Génération de phrases sensées selon des structures syntaxiques

Thomas Blondelle
PAr 142
Laboratoire LIRIS
École Centrale de Lyon
4 avril 2017

Résumé

La recherche sur le traitement des langues naturelles est aujourd'hui en plein essor grâce à des techniques de traitement performantes et des applications pratiques omniprésentes. La génération de texte en particulier attire les regards car elle constitue une part fondamentale de l'intelligence artificielle. Les recherches dans ce domaine utilisent des méthodes élaborées pour discerner à la fois des motifs syntaxiques et des cheminements sémantiques à travers de très nombreux exemples de texte. Cependant les règles formelles de construction de phrases ne sont pas toujours respectées par cet apprentissage inductif. Nous proposons dans ce rapport une méthode qui utilise les aspects créatifs de la précédente technique tout en respectant un cadre syntaxique rigide. Nous montrons que des réseaux de neurones simples peuvent déjà mener à des résultats très satisfaisants. Pour augmenter les performances du système, cependant, d'autres types de réseaux sont nécessaires.

1 Introduction

Le traitement des langues naturelles (*Natural Language Processing* ou NLP) est un domaine dynamique où les applications sont de plus en plus nombreuses, les exemples les plus connus étant Siri, Cortana ou Google Assistant. D'anciens problèmes comme la traduction automatique d'une langue à une autre ou la correction orthographique nécessitent l'utilisation de méthodes de NLP. L'émergence de machines dotées d'une grande puissance de calcul et de nouvelles théories ont récemment amélioré la performance de ces outils. Elle a également ouvert d'autres horizons : il est possible de créer des résumés automatiques de document sans intervention humaine. Ces applications restent cependant des outils d'analyse. Pour s'approcher d'une éventuelle intelligence artificielle, les outils d'analyse, aussi complexes qu'ils soient, ne peuvent suffire. Il faut en effet prendre aussi en compte le facteur "création". Nous étudierons ici ce facteur à travers la génération de texte (*Natural Language Generation* ou NLG). La génération de texte étant un domaine très vaste, nous nous limiterons à une génération soumise à des contraintes précises détaillées ci-dessous.

L'objectif de ce projet est de créer un programme informatique permettant de renvoyer une suite de trois phrases, cohérentes individuellement et collectivement, à partir des données suivantes : trois noms et un verbe. La première phrase utilise deux noms et le verbe donnés en entrée : elle permet de fixer le sujet de la conversation. La deuxième phrase doit être inventée par le programme sachant que les trois phrases doivent rester cohérentes. La troisième phrase doit aussi être inventée mais doit comporter le nom donné en entrée (voir l'exemple en figure 1). Les structures syntaxiques choisies pour les trois phrases seront choisies au hasard parmi des structures existantes. Nous travaillerons en langue anglaise.

2 État de l'art

Dans cet état de l'art, nous faisons une rapide mise en contexte historique du sujet. Puis nous parlons des applications existantes ainsi que les défis qui restent à résoudre. Nous discutons

It **seems** to us **merchants** that **fortune** was smiling on us.
But then, one day, after a particurlarly long voyage, the captain came up onto the deck, crying out :
"Oh Alas ! For my children will be **orphans** !"

FIGURE 1 – Exemple de résultat désiré. Le texte en gras est donné en entrée. Le reste du texte doit être inventé par l'algorithme.

ensuite de quelques éléments de modélisation du langage ainsi que quelques méthodes importantes en Deep Learning. Nous terminons par les recherches en cours sur le traitement des langues naturelles.

2.1 Une brève introduction historique au traitement des langues naturelles

Le traitement de langues naturelles (ou NLP) est une branche très vaste de la linguistique utilisée en informatique. Les recherches dans ce domaine consistent à inculquer à une machine le savoir typiquement humain de la communication textuelle. Ce domaine est d'une grande complexité car il implique que la machine soit capable de comprendre un message, de posséder une culture et de produire un discours. La NLP est aussi fortement liée au domaine de l'intelligence artificielle : en effet, un système capable d'échanger et de retenir l'information schématise à grand traits une intelligence.

Cela explique l'engouement ancien pour ce domaine qui est apparu à la même époque que les ordinateurs [8]. En 1950, Alan Turing propose le test de Turing [16] comme critère d'intelligence. Il s'agit, pour un programme informatique de personnifier un humain dans une conversation écrite en temps réel de manière suffisamment convaincante pour qu'une personne ne puisse distinguer cette conversation d'une conversation entre humains. Par ailleurs, la NLP s'est largement développée dans les années 1960 [8] pour construire un système pouvant traduire automatiquement n'importe quel texte. Cependant la tâche s'est révélée plus difficile que prévu. En 1974, le gouvernement américain coupa les fonds. Pendant deux décennies, la recherche sur les technologies de langage avance à petits pas [1]. De nombreuses théories sont restées au point mort à cause de l'approche basée sur l'apprentissage de règles grammaticales formelles (*rule-based system*) : le langage est parsemé d'ambiguïtés et d'exceptions. Les systèmes basés sur ce concept sont alors trop complexes et échouent à leur tâche. La deuxième approche est basée sur la mémorisation de nombreux exemples (*data-driven system*) et l'apprentissage de motifs souvent répétés.

Aujourd'hui, grâce à la puissance croissante des processeurs, des coûts de stockage de données en chute libre et surtout de l'explosion des données disponibles, cette approche a finalement porté ses fruits. La dernière percée concerne l'arrivée du Deep Learning avec les réseaux de neurones profonds.

2.2 Quelles applications aujourd'hui ?

On peut actuellement distinguer trois domaines d'application.

Le premier concerne les assistants personnels. Les plus connus sont Alexa d'Amazon, Google Assistant de Google, Cortana de Microsoft et Siri par Apple. Ces systèmes miment des assistants pour prévoir des rendez-vous, des rappels, envoyer des mails. Ils visent à simplifier la vie de l'utilisateur en remplaçant les multiples interfaces d'applications par une interface unifiée de questions-réponses. Ces systèmes s' "humanisent" très progressivement : ils nous disent bonjour, ils peuvent nous raconter des histoires drôles. Cependant, ce n'est pas un système créatif et subtil qui nous répond mais une base de données où les répliques appropriées ont été écrites manuellement.

Le deuxième champ d'application concerne le traitement des données. De très nombreuses données sont produites : en deux jours, nous produisons 5 exaoctets (Eo soit 10^{18} octets) d'in-

1. The Economist, January 7th 2017. Technology Quaterly : Langugage. Finding a voice.

formations. Cela rend impossible un traitement traditionnel de ces données pour en extraire des analyses utiles. La NLP permet d'automatiser le traitement de ces données. Parmi les services proposés, on peut classifier des documents, donner les opinions dominantes dans des tweets, des messages, des documents. On peut extraire les informations comme les noms, les lieux, les organisations dans un texte. Par ailleurs, au-delà de la compréhension des documents, on peut même écrire automatiquement des résumés de documents.

Le dernier champ d'application concerne encore la traduction automatique. Les traductions actuelles permettent au moins de comprendre le sujet d'un texte et fournissent une approximation utilisable. Actuellement tous les systèmes traduisent phrase par phrase ce qui peut poser des problèmes de cohérence.

2.3 Les défis

Malgré les progrès récents, les systèmes actuels de NLP échouent à comprendre le sens des mots. Pour certaines tâches spécifiques comme répondre à une question sur la météo, le système répondra de manière tout à fait satisfaisante. Mais dès qu'arrivent des ambigüités comme un passage ironique ou des métaphores [14], le système commet des erreurs. De même, les sous-entendus sont compris au premier degré par le système. Ces échecs proviennent de l'absence de représentation du monde pour la machine, qui travaille en aveugle.

Par ailleurs, d'autres problèmes plus abordables font l'objet de recherches actuellement :

- Comprendre et réagir sans contresens à des expressions de langage complexes : les figures de style (ironie, métaphore, ellipse), les mots étrangers, les sous-entendus sont difficiles à comprendre ;
- Gérer les problèmes de polysémie : c'est un obstacle majeur en traduction aujourd'hui ;
- Trouver le sujet d'une conversation, repérer les mots-clés sont des tâches nécessaires pour résumer des textes ;
- Produire un discours respectant les règles en vigueur de grammaire ;
- Répondre à des questions factuelles sur un texte. Par exemple, dans les phrases "Bob coupe du bois. Alice va l'aider.", il faudra répondre aux questions "que fait Bob ?" ou "qui Alice va-t-elle aider ?" ;
- Indiquer la nature, la fonction d'un mot dans une phrase (voir POS-Tagging) ;
- Définir des groupes de mots (comme un groupe nominal, un groupe verbal, une subordonnée).

Les progrès faits depuis l'arrivée des méthodes de Deep Learning en 2005 ont permis l'émergence de systèmes utilisables pour des tâches spécifiques, comme répondre à des clients dans un service après-vente téléphonique. Cependant, ces systèmes sont encore trop superficiels pour pouvoir être confondus avec une personne. La demande est pourtant là : Amazon offre un prix de 1 million d'euros pour un système capable de mener une conversation "cohérente et engageante" pendant 20 minutes.

2.4 La modélisation des langages

Les mots ont été modélisés par des vecteurs dès 1986 par Hochreiter *et al* [7]. La modélisation des séquences de mots (n-grams) se base sur des statistiques sur leur fréquence d'apparition et utilise surtout les techniques du *Bag-of-Words* [6] et *Skip-gram* [11]. Cependant les modèles à base de n-grams sont chronophages et moins performants que les réseaux de neurones récurrents (RNN), notamment dans la détection de similarité de sens entre des mots [12].

2.5 Une multitude de méthodes en Deep Learning

Nous présentons ici quelques types de réseaux utilisés en Deep Learning.

2.5.1 Les réseaux de neurones

Les réseaux de neurones sont apparus dès les années 1950 [10]. Ce sont des ensembles structurés d'unités de calcul appelés neurones. Ces neurones sont interconnectés : ils prennent des valeurs en entrée et renvoient une certaine valeur en sortie. On peut ainsi voir un neurone comme une

fonction. Cette fonction dépend de paramètres qui sont appris lors de l'entrainement du réseau. Généralement cette fonction est la composition d'une fonction affine et d'une fonction non-linéaire mais ce n'est pas toujours le cas (voir les LSTM, GRU). La combinaison de ces neurones forme un réseau de neurones.

Théoriquement, il existe une infinité de réseaux possibles et une infinité d'architectures possibles. En pratique, on est beaucoup plus restreint par des problèmes de performance : les fonctions, par exemple, doivent avoir un gradient facilement calculable par ordinateur.

L'entrainement du réseau, c'est-à-dire l'ajustement des paramètres des neurones, utilise des couples valides (donnée, label) = (valeur d'entrée, valeur de sortie). Lors de l'entrainement, la valeur d'entrée passe à travers le réseau et la valeur de sortie trouvée par le réseau est comparée à la valeur de sortie valide. On se retrouve alors dans un problème d'optimisation où la valeur à minimiser est la différence des valeurs de sortie. On peut consulter à ce sujet la technique de rétro-propagation du gradient [9].

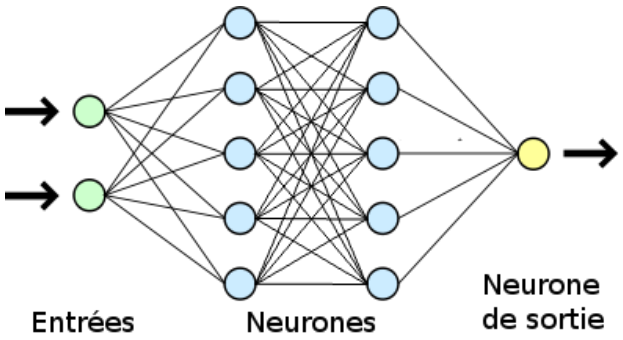

FIGURE 2 – Schéma d'un réseau de neurones

Le réseau de neurones le plus basique est le réseau linéaire (voir figure 2). Les réseaux de neurones profonds (Deep Neural Networks) sont des réseaux de neurones plus complexes qui possèdent souvent de nombreuses couches : cette multitude de couches permet de modéliser (i.e. de retenir des motifs) des phénomènes encore plus complexes.

2.5.2 Les réseaux de neurones récurrents (RNN)

Un réseau de neurones récurrent est schématiquement un réseau de neurones classique en boucle fermée. L'information reçue par les neurones ne provient pas uniquement des valeurs d'entrée mais également des anciennes valeurs de sortie. L'avantage de ce type d'architecture est que ce réseau est capable de retenir l'information qu'on vient de lui donner. Il est particulièrement utile pour le traitement de séquences, notamment en NLP. Il souffre cependant de problèmes techniques lors de l'entrainement (voir le problème du "Vanishing gradient" [7]) et sa "mémoire" est relativement courte.

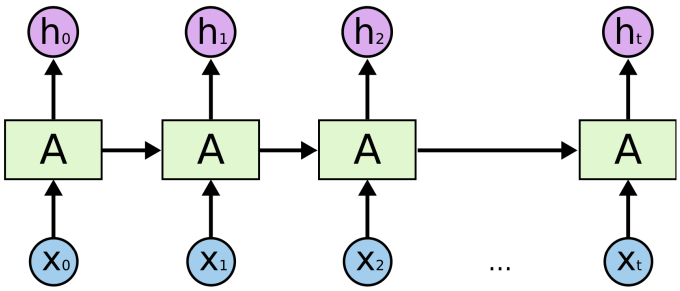

FIGURE 3 – Schéma d'un réseau de neurones récurrent. X_t représente les neurones par lesquels on donne les valeurs d'entrée. h_t représente les neurones de sortie.

2.5.3 Long Short-Term Memory et Gated Recurrent Units

Une évolution majeure est apparue avec l'utilisation des neurones à base de portes logiques. Nous avons vu que le passage de l'information dépend des paramètres associés aux neurones. Or il peut être intéressant de laisser passer ou pas l'information selon cette même information et pas uniquement sur la base du neurone par lequel il transite. L'architecture Long Short-Term Memory (ou LSTM) [7] ajoute au neurone trois portes logiques :

— Une porte "forget" : si cette porte vaut 1, l'information précédente est conservée en mémoire pour le calcul de la valeur de sortie du neurone ;
— Une porte "input" : si cette porte vaut 1, l'information nouvelle est prise en compte dans le calcul de la valeur de sortie ;
— Une porte "output" : si cette porte vaut 1, on transmet le résultat du calcul au neurone suivant.

L'intérêt de ce type de réseau est d'avoir un plus grand contrôle sur l'information qu'on laisse passer. En effet, si l'on traite une phrase, comme "*Les LSTM semblent de premier abord intimidants*", on a envie de voir les mots "*LSTM*" et "*intimidants*" perdurer dans le temps car ils sont importants. Au contraire, les mots "*les*", "*semblent*", "*de*", "*premier*", "*abord*" peuvent être oubliés sans perte d'information conséquente. La "mémoire" des LSTM est largement supérieure aux réseaux de neurones profonds : en 2016, la plupart des entreprises de technologie comme Google, Apple, Microsoft, Baidu utilisent les réseaux LSTM comme éléments de base de leurs produits. L'architecture de réseau GRU [3] est une variante de l'architecture LSTM : les portes "forget" et "input" sont combinées en une seule porte "update". Ses performances sont similaires aux LSTM mais avec moins de paramètres donc plus efficace numériquement.

2.6 Les recherches actuelles en NLG

L'utilisation des RNN pour répondre à certaines problématiques comme la génération de texte a donné lieu à d'intéressants résultats [15]. Cependant, ces modèles, basés sur la génération mot par mot, ne comprennent pas des concepts de plus haut niveau comme le sujet d'une conversation et échouent à créer des textes sensés. Les LSTM [7], une architecture basée sur les RNN qui améliore le stockage et l'accès aux informations ont fait leur preuve en NLP et en NLG [5].

Les travaux de Bowman [2], qui se basent sur les LSTM et les auto-encodeurs permettent d'améliorer significativement la cohérence des phrases générées.

Cependant nous constatons que les RNN précédemment vus se chargent de construire des phrases correctes à la fois sémantiquement et syntaxiquement.

Une approche que nous proposons est de distinguer ces rôles entre deux modèles, l'un chargé de construire la structure syntaxique, l'autre, chargé de remplir ce modèle des entités sémantiquement cohérentes.

Concernant l'évaluation de la combinaison de nos deux modèles, l'approche automatique est prometteuse mais un jugement humain est généralement conseillé [1].

3 Déroulement du projet

Le projet s'est déroulé en trois grandes étapes s'étalant de septembre 2016 jusqu'à avril 2017.

3.1 Documentation générale

La première étape, du début du projet jusqu'au 8 novembre (date du RVP 1), a consisté à acquérir des connaissances de base dans le domaine, pour lequel je suis parti de zéro. J'ai pour cela suivi les cours très complets de "Deep Learning for NLP" sur le site de Stanford. Ces cours m'ont permis de construire une vue générale de ce qui se fait en NLP, ainsi que les méthodes modernes utilisées pour résoudre les problèmes de NLP. J'ai aussi eu l'occasion de m'exercer sur des exercices de programmation pour la NLP. J'ai notamment implémenté un algorithme de construction de vecteur de mots et un autre algorithme qui catégorise les mots selon leur fonction (lieu, entités, organisation ou autre). J'ai également consulté un ouvrage plus général sur le Deep Learning [4] fourni par Alexandre Saïdi.

3.2 Définition du problème

La deuxième étape s'étend sur la période de novembre. Grâce à la vue d'ensemble que j'ai acquise, j'ai pu définir clairement le sujet et l'objectif de mon projet (voir l'introduction). J'y ai également mené la majeure partie de l'activité de recherche pour former l'état de l'art du domaine concernant la génération de phrases cohérentes. À partir des travaux déjà menés, j'ai pu construire la démarche qui m'a permis de résoudre le problème que je me suis fixé.

Cette démarche est la suivante :

1. Obtenir les données et les créer si nécessaire.
2. Implémenter un modèle à base de neurone ayant pour tâche de deviner la distribution de probabilité d'apparition des mots des trois phrases en sortie à partir des mots donnés en entrée. C'est une simplification forte : avec ce modèle, on ne peut savoir que si un mot ou non va apparaitre dans les phrases de sortie, mais on ne sait pas où précisément.
3. Implémenter un programme Python qui va recréer des phrases à partir de cette distribution de probabilité et des structures syntaxiques fournies.
4. Évaluer la qualité des phrases à partir des tableaux 1 et 2.

3.2.1 Évaluation

L'évaluation est effectuée par une personne et non par une machine. L'étude de Belz [1] nous montre que malgré les nombreux outils d'évaluation automatique, les problèmes de NLG sont mieux évalués par des méthodes humaines.

Pour évaluer les performances des deux algorithmes, j'ai établi une évaluation sur deux points importants : à quel point les phrases générées sont justes au niveau syntaxique et à quel point les phrases générées ont un sens.

Je suivrai la grille ci-dessous en attribuant pour chaque ensemble de trois phrases générées deux notes variant de 0 à 4 correspondant aux deux aspects mentionnés (voir tableaux 1 et 2).

Note	Signification
4	Tous les mots sont correctement orthographiés.
3	Entre 75 et 99% des mots sont orthographiés correctement.
2	Entre 50 et 75% des mots sont orthographiés correctement.
1	Entre 25 et 50% des mots sont orthographiés correctement.
0	Entre 0 et 25% des mots seulement sont orthographiés correctement.

Table 1 – Évaluation sur la syntaxe des phrases générées

Note	Signification
4	Un message se distingue clairement dans les phrases et peut avoir un sens dans une situation courante.
3	Un message se distingue clairement dans les phrases et peut avoir un sens dans une situation très particulière.
2	Aucun message ne se distingue clairement mais les mots utilisés restent dans le cadre d'une thématique. Certaines parties de phrase ont un sens mais l'ensemble n'en a pas.
1	Aucun message ne se distingue clairement mais les mots utilisés restent dans le cadre d'une thématique. Ni des morceaux de phrases, ni l'ensemble des phrases n'a de sens.
0	Aucun message ne se distingue clairement et les mots utilisés semblent avoir été choisis au hasard.

Table 2 – Évaluation sur le sens des phrases générées

3.3 Réalisation du livrable

La troisième phase, de décembre jusqu'à fin mars, consiste en l'implémentation du programme. Nous décrivons cette partie en détail dans la partie suivante.

4 Implémentation

L'implémentation d'un modèle fonctionnel s'est basée sur de nombreuses variations d'un modèle originel. Je présente dans la suite trois variations.

4.1 Première approche : exhaustive

4.1.1 Données : obtenir une bibliothèque de phrases

Pour mon modèle, j'ai besoin de nombreuses séquences de trois phrases contiguës. Je me suis basé à la fois sur une base de données existantes de contes de Grimm [2] et un ensemble de paragraphes d'encyclopédie en ligne Wikipédia [3]. Utiliser des phrases de contes de fées permet d'obtenir facilement des phrases simples à comprendre, avec des mots également simples. Ainsi, la tâche de l'ordinateur est simplifiée. Concernant les données Wikipédia, elles concernent des sujets très variés et élargissent la connaissance qui peut être apprise par l'ordinateur.

J'ai récupéré ces données à l'état brut. J'ai alors appliqué un ensemble de traitements pour utiliser ces données. J'ai notamment divisé le texte en un ensemble de phrases. Dans un fichier texte, je place alors à chaque ligne une phrase unique. Cette tâche n'est pas aussi facile qu'on pourrait le croire car une définition simpliste de phrase en tant que séquence de mots commençant par une majuscule et terminant par un point implique de nombreuses exceptions : les phrases ne se terminent par toutes par un point, mais aussi par des points d'exclamation, des points d'interrogation, des points de suspension... Il est aussi courant de voir des points apparaitre au milieu des phrases, pour signifier une abréviation par exemple. J'ai pour cela utilisé des expressions régulières pour rechercher les séquences de caractères indésirables et pour les remplacer par celles désirées.

Ainsi, pour obtenir des données, je n'ai qu'à prendre trois lignes consécutives. Cependant le propos des phrases varient beaucoup et il faut également séparer les phrases discutant d'un sujet et les autres. Pour cela, j'ai introduit le symbole "+++$+++" entre les phrases pour distinguer des propos différents. Un ensemble de phrases séparé par le symbole "+++$+++" représente donc un sujet de conversation : on dénomme cet ensemble par paragraphe dans la suite.

A la fin du traitement, j'obtiens cinq fichiers : le premier contenant toutes les données de contes de Grimm et les quatres autres pour les données Wikipédia. Les données obtenues sont résumées dans le tableau 3.

Nom du fichier	Nombre de paragraphes	Nombre de lignes	Poids (Mo)
`001.txt`	48	12 726	1,4
`002.txt`	25 648	120 154	13,2
`003.txt`	23 344	121 575	12,3
`004.txt`	26 119	124 285	13,1
`005.txt`	7 292	164 319	18,2

TABLE 3 – Quelques chiffres sur les données utilisées pour le modèle 1

Pour l'ensemble de ces données, on comptabilise 85 151 paragraphes, 543 059 lignes, 20 725 443 mots dont 47 883 uniques.

Pour entraîner un réseau, il est nécessaire d'avoir des données d'entraînement sous la forme (`données en entrée`, `données en sortie`). Dans notre cas, les données d'entrée sont les trois noms et le verbe et les données en sortie sont les phrases générées à partir de ces mots. Nous avons d'ores et déjà les phrases ; on va donc faire le processus inverse et extraire les mots de ces

2. `https://github.com/bscofield/fairy-tale-remix/blob/master/data/fairy-tales.json`
3. `https://github.com/tuili/wikipedia_sentences_dataset`

phrases pour avoir les données d'entrée. On pourrait stocker tous ces couples de données mais cela occuperait un espace disque trop important. Au lieu de cela, nous conservons les phrases et nous générons à la volée les couples de données à partir d'un générateur sous Python.

4.1.2 Données : obtenir une bibliothèque de séquences de POS-Tag

Une autre partie de l'implémentation nécessite l'utilisation de structures syntaxiques. Une structure syntaxique est une séquence de POS-Tags. Il existe déjà de telles bibliothèques de séquences (telle que le corpus Brown) où un rapide traitement est suffisant pour récupérer de telles séquences. Il existe également des outils (comme Syntaxnet) qui créent les POS-Tags à partir de n'importe quel texte. Afin d'obtenir toujours les même résultats, j'ai décidé de n'utiliser que Syntaxnet pour créer les structures syntaxiques.

4.1.3 Modèle

Pour rappel, nous voulons qu'à partir de quatre mots, notre modèle construise trois phrases. Cette tâche est très complexe, c'est pourquoi nous allons la simplifier. Le réseau que nous allons utiliser aura pour tâche de déterminer la distribution des mots dans les phrases d'arrivée au lieu des phrases complètes à partir des quatre mots. Les phrases complètes seront formées à partir de cette distribution par un autre programme. Cette distribution est représentée sous la forme d'un vecteur de la taille du vocabulaire. Dans notre cas, les données utilisent 47 883 mots différents. On aura alors un vecteur de taille 47 883. Chaque scalaire de ce vecteur représente la probabilité d'apparition d'un mot dans la phrase d'arrivée. C'est donc ce vecteur que le réseau devra apprendre. En entrée de ce réseau, un autre vecteur de taille 47 833 représentera les quatre mots en entrée. En pratique, ce vecteur sera rempli de zéros sauf aux indices correspondants aux mots effectivement donnés.

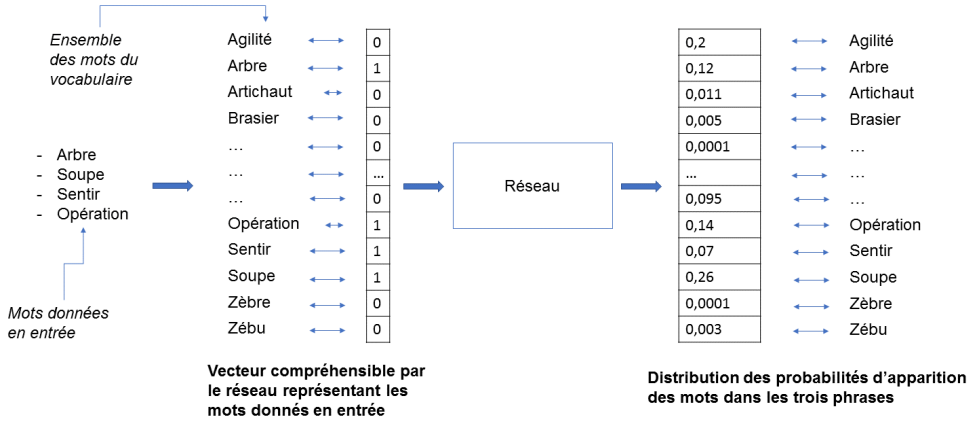

FIGURE 4 – Obtention de la distribution de probabilité d'apparition des mots à partir des quatre mots en entrée

Le réseau en lui-même est relativement peu complexe. On applique au vecteur de mots v_1 donné en entrée de taille 44 883 une matrice de taille $44\,883 \times 8\,000$:

$$v_2 = v_1 \times W_2 \tag{1}$$

Cette opération consiste finalement à associer à chaque mot un vecteur de taille 8 000 : c'est un embedding.

On applique au vecteur v_2 une transformation affine avec W_2 de taille $8\,000 \times 5\,000$ puis une opération non linéaire (ReLU) qui représente la première couche du réseau :

$$v_3 = \sigma(v_2 \times W_3 + b_3) \tag{2}$$

On applique de même une deuxième couche affine avec W_4 de taille $5\,000 \times 44\,883$, suivie d'une normalisation pour représenter des probabilités :

$$v_4 = \frac{v_3 \times W_4 + b_4}{||v_3 \times W_4 + b_4||} \tag{3}$$

Le vecteur est le vecteur représentant la distribution de probabilité. Il a été obtenu à partir de calculs reposant sur les données des matrices W_2, W_3 et W_4 et des vecteurs b_3 et b_4. Ce sont les valeurs de ces variables qui vont être modifiées pour obtenir un réseau ayant appris une connaissance.

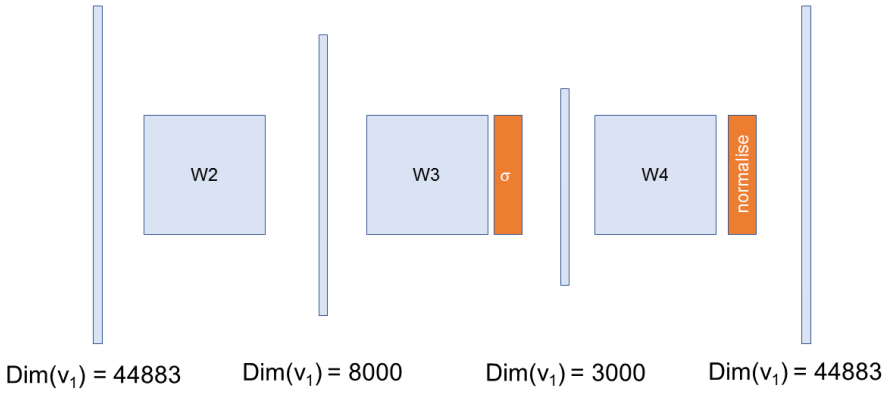

FIGURE 5 – Schéma du réseau de neurones mis en place dans ce premer modèle

En parallèle, on calcule à partir des trois phrases recherchées l'autre distribution $v_{4,\text{recherché}}$. Chaque valeur de ce vecteur représente le nombre d'occurrences dans les trois phrases de chaque mot. On normalise ensuite ce vecteur.

On peut finalement comparer les deux vecteurs v_4 et $v_{4,\text{recherché}}$, avec une norme L2 et chercher à minimiser la distance entre ces deux vecteurs. C'est alors un problème d'optimisation pour lequel il s'agit de trouver les valeurs de W_2, W_3, W_4, b_3 et b_4 minimisant les distances pour chaque essai.

Je n'ai pas eu à coder cette partie d'optimisation : TensorFlow s'en occupe automatiquement, ce qui fait sa force. La méthode d'optimisation employée est spécifiée par l'utilisateur ; dans notre cas, l'optimiseur ADAM, basé sur une descente de gradient stochastique.

Désormais, nous avons la distribution de probabilité d'apparition des mots dans les trois phrases. On peut associer à chacun de ces mots un POS-tag parmi 42 possible (par exemple : déterminant, nom singulier, nom pluriel, verbe à l'infinitif, préposition...). On classifie tous les mots dans une des catégories à laquelle il appartient. Par ailleurs, on conserve la probabilité associée à chaque mot dans la distribution.

Après avoir classifié tous les mots, il faut remplir les structures syntaxiques. Ces structures syntaxiques sont des séquences de POS-tag (voir le schéma en figure 7). Chaque POS-tag doit être remplacé par un mot ayant un tel POS-tag. Il suffit d'en choisir un parmi ceux classé dans la liste précédemment créée avec le POS-tag correspondant. Pour le choisir, on choisit le mot ayant la plus forte probabilité puis on l'enlève de la liste.

Pour respecter les consignes du projet qui sont de voir apparaître dans les trois phrases les quatre mots donnés en entrée, on ajoute les mots en entrée dans les listes auxquelles ils s'appartiennent, c'est-à-dire la liste "mot singulier" et "verbe" avec des probabilités telles qu'ils apparaissent dans les phrases au moment voulu.

4.1.4 Résultats

En entraînant le réseau, la valeur à minimiser, qui est la somme des distances entre distributions diminue rapidement : au bout de 200 itérations, on atteint un limite à 14.

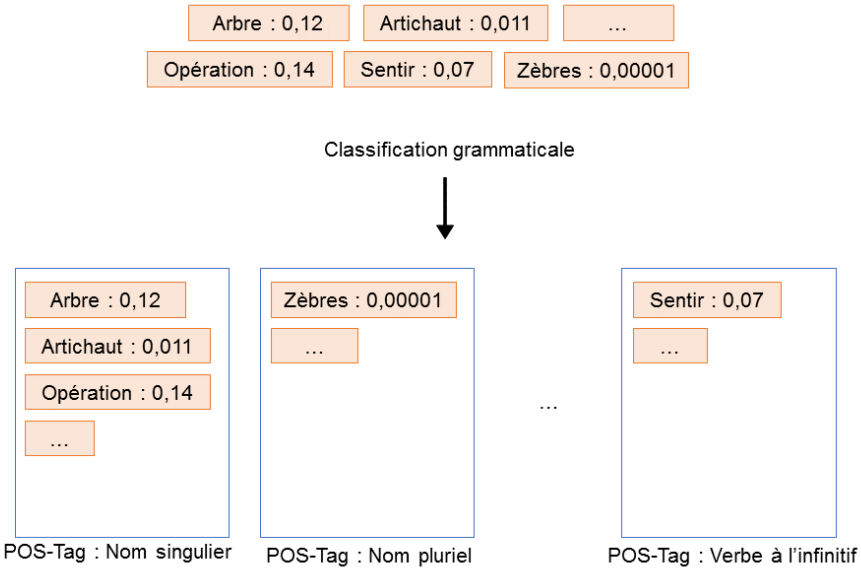

FIGURE 6 – Classification des mots selon leur POS-tag

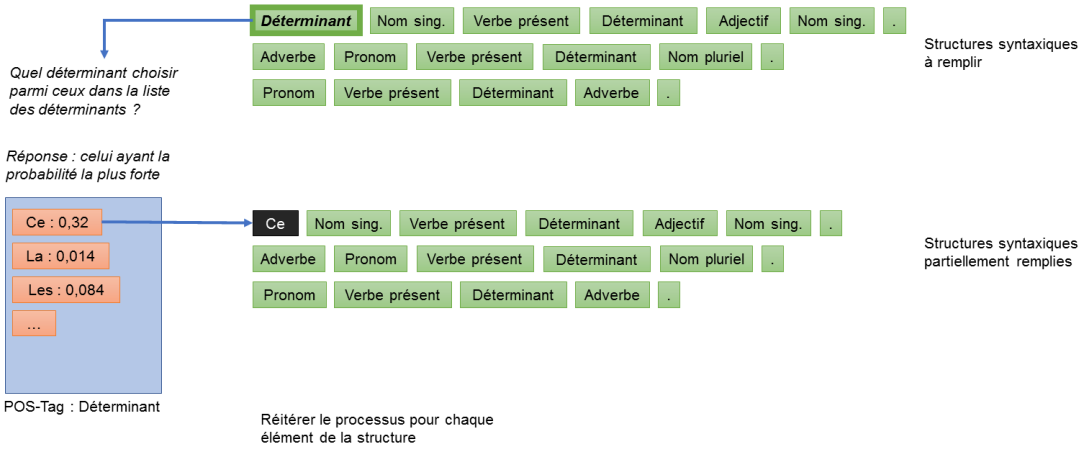

FIGURE 7 – Construction des phrases à partir des structures syntaxiques et des listes de mots

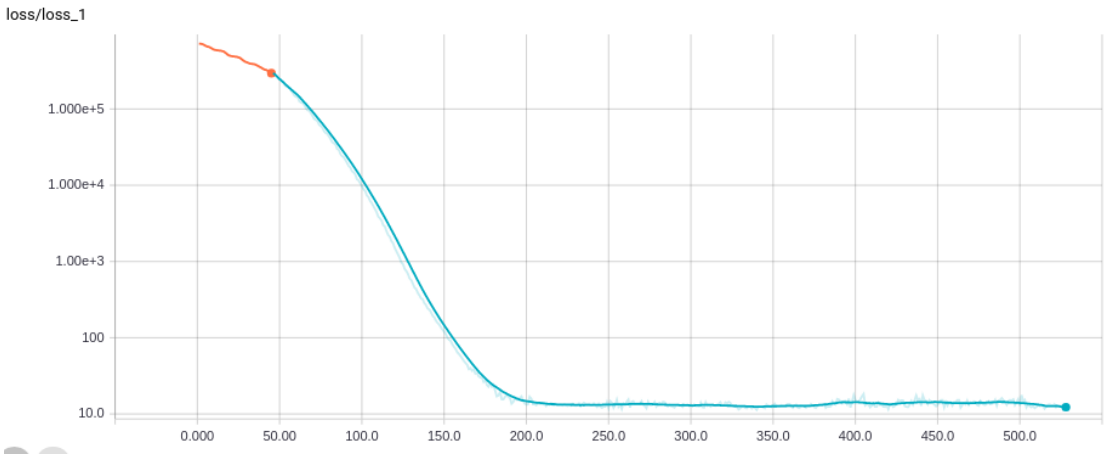

FIGURE 8 – Évolution du coût à minimiser en fonction du nombre d'itérations (échelle logarithmique)

Nom 1, Nom 2, Verbe 1, Nom 3	Phrases obtenues	Note syntaxe	Note sens
life, moon, slept, cake	Into tyotkino life any moon slept a useless embracing. Whether ither famine hibiscus yen funded andthin cultural capitalization. On er keiretsu ufdc envoy drifted thy unabashed cake.	2 (70%)	0
witch, dog, wandered, men	Into tyotkino witch any dog wandered a useless embracing. Whether ither famine hibiscus yen funded andthin cultural capitalization. On er keiretsu ufdc envoy drifted thy unabashed men.	2 (70%)	0
earth, men, replied, voice	Into tyotkino earth any men replied a useless embracing. Whether ither famine hibiscus yen funded andthin cultural capitalization. On er keiretsu ufdc envoy drifted thy unabashed voice.	2 (70%)	0
mother, people, predicted, thing	Into tyotkino mother any people predicted a useless embracing. Whether ither famine hibiscus yen funded andthin cultural capitalization. On er keiretsu ufdc envoy drifted thy unabashed thing.	2 (70%)	0
city, kingdom, helped, language	Into tyotkino city any kingdom helped a useless embracing. Whether ither famine hibiscus yen funded andthin cultural capitalization. On er keiretsu ufdc envoy drifted thy unabashed language.	2 (70%)	0
pope, earth, sail, nut	Into tyotkino pope any earth sail a useless embracing. Whether ither famine hibiscus yen funded andthin cultural capitalization. On er keiretsu ufdc envoy drifted thy unabashed nut.	2 (70%)	0
sky, chairs, lost, princes	Into tyotkino sky any chairs lost a useless embracing. Whether ither famine hibiscus yen funded andthin cultural capitalization. On er keiretsu ufdc envoy drifted thy unabashed princes.	2 (70%)	0

TABLE 4 – Résultats obtenus avec le premier modèle : mots en entrée, phrases en sortie et évaluation

On obtient avec les distributions obtenus et les mots donnés en entrée les résultats du tableau 4. On constate alors que les résultats ne sont pas du tout concluant : de nombreux mots inconnus apparaissent régulièrement, alors qu'ils ont la probabilité d'apparition la plus probable : ces mots proviennent des données des articles Wikipédia, qui utilise un vocabulaire à la fois vaste et spécifique. Finalement, malgré les différents mots fournis en entrée, le réseau renvoie quasi systématiquement les mêmes distributions de probabilité.

Ces performances décevantes se traduisent logiquement par des scores peu élevés.

4.2 Deuxième approche : réduction du vocabulaire

Pour résoudre quelques problèmes du modèle précédent, on décide de réduire la taille du vocabulaire et plus spécifiquement de restreindre les données aux contes de fées uniquement. Les résultats sont alors corrects mais il reste un problème de répétitivité.

4.2.1 Données

Le modèle précédent demande un vocabulaire trop important : en effet, les articles de Wikipédia incluent de nombreux mots peu fréquents, comme des noms propres ou du vocabulaire très spécifique. J'ai alors restreint la base de données à une partie seulement des contes de Grimm (voir tableau 5). On passe alors d'un vocabulaire de 44 883 mots à un vocabulaire de 2 218 mots.

Nom du fichier	Nombre de paragraphes	Nombre de lignes	Poids (Ko)
`001_small.txt`	4	1 142	111

TABLE 5 – Quelques chiffres sur les données utilisées pour le modèle 2

En conservant uniquement les textes de Grimm, on conserve les mots les plus fréquemment utilisés au quotidien, car ce sont des histoires destinées aux enfants. Un effet secondaire cependant réside dans le fait que ces contes ont été écrits au XIX^{e} siècle, d'où la présence occasionnelle de mots désuets.

4.2.2 Modèle

On conserve le même modèle que dans la partie précédente, à savoir un réseau avec deux couches cachées de taille 8 000 et 5 000, suivie d'un post-traitement pour recréer les trois phrases (voir la figure 9).

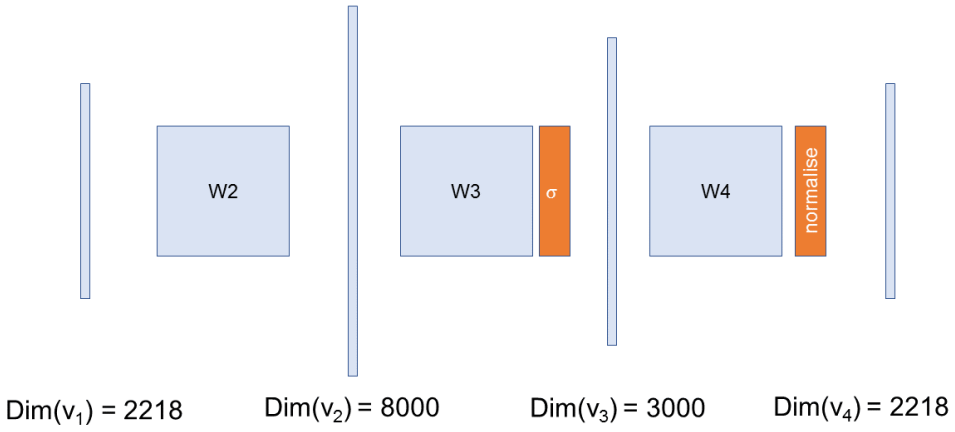

FIGURE 9 – Schéma du réseau de neurones mis en place dans ce deuxième modèle

4.2.3 Résultats

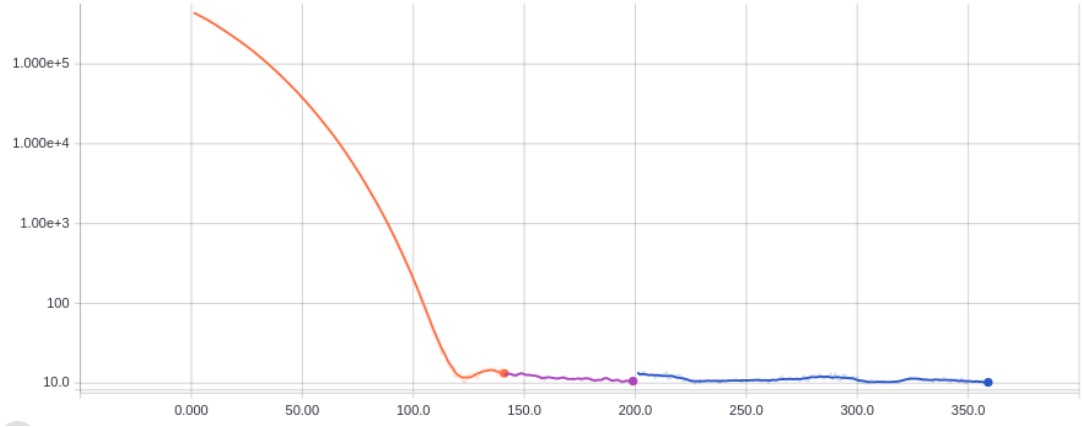

FIGURE 10 – Évolution du coût à minimiser en fonction du nombre d'itérations pour le modèle 2 (échelle logarithmique)

Avec ce modèle, la convergence des résultats est plus rapide (voir figure 10) : après 120 itérations, on atteint la limite à environ 10. Les résultats textuels (voir tableau 6) sont bien plus satisfaisants : les groupes de trois mots ont un sens et globalement, les phrases se lisent facilement.

Du point de la correction de la langue, on note surtout des erreurs d'accord, notamment entre les déterminants et les noms. Ceci s'explique par le fait que le réseau n'est pas prévu pour gérer ces détails : ces liens grammaticaux ne sont pas conservés dans la distribution de probabilité.

Concernant le sens, il est difficile d'attribuer un sens aux phrases produites, sachant qu'elles se ressemblent toutes. De même que dans le modèle précédent, quels que soient les mots donnés en entrée, la distribution renvoyée est toujours quasiment la même (expliquant les phrases similaires). En y regardant d'un peu plus près, on constate que ce ne sont pas exactement les mêmes mots, démontrant ainsi que les variations entre les distributions de probabilité créées existent mais sont trop faibles pour avoir une influence.

4.3 Troisième approche : suppression de la non-linéarité

4.3.1 Données

Cette fois-ci, on reprend la base de donnés des contes de fées mais en ayant un peu plus de données que dans le modèle précédent (voir tableau 7. En effet, trop peu de données est une cause potentielle des résultats toujours identiques vu précédemment.

Avec ces données, on obtient un total de 4 877 mots, pour 15 paragraphes, c'est-à-dire 15 contes.

4.3.2 Modèle

Nous choisissons de travailler sur un modèle extrêmement simplifié (voir figure 11) : il n'y a dans ce réseau plus qu'une seule couche et surtout, nous supprimons la fonction de non-linéarité (ReLU).

4.3.3 Résultats

La convergence du modèle (voir figure 12) s'obtient au bout du 160 itérations ; le sursaut que l'on peut voir à l'itération 120 correspond à un overfitting : le modèle est tombé dans un minimum local mais a réussi à s'en extraire pour aller dans un minimum où la convergence est atteinte à 13.

Nom 1, Nom 2, Verbe 1, Nom 3	Phrases obtenues	Note syntaxe	Note sens
life, moon, slept, cake	In this life an moon slept the beautiful fidelity. Upon every mourning some wonder said any unrecognizable deathbed. Of those dress each coat stood a whole cake.	3 (88%)	1
witch, dog, wandered, men	In this witch an dog wandered the beautiful fidelity. Upon every mourning some wonder said any unrecognizable deathbed. Of those dress each coat stood a whole men.	3 (88%)	1
earth, men, replied, voice	In this earth an men replied the beautiful fidelity. Upon every mourning some wonder said any unrecognizable deathbed. Of those dress each coat stood a whole voice.	3 (88%)	2
mother, people, predicted, thing	In this mother an people predicted the beautiful fidelity. Upon every mourning some wonder said any unrecognizable deathbed. Of those dress each coat stood a whole thing.	3 (88%)	2
city, kingdom, helped, language	In this city an kingdom helped the beautiful fidelity. Upon every mourning some wonder said any unrecognizable deathbed. Of those dress each coat stood a whole language.	3 (88%)	2
pope, earth, sail, nut	Around these pope a earth sail both next stone. Past each fashion any well sewed those angry dungeon. Up some way every household happened the watchful nut.	3 (81%)	1
sky, chairs, lost, princes	Around these sky a chairs lost both next stone. Past each fashion any dungeon ran those angry household. Across some furniture every vessel sewed the watchful princes.	3 (88%)	1

Table 6 – Résultats obtenus avec le modèle 2 : mots en entrée, phrases en sortie et évaluation

Nom du fichier	Nombre de paragraphes	Nombre de lignes	Poids (Ko)
`001.txt`	15	8 400	557

Table 7 – Quelques chiffres sur les données utilisées dans le modèle 3

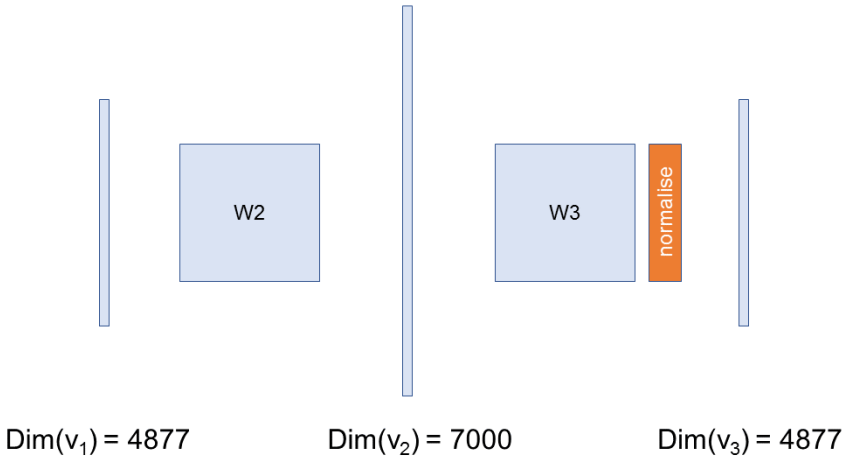

FIGURE 11 – Schéma du réseau de neurones mis en place dans le troisième modèle

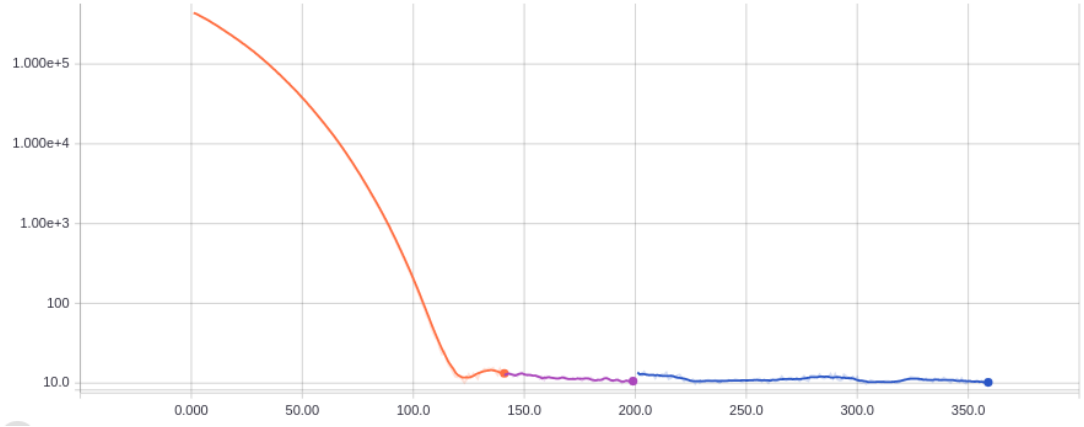

FIGURE 12 – Évolution du coût à minimiser en fonction du nombre d'itérations pour le modèle 3 (échelle logarithmique)

Nom 1, Nom 2, Verbe 1, Nom 3	Phrases obtenues	Note syntaxe	Note sens
life, moon, slept, cake	Before each life these moon slept some worried hen. Besides thy writing another ink opposed every overjoyed difficulty. Lest the miller any court pitied this broad cake.	3 (96%)	2
witch, dog, wandered, men	Towards every witch the dog wandered another favorite roe. With this bail both fair sprang each thorough music. Along these vessel any countenance sniffed some heavy men.	3 (93%)	2
earth, men, replied, voice	Since an earth every men replied the third moat. Under no stroke both waggon rode some rascal chin. Lest those swoon another eye feigned each pure voice.	3 (93%)	2
mother, people, predicted, thing	Along an mother this people predicted some delicate story. Besides both midnight the troop gave another watchful sweat. Since each awakening these knelt slaughtered those certain thing.	3 (96%)	2
city, kingdom, helped, language	By the city an kingdom helped some pleasant pig. Under another lip both foal slaughtered each smooth oxen. Except these parade a bowl bemoaned thy innocent language.	3 (93%)	2
pope, earth, sail, nut	Unto another pope some earth sail both sooty vault. With this dearth each harm carried these sure pantry. From any remembrance the greyhound began thy basketful nut.	3 (93%)	2
sky, chairs, lost, princes	Before this sky thy chairs lost a helpless pinafore. Along another cottage both mare stopped each anxious ditch. Besides these self the chalk pitied any responsible princes.	3 (96%)	2

TABLE 8 – Résultats obtenus avec le modèle 3 : mots en entrée, phrases en sortie et évaluation

Les résultats textuels (voir tableau 8) sont finalement satisfaisants : non seulement les phrases utilisent les mots donnés en entrée, mais l'ensemble des mots générés forme des phrases syntaxiquement correctes et dont les phrases changent avec les mots donnés en entrée. Cela montre que ces mots ont une influence sur les probabilités d'apparition des mots.

Cependant, les situations présentées dans les phrases nécessitent toujours une grande imagination et il reste difficile d'établir un lien solide de cause à effet entre les mots donnés en entrée et ceux générés par le réseau.

Par ailleurs, on peut se demander pourquoi la suppression de la non-linéarité a permis de générer des phrases différentes dans ce modèle. (À partir d'autres modèles, on peut vérifier que le nombre de couches n'est pas à l'origine ce phénomène). Une explication pourrait être que la non-linéarité autorise plus facilement le réseau à toujours se concentrer sur les mêmes mots quels que soient les mots en entrée. Sa suppression refait passer le réseau en un modèle linéaire qui l'oblige à prendre en compte les mots en entrée.

5 Participation extérieure

Au delà de mon projet, j'ai pu collaborer avec le laboratoire LIRIS en aidant un groupe de troisième année. Je leur ai notamment apporté des bases de données de texte dont ils avaient besoin pour leur projet sur le traitement de texte avec des méthodes de machine-learning. J'ai également pu assister à une consistance du LIRIS. Je remettrai enfin à Alexandre Saïdi un rapport synthétique des techniques utilisées pour prendre en main TensorFlow, outil récent sans grande documentation en français.

Par ailleurs, j'ai envoyé mes programmes sur la plateforme publique GitHub [4] afin que quiconque souhaitant continuer mon travail ait les moyens de continuer. Les données d'entraînement s'y trouvent également [5].

6 Conclusion

La génération de phrases est l'un des domaines les plus complexes : il est difficile de juger en toute objectivité de la qualité des phrases créées. Cela demande également d'avoir une connaissance fine du sens de la communication et de son interlocuteur, ce que les systèmes actuels ne sont pas encore en mesure de réaliser.

Cependant, nous avons pu voir qu'avec des méthodes très simples, basées sur la séparation initiale entre structure grammaticale et sémantique, il est possible de créer des phrases compréhensibles. Cette compréhension est notamment due aux structures grammaticales. Concernant le sens des phrases, le résultat est satisfaisant pour une architecture de réseau aussi basique.

La combinaison d'architectures plus complexes et de structures syntaxiques peut mener à des améliorations significatives concernant le sens des phrases.

Glossaire

Auto-encodeur Réseau de neurones utilisé pour encoder un ensemble de données, souvent dans le cadre de la réduction de dimensionnalité. Le concept d'auto-encodeur est de plus en plus utilisé pour apprendre des modèles génératifs.

Bag-of-Word Modèle de représentation simplifiée utilisé en NLP. Dans ce modèle, un texte est représenté comme un ensemble de mots non ordonné. Les occurences des mots servent de seule variable d'analyse.

Chatbot Logiciel autonome qui dialogue avec un utilisateur.

Continuous Bag-of-Word (CBOW) Modèle de prédiction d'un mot manquant, connaissant les mots environnants (contexte). Le but de ce modèle est de renvoyer des vecteurs de mots performants pour d'autres tâches de NLP. Voir aussi Skip-gram.

4. `https://github.com/tuili/PAr142`
5. `https://github.com/tuili/wikipedia_sentences_dataset`

Data-driven system Approche d'apprentissage des langues qui se base sur la mémorisation d'exemples corrects.

Expression régulière Chaîne de caractères permettant la recherche d'une séquence de caractères. Construite à partir d'une syntaxe précise, elle autorise les motifs les plus complexes.

Générateur Dans le langage Python, instance d'une fonction retournant une séquence de valeurs plutôt qu'une unique valeur. Il est possible d'obtenir une valeur, continuer le script, et obtenir la valeur suivante, sans avoir à tout recalculer.

GitHub Service d'hébergement de logiciels et de gestion de version. Il s'appuie sur le logiciel Git qui permet de faciliter le travail en équipe sur un même code source.

Hyperparamètres Paramètres décrivant l'architecture d'un réseau de neurone. S'utilisent par opposition aux simples paramètres qui décrivent la connaissance apprise par le réseau.

Language Model Distribution de probabilité sur les séquence de mots, utilisé dans les applications de génération de texte (voir NLG).

Long short-term memory (LSTM) Architecture de RNN adaptée pour les expériences requérant une mémoire à long terme.

Machine learning Champ d'étude de l'intelligence artificielle qui concerne les méthodes permettant à un système d'évoluer et de remplir des tâches plus difficiles à résoudre par les moyens algorithmiques classiques. L'apprentissage est induit de nombreux exemples. Les réseaux de neurones font partie des méthodes de machine learning.

n-gram Séquence d'entités (souvent des mots) contigües utile pour les Language Models.

Natural Language Generation (NLG) Génération automatique de texte. Domaine d'application de la NLP qui vise à générer un texte à partir d'un système de représentation comme des bases de connaissance (voir Data-driven system) ou de formes logiques (voir Rule-based system)

Natural Language Processing (NLP) Traitement de langues naturelles. Discipline à la frontière de la linguistique, de l'informatique et de l'intelligence artificielle, qui concerne l'application de programmes et techniques informatiques à tous les aspects du langage humain.

Parser Programme informatique qui reçoit une séquence d'entités et qui la sépare en séquences plus petites. Pour le language, il s'agit de distinguer dans une phrase les ensembles de mots exécutant la même fonction grammaticale (COD, sujet, complément circonstanciel...).

POS-Tag Information grammaticale correspondant à un mot d'un texte comme le genre, le nombre, la nature grammaticale.

ReLU Rectified Linear Unit : fonction d'activation définie par $f(x) = \max(0, x)$.

Réseau de neurones Ensemble de neurones formels interconnectés permettant la résolution de problèmes complexes tels que la reconnaissance des formes ou le traitement du langage naturel, grâce à l'ajustement des coefficients de pondération dans une phase d'apprentissage.

Réseau de neurones récurrent Modèle de réseau de neurones bouclé.

Rule-based system En NLG, système de construction d'un Language Model à partir de la connaissance de règles syntaxiques et sémantiques.

Skip-gram Modèle de prédiction des mots environnant dans une phrase (contexte) à partir d'un mot central. Utile pour créer des vecteurs de mots performants dans les tâches de NLP. Voir aussi CBOW.

Structure syntaxique Ensemble de POS-Tags ordonnés représentant la contruction syntaxique d'une phrase.

Syntaxnet Outil de NLP capable d'associer à chaque mot un POS-Tag. C'est aussi un parser.

TensorFlow Outil de programmation spécialisé dans la création de modèles de machine-learning. S'utilise principalement en tant que bibliothèque Python.

Variable latente Variable qui n'est pas directement mesurable mais inférée à partir d'autres variables avec un modèle mathématique.

Vecteur de mot Représentation de mot d'un dictionnaire par un vecteur afin de faciliter leur analyse sémantique et syntaxique. Cette représentation permet de réduire l'espace de dimensionalité (de l'ordre de la dimension 100).

Références

[1] Anja Belz and Ehud Reiter. Comparing automatic and human evaluation of NLG systems. In *EACL*, pages 313–320. The Association for Computer Linguistics, 2006.

[2] Samuel R. Bowman, Luke Vilnis, Oriol Vinyals, Andrew M. Dai, Rafal Józefowicz, and Samy Bengio. Generating sentences from a continuous space. *CoRR*, abs/1511.06349, 2015.

[3] Kyunghyun Cho, Bart van Merrienboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation : Encoder-decoder approaches. *CoRR*, abs/1409.1259, 2014.

[4] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`. Consulté le 9 janvier 2017.

[5] Alex Graves. Generating sequences with recurrent neural networks. *CoRR*, abs/1308.0850, 2013.

[6] Zellig Harris. Distributional structure. *Word*, 10(23) :146–162, 1954.

[7] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8) :1735–1780, 1997.

[8] Karen Sparck Jones. *Natural Language Processing : A Historical Review*, pages 3–16. Springer Netherlands, 1994.

[9] Seppo Linnainmaa. The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors. Master's thesis, Univ. Helsinki, 1970.

[10] Warren S. McCulloch and Walter Pitts. Neurocomputing : Foundations of research. chapter A Logical Calculus of the Ideas Immanent in Nervous Activity, pages 15–27. MIT Press, Cambridge, MA, USA, 1988.

[11] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

[12] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc., 2013.

[13] David E. Rumelhart, Paul Smolensky, James L. McClelland, and Geoffrey E. Hinton. Schemata and sequential thought processes in pdp models. In *Parallel Distributed Processing. Volume 2 : Psychological and Biological Models*, pages 7–57. MIT Press, Cambridge, MA, 1986.

[14] Ekaterina Shutova, Lin Sun, Dario Gutierrez, Patricia Lichtenstein, and Srini Narayanan. Multilingual metaphor processing : Experiments with semi-supervised and unsupervised learning. *Computational Linguistics*, 2017. (to appear).

[15] Ilya Sutskever, James Martens, and Geoffrey Hinton. Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on Machine Learning*, pages 1017–1024. ACM, 2011.

[16] Alan M. Turing. Computers & Thought. chapter Computing Machinery and Intelligence, pages 11–35. MIT Press, Cambridge, MA, USA, 1995.