecture QAnet

2.5.1 Le modèle QAnet

Il s'agit d'un modèle proposé dans la catégorie « compréhension écrite », dans l'article [19] intitulé *Combining Local Convolution with Global Self-Attention for Reading Comprehension.*

Dans cette section, nous détaillons notre appropriation et reprogrammation simplifiée du modèle QAnet, qui conçoit un réseau de neurones capable de répondre à des questions sur un texte appelé contexte, comprenant les réponses [19], en indiquant le passage du texte où se situe la réponse (d'où l'appelation *retriever* pour « récupérer »). Nos simplifications résident dans le fait que nous n'avons pas répété les empilement de couches de convolutions et attention répétée dans les blocs encodeurs (voir descriptions qui suivent). Concrètement, nous avons recréé un réseau qui se rapproche de l'architecture QAnet à partir de l'article et du diagramme seulement (voir figure 2.18). Les paramètres d'entraînement, la fonction de *loss*, et l'arrangement précis des couches de neurones n'ont pas suivi à la lettre l'article. Nous avons ensuite essayé de comparer les performances de variantes de l'architecture. Expliquons brièvement d'abord le fonctionnement du modèle et ses principaux composants.

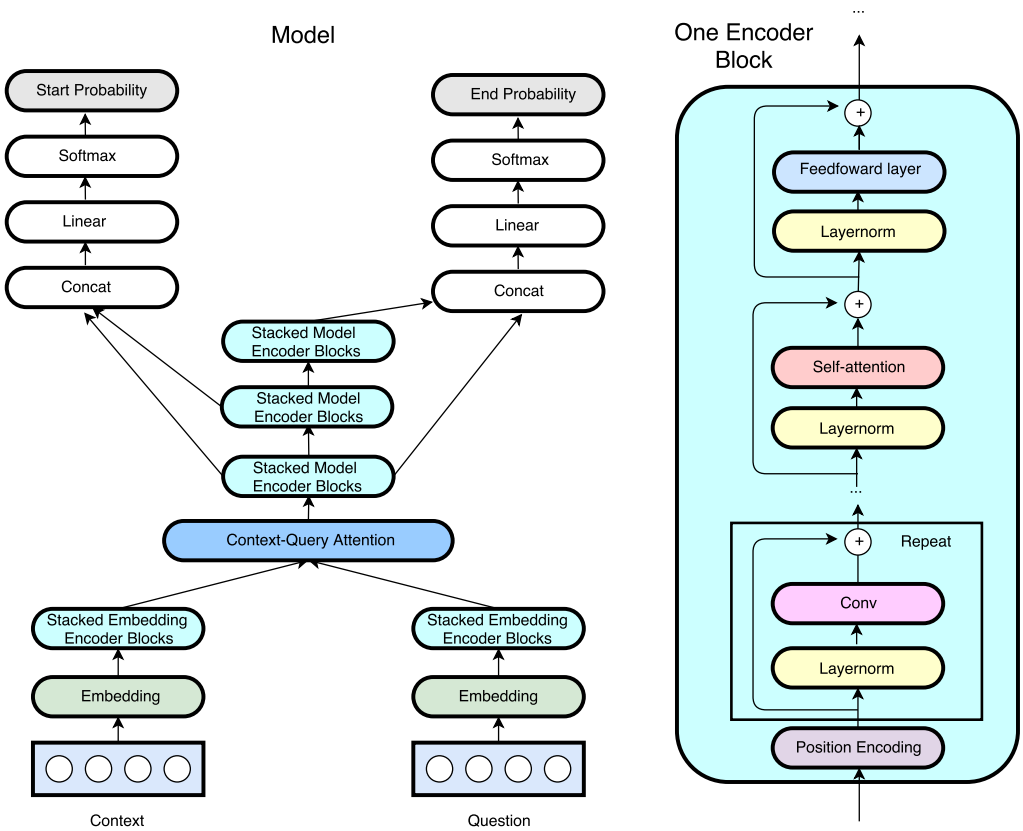


Figure 2.18 : Modèle QAnet (schéma issu de [19])

2.5.2 L'architecture ResNet

Le modèle QAnet est assez imposant comme réseau de neurones (voir les sections suivantes). Il exploite amplement un principe d'architecture appelé *Residual Networks*. Le problème que résout cette architecture, est que dans les gros réseaux, les couches successives n'apprennent pas simultanément, mais plutôt de façon graduelle en partant de la fin. Pour réduire le temps d'apprentissage, on crée une liaison entre couches de neurones qui en contourne d'autres. Concrètement, le résultat r_n après la couche h_n est le résultat $r_n = h_n(r_{n-1}) + r_{n-1}$ Chaque unité h_n est appelée *unité résiduelle*. Les unités résiduelles apparaissent dans l'architecture de l'`EncoderBlock` (figure 2.20). Cela a un avantage lorsque les données d'une couche à l'autre ne sont « pas beaucoup » transformées : chaque couche apprend alors le *résidu*, c'est-à-dire la différence avec l'identité. Pour le traitement de texte, les auteurs de l'architecture QAnet se sont basées sur des recherches précédentes, mais vu que notre réseau ne cherche qu'à *retrouver* l'information dans le texte, on peut faire l'hypothèse que les blocs encodeurs ne transforment pas radicalement l'information, mais plutôt cherche à trouver les relations sémantiques qui lie les informations des mots entre eux (notamment avec la convolution et l'attention, pour mesurer les similarités du texte avec lui-même). D'où les blocs résiduels dans chaque bloc encodeur.

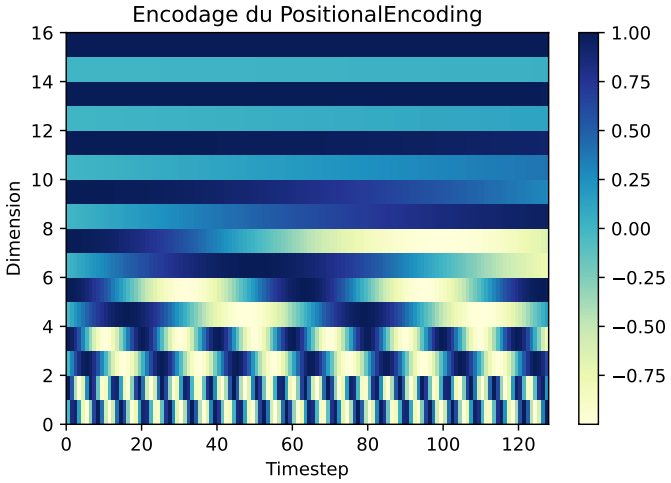


Figure 2.19 : Fonction du positional encoding

2.5.3 Positional Encoding

Le *Positional Encoding* est une technique permettant de modifier les données d'entrées d'un réseau de neurones pour permettre d'encoder comme son nom l'indique, les relations temporelles des éléments d'une série temporelle. La technique apparaît pour la première fois dans l'architecture Transformers sur l'attention [18].

L'intuition des auteurs derrière ce concept, était que cette méthode permettrait au modèle d'apprendre les positions relatives des différents vecteurs, étant donné que l'encodage à une position dans le temps est une fonction linéaire de l'encodage à un pas de temps différent. Le *positional encoding* est une fonction de deux variables discrètes (indice temporel t et indice des *features* r) telle que :

$$PE(r,t) = \begin{cases} \sin\left(\frac{t}{10^{4\times\frac{2k}{d}}}\right) & \text{si } t = 2k \\ \cos\left(\frac{t}{10^{4\times\frac{2k}{d}}}\right) & \text{si } t = 2k+1 \end{cases} \tag{2.6}$$

Cette couche n'existe pas de manière native dans TensorFlow mais on peut facilement la créer en surchargeant la classe `tf.keras.layers.Layer`. Nous avons pris le parti de reprendre le code de François Chollet (le fondateur de la librairie `keras`) proposé dans son ouvrage *Hands-On Machine Learning with Sci-kit Learn, Keras and TensorFlow* (voir code 1). Suivant ce code, nous avons tracé en figure 2.19, les valeurs prises par la fonction sur un ensemble de valeurs discrètes pour une dimension d'encodage de 16 et un pas de temps maximal de 128.

2.5.4 Classe `EncoderBlock`

Le bloc de base constituant le modèle QAnet est appelé *encoder block*. Vous pouvez regarder le diagramme donnant l'architecture du bloc en figure 2.20. Il y en a 5 dans le modèle QAnet. Un bloc se compose de blocs de convolution empilés et entrecoupés de normalisations, suivis de têtes d'auto-attention et de couches denses.

La fonction MultiHeadAttention Rappelons succintement ce qu'est une couche de `MultiHeadAttention` utilisée en auto-attention : On note d la dimension de l'espace latent

```python
class PositionalEncoding(tf.keras.layers.Layer): # dans Hands-On Machine Learning for Python
    def __init__(self, sizes, dtype=tf.float32, **kwargs):
        super().__init__(dtype=dtype, **kwargs)
        max_steps = sizes[0]
        max_dims = sizes[1]
        if max_dims % 2 == 1: max_dims += 1 # max_dims must be even
        p, i = np.meshgrid(np.arange(max_steps), np.arange(max_dims // 2))
        pos_emb = np.empty((1, max_steps, max_dims))
        pos_emb[0, :, ::2] = np.sin(p / 10000**(2 * i / max_dims)).T
        pos_emb[0, :, 1::2] = np.cos(p / 10000**(2 * i / max_dims)).T
        self.positional_embedding = tf.constant(pos_emb.astype(self.dtype))
    def call(self, inputs):
        shape = tf.shape(inputs)
        return inputs + self.positional_embedding[:, :shape[-2], :shape[-1]]
```

Listing 1 : Classe `PositionalEncoding` de François Chollet

(ou le nombre de *features* du modèle d'entrée : en ce qui nous concerne, c'est la dimension choisir pour chaque vecteur du Word2Vec). On note n le nombre d'éléments d'une requête (le nombre de pas de temps pour une séquence temporelle).

L'attention multi-tête prend en entrée trois éléments : *query* $Q \in \mathscr{M}_{n_Q,d}(\mathbb{R})$, *key* $K \in \mathscr{M}_{n_K,d}(\mathbb{R})$, *value* $V \in \mathscr{M}_{n_K,d}(\mathbb{R})$, et retourne le vecteur

$$\text{Attention}(K,Q,V) = (\text{softmax}\left(\frac{QK^\intercal}{\sqrt{d}}\right)V \tag{2.7}$$

Dans le cas de l'auto-attention, toutes les matrices sont égales et valent la phrase d'entrée, que l'on compare avec elle-même : c'est une méthode bien connue. Dans les tests suivants, nous avons choisi 2 têtes d'attention dans chaque `EncoderBlock` car c'était la valeur qui donnait les meilleurs résultats expérimentaux.

2.5.5 Choix de la fonction Loss

Cross-categorical entropy

Les vecteurs retournés en sortie du modèle étant des distributions de probabilité pour les positions de début et fin de la réponse, il est naturel de prendre la fonction d'entropie croisée (*cross-categorical entropy*).

La fonction « loss » de *cross-categorical entropy* est l'entropie croisée de Shannon des suites (finies) des microétats de la variable aléatoire des données d'entraînement et de celle donnée par le réseau :

$$l(y^{\text{pred}}, y^{\text{true}}) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{d} y_{i,d}^{\text{true}} \log y_{i,d}^{\text{pred}} \tag{2.8}$$

où $y_{i,d}^{\text{pred}} = 1$ si la donnée d'indice i est catégorisée d et 0 sinon. Plus généralement, l'entropie croisée entre deux lois de probabilité mesure le nombre de bits moyen nécessaires pour discriminer les deux événements. La fonction `tf.keras.losses.CategoricalCrossentropy` qui sert de fonction de perte, est minimale lorsque les deux distributions sont égales et vaut alors l'entropie de y^{true} (la divergence de Kullback-Leibler associée aux deux distributions est alors nulle).

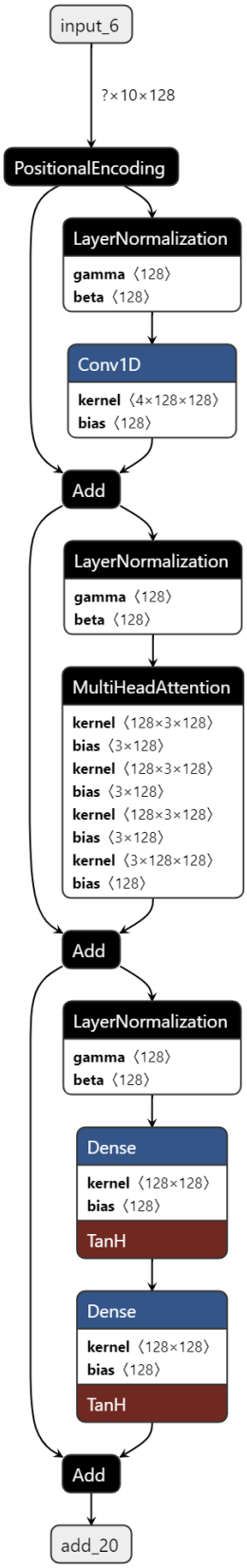


Figure 2.20 : Diagramme du Layer `EncoderBlock` pour des séquences de longueur 10 sur une dimension de 128 pour l'embedding

Root-Mean-Squared Error

Elle s'écrit

$$l(y^{\text{pred}}, y^{\text{true}}) = \frac{1}{n}\sum_{i=1}^{n} \|y^{\text{pred}} - y^{\text{true}}\|_2^2 \tag{2.9}$$

La fonction d'erreur quadratique sanctionne les différences cumulées entre les composantes de vecteurs. C'est leur norme au carré. Le problème de cette fonction, appliqué à deux vecteurs de probabilités, est qu'elle ne rend pas compte de l'éloignement entre deux pics de la distribution.
On choisira donc l'entropie croisée.

2.5.6 Données d'entraînement

Base de données SQuAD Pour entraîner QAnet, nous avons d'abord cherché à utiliser la base de données SQuAD, qui comprend pour son *dataset* plus de 80000 4-uplets question, contexte, indices de début et fin de la réponse en langue anglaise pour un vocabulaire d'environ 54000. Nous avons assez rapidement fait face à des limites de nos ordinateurs pour entraîner ce modèle (les textes du contexte excédaient plusieurs centaines de *tokens*), en effet, pour donner un ordre de grandeur le modèle, dont le nombre de paramètres s'élevait à plusieurs millions, s'adaptait aux données passées en entrée[2] et en en complexité mémoire pour la langue anglaise et une taille normale de 64 dimensions : 80000 taille des données $\times$ [30 tokens (question) + 500 tokens (contexte)] $\times$ 64 dimension embedding $\times$ 4 octets (type `tf.float32`) $\approx$ 10 giga octets. Ces tenseurs sont stockés dans la mémoire RAM lors de l'entraînement. Il est facilement compréhensible pourquoi on atteint là les limites des ordinateurs portables moyens. Après avoir réduit la taille du set d'entraînement et la dimension de l'embedding, mais sans réel succès sur les valeurs de précision de prédiction du modèle, faute d'entraînement et de données suffsantes (en se limitant à 10000 couples par exemple). La preuve du concept n'a donc pas abouti sur la base SQuAD.

Nouvelles données avec le règlement de scolarité Nous avons ensuite cherché à créer un dataset à partir du règlement de scolarité en reprenant le travail détaillé en section 2.1.5. Nous avons utilisé `pandas` pour la gestion des données. En plus de cela, nous avons fabriqué des questions à partir de paragraphes des documents de scolarité (le règlement notamment). Une fois la réponse dupliquée, qui est une sous-chaîne de caractères de la chaîne de contexte qui respecte la condition de coupure hors des mots, nous avons tokénisé les deux chaînes obtenues. Sous l'hypothèse que la tokénisation de `spacy` ne dépend pas du contexte de la phrase (i.e., est invariante par augmentation ou réduction de la phrase), ce qui s'est vérifié expérimentalement, nous obtenons deux listes de tokens, dont l'une (la réponse) est une sous-suite de l'autre (le contexte). À l'aide d'un automate de recherche créé par l'algorithme Knuth-Moriss-Pratt, nous obtenons ainsi l'indice de commencement de la réponse dans le contexte (en tant que listes de tokens). Grâce à la longueur de la réponse on retrouve aussi l'indice de fin. Ces indices sont les données de sortie du réseau QAnet.

Cette méthode a permis de constituer une centaines de questions/réponses sur lesquelles entraîner QAnet. En voici les 10 premières :

Le résumé du modèle final est présenté ci-dessous : la taille est raisonnable (voir résumé ci-dessous).

[2] Plus les phrases étaient longues, plus les tenseurs d'entrée étaient grands, plus les couches de type Dense comprenaient de paramètres.

question	contexte
Qui approuve le règlement de scolarité ?	Le Règlement de Scolarité est approuvé par le C...
Par quelle voie s'effectue l'admission en premi...	L'admission en 1ère année de la formation ingén...
Comment est l'inscription pour les élèves en pr...	Pour les élèves ayant signé un contrat de profe...
Que les élèves doivent-ils avoir lu et approuvé ?	Les élèves ont lu, approuvé et signé les charte...
Pour quoi la formation peut-elle être aménagée ?	La formation peut être aménagée pour effectuer ...
Qu'est-ce qu'une action de formation ?	Une Action de Formation est un ensemble cohéren...

Table 2.2 : 6 premières phrases du dataset constitué avec les documents de la scolarité

```
Model: "qa_retriever_1"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 encoder_block_5 (EncoderBlock)  multiple                  13792
 encoder_block_6 (EncoderBlock)  multiple                  13792
 encoder_block_7 (EncoderBlock)  multiple                  13792
 encoder_block_8 (EncoderBlock)  multiple                  13792
 encoder_block_9 (EncoderBlock)  multiple                  13792
 embedding_1     (Embedding)     multiple                  28736
 masking_1       (Masking)       multiple                  0
 attention_1     (Attention)     multiple                  0
 concatenate_1   (Concatenate)   multiple                  0
 dense_14 (Dense)                multiple                  33
 dense_15 (Dense)                multiple                  33
 dense_16 (Dense)                multiple                  68635
 dense_17 (Dense)                multiple                  68635
=================================================================
Total params: 235,032
Trainable params: 235,032
Non-trainable params: 0
_________________________________________________________________
```

2.5.7 Entraînement

Le modèle a été entraîné avec l'optimiseur ADAM et ses paramètres par défaut, un *validation split* égal à 20% et 100 itérations. Le *word embedding* est produit par la couche `tf.keras.layers.Embedding`, sur 32 dimensions. Les résultats de l'entraînement avec courbes d'apprentissage sont donnés en figure 2.21. Malheureusement, les résultats sont loin d'être bons. Un dataset de seulement une centaines de couples question/réponses n'a pas suffit à entièrement faire comprendre les relations sémantiques entre les questions et les réponses. Néanmoins le modèle parvient à être meilleur qu'un simple choix au hasard des éléments dans le contexte. On peut le voir en lui proposant des phrases inconnues des ensembles d'entraînement et de test. Nous prenons ici l'exemple du contexte :

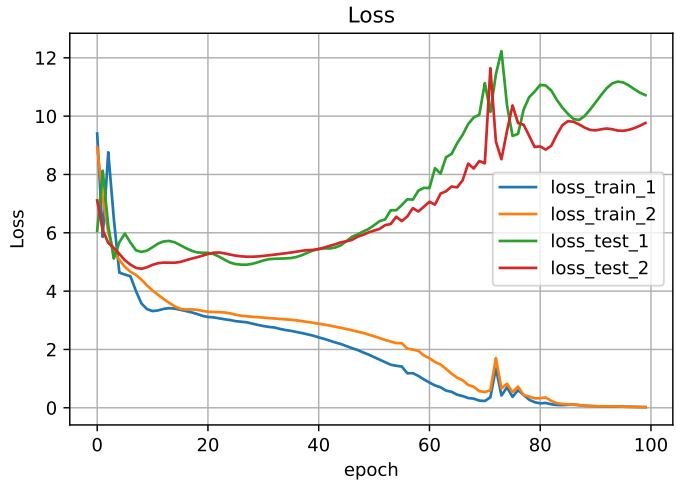
```
L'élève doit passer l'examen de seconde session si le directeur le décide
pour valider sa mobilité.
```

qui n'a aucun sens au regard de la scolarité à Centrale Lyon mais dont la grammaire est bien correcte et dont l'information est présente pour répondre correctement. Et la question :

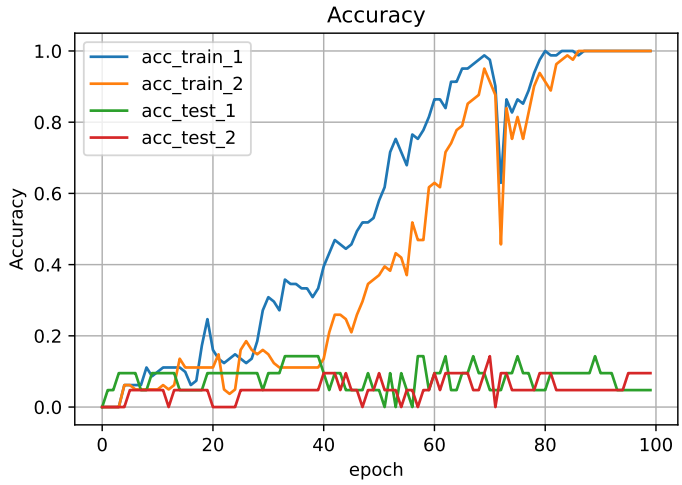
```
Qui doit passer un examen de seconde session ?
```

Notre modèle prédit la chose suivante (figure 2.22) :

```
l' examen de seconde session
```



(a) loss



(b) accuracy

Figure 2.21 : Courbes d'entraînement de QAnet sur les données de la scolarité

Autrement dit, le modèle a confondu sujet et complément. Mais la réponse n'est absurde. En figure 2.22, vous pouvez voir la distribution de probabilité associée au contexte pour le début et la fin de réponse. On peut déjà s'apercevoir que le résultat n'est pas aléatoire et que les distributions semblent isoler un groupe nominal.

L'ajout de couches `tf.keras.layers.Dropout` entre chaque `EncoderBlock` avec des taux d'élimination entre 3 et 5% permet de limiter le sur-apprentissage, mais au prix d'une précision encore moindre, ce qui n'est pas non plus satisfaisant.

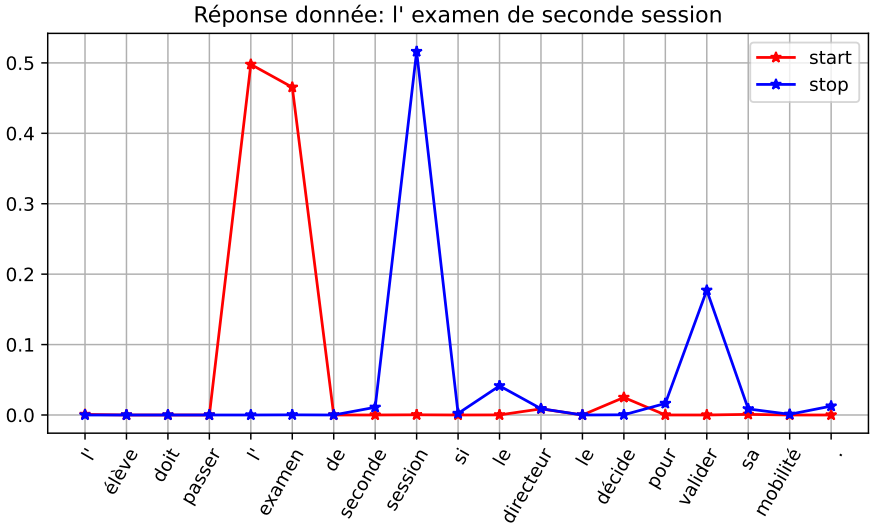


Figure 2.22 : Probabilités associée à la prédiction pour le contexte « L'élève doit passer l'examen de seconde session si le directeur le décide pour valider sa mobilité. » et la question « Qui doit passer un examen de seconde session ? ». La distribution a été coupée à la longueur du contexte, et les valeurs suivantes dans la série sont donc considérées comme non porteuses de sens.

2.5.8 Conclusion intermédiaire : QAnet

L'architecture QAnet a déjà fait ses preuves. Les tailles de données typiques pour lesquelles elle est prévue sont supérieures de plusieurs ordres de grandeur (au moins 3) à nos données (100 questions/réponses, contre proche de 10^5 pour SQuAD). Les courbes d'apprentissage que nous obtenons sont caractéristiques d'un sur-apprentissage par le réseau de neurones (*overfitting*). Nous concluons sans peine que le manque conséquent de données ne permet pas de conclure précisément sur la précision de QAnet, mais un meilleur entraînement serait un excellent atout dans la conception de l'approche récupérative (section 1.3.1).

2.6 Gestion de projet

2.6.1 Tâches réalisées

Globalement, la plupart des tâches qui devaient être réalisées à jour l'ont été. Certaines se sont même terminées en avance. Par exemple, la tâche « Entraîner le Word-Embedding sur les docs sélectionnés Skip-gram, CBOW » a été finie avec une avance de quatre semaines, de même que la tâche « pré-traiter les données textuelles ». La tâche « Dresser la liste des questions réponses » a été modifiée, car elle a pris plus de temps que prévu, notamment au niveau de la manipulation des données et de leur collecte auprès des étudiants.

Dresser la liste des questions réponses Cette tâche consistait à générer un ensemble de questions réponses en parcourant les documents de la scolarité. Cependant, de nombreuses questions sont posées par les étudiants lors des AGs d'information ou par d'autres moyens. Après discussion avec des membres de l'administration, il a été possible de mettre en place une méthode pour récupérer ces questions ainsi que les réponses formulées, mais pas de récupérer les réponses auprès de la scolarité. Continuer cette tâche dans des projets futurs permettrait d'obtenir à terme des données de plus grande qualité.

2.6.2 Outils de gestion de projet et GANTT

Au cours de ce PAr, les outils classiques de gestion de projet ont été exploités. Le diagramme GANTT de janvier 2022 est représenté en figure 2.23 et représente l'état de l'avancement du projet sous la forme d'un ensemble de tâches annotées.

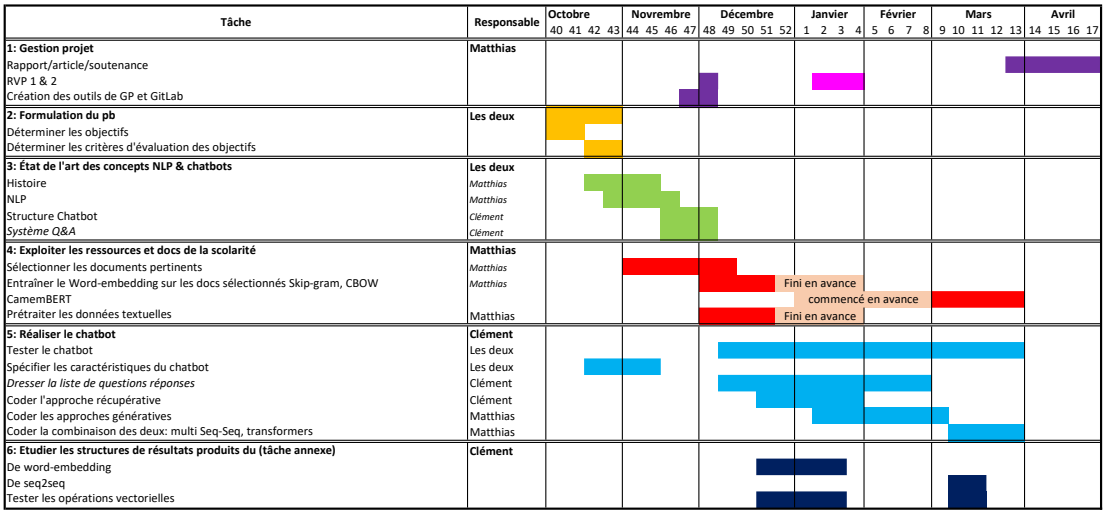


Figure 2.23 : Gantt Janvier 2022

2.6.3 Difficultés d'avancement au cours du PAr

Le projet semble avoir avancé au rythme attendu. Les différentes difficultés rencontrées, mise en place d'un environnement pour TensorFlow, difficultés de prise en main des différentes librairies, ont pu être résolues dans les temps avec néanmoins de forts blocages au début.

Chapitre 3

Conclusion générale

Le sujet de ce Projet d'Application Recherche portait majoritairement sur le traitement du langage naturel appliqué à la conception d'un *chatbot*. Nous avons fait le choix de reprendre le problème de zéro et reconstituer les éléments d'une approche récupérative qui constitueraient un chatbot élémentaire. Nos résultats sont mitigés : les approches classiques de représentation des mots ont donné d'excellents résultats avec la sémantique de notre corpus suffisamment bien captée, mais le manque de données ne nous a pas permis de tester en largeur l'efficacité du modèle QAnet.

Néanmoins, Le travail effectué reste relativement étendu dans les méthodes et les outils employés au regard des techniques classiques en Machine Learning, et il constitue une base solide de travail pour les travaux suivants qui voudraient éventuellement persister dans une approche récupérative, et poursuivre ensuite vers d'autres techniques plus évoluées (approche générative) ou ajouter des jalons et améliorer nos réseaux.

Bibliographie

[1] E. Adamopoulou and L. Moussiades. An Overview of Chatbot Technology. In I. Maglogiannis, L. Iliadis, and E. Pimenidis, editors, *Artificial Intelligence Applications and Innovations*, IFIP Advances in Information and Communication Technology, pages 373–383, Cham, 2020. Springer International Publishing.

[2] D. Bahdanau, K. Cho, and Y. Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. *arXiv :1409.0473 [cs, stat]*, May 2016. arXiv : 1409.0473.

[3] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin. A Neural Probabilistic Language Model. 2008.

[4] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *arXiv :1412.3555 [cs]*, Dec. 2014. arXiv : 1412.3555.

[5] R. Collobert and J. Weston. A Unified Architecture for Natural Language Processing : Deep Neural Networks with Multitask Learning.

[6] L. Deng and Y. Liu. *Deep Learning in Natural Language Processing*. Springer Singapore, May 2018.

[7] P. Goyal, S. Pandey, and K. Jain. *Deep Learning for Natural Language Processing : Creating Neural Networks with Python*. 2018.

[8] A. Géron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow : Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, Inc, USA, Beijing ; Boston, Mar. 2017.

[9] B. Jang, I. Kim, and J. W. Kim. Word2vec convolutional neural networks for classification of news articles and tweets. *PLOS ONE*, 14(8) :e0220976, Aug. 2019. Publisher : Public Library of Science.

[10] A. Kulkarni and A. Shivananda. *Natural Language Processing Recipes Unlocking Text Data with Machine Learning and Deep Learning Using Python*. 2021.

[11] L. Martin, B. Muller, P. J. O. Suárez, Y. Dupont, L. Romary, V. de la Clergerie, D. Seddah, and B. Sagot. CamemBERT : a Tasty French Language Model. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7203–7219, 2020. arXiv : 1911.03894.

[12] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient Estimation of Word Representations in Vector Space. *arXiv :1301.3781 [cs]*, Sept. 2013. arXiv : 1301.3781.

[13] S. Raj. *Building Chatbots with Python : Using Natural Language Processing and Machine Learning*. Apress, Dec. 2018.

[14] B. R. Ranoliya, N. Raghuwanshi, and S. Singh. Chatbot for university related FAQs. In *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pages 1525–1530, Sept. 2017.

[15] S. Russell. *Artificial Intelligence : A Modern Approach*. Pearson, 2020.

[16] S. Russell and P. Norvig. *Artificial Intelligence : A Modern Approach, eBook, Global Edition*. Pearson Higher Ed, Apr. 2021.

[17] A. Singh, K. Ramasubramanian, and S. Shivam. *Building an enterprise chatbot : work with protected enterprise data using open source frameworks*. 2019. OCLC : 1120770810.

[18] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention Is All You Need. *arXiv :1706.03762 [cs]*, Dec. 2017. arXiv : 1706.03762.

[19] A. W. Yu, D. Dohan, M.-T. Luong, R. Zhao, K. Chen, M. Norouzi, and Q. V. Le. QANet : Combining Local Convolution with Global Self-Attention for Reading Comprehension. *arXiv :1804.09541 [cs]*, Apr. 2018. arXiv : 1804.09541.