Université de Montréal

Un système de question-réponse simple appliqué à SQuAD

par

Ilan Elbaz

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté à la Faculté des arts et des sciences
en vue de l'obtention du grade de
Maître ès sciences (M.Sc.)
en Informatique

mars 2020

Université de Montréal

Faculté des arts et des sciences

Ce mémoire intitulé

Un système de question-réponse simple appliqué à SQuAD

présenté par

Ilan Elbaz

a été évalué par un jury composé des personnes suivantes :

François Major
(président-rapporteur)

Philippe Langlais
(directeur de recherche)

Guillaume Lajoie
(membre du jury)

Mémoire accepté le :
mai 2020

Sommaire

La tâche de question-réponse (*Question-Answering*, QA) est bien ancrée dans la communauté de Traitement Automatique du Langage Naturel (TALN) depuis de nombreuses années. De manière générale, celle-ci consiste à répondre à des questions données à l'aide de documents (textuels ou autres) ou de conversations en faisant au besoin usage de connaissances et en mettant en œuvre des mécanismes d'inférence. Ainsi, dépendamment du jeu de données et de la tâche lui étant associée, il faut que le système puisse détecter et comprendre les éléments utiles pour répondre correctement à chacune des questions posées. De nombreux progrès ont été réalisés depuis quelques années avec des modèles neuronaux de plus en plus complexes, ces derniers sont cependant coûteux en production, et relativement opaques. Du à leur opacité, il est difficile d'anticiper avec précision le comportement de certains modèles et d'ainsi prévoir quand ces systèmes vont retourner de mauvaises réponses. Contrairement à la très grande majorité des systèmes proposés actuellement, nous allons dans ce mémoire tenter de résoudre cette tâche avec des modèles de taille contrôlable, on s'intéressera principalement aux approches basées sur les traits (*features*). Le but visé en restreignant la taille des modèles est qu'ils généralisent mieux. On pourra alors mesurer ce que ces modèles capturent afin d'évaluer la granularité de leur "compréhension" de la langue. Aussi, en analysant les lacunes de modèles de taille contrôlable, on pourra mettre en valeur ce que des modèles plus complexes ont capturé. Pour réaliser notre étude, on s'évalue ici sur SQuAD: un jeu de données populaire proposé par l'Université Standford.

Mots-clés: Question-Réponse, SQuAD

Summary

The Question-Answering task (QA) is a well established Natural Language Processing (NLP) task. Generally speaking, it consists in answering questions using documents (textual or otherwise) or conversations, making use of knowledge if necessary and implementing inference mechanisms. Thus, depending on the data set and the task associated with it, the system must be able to detect and understand the useful elements to correctly answer each of the questions asked. A lot of progress has been made in recent years with increasingly complex neural models. They are however expensive in production, and relatively opaque. Due to this opacity, it is difficult to accurately predict the behavior of some models and thus, to predict when these systems will return wrong answers. Unlike the vast majority of systems currently proposed, in this thesis we will try to solve this task with models with controllable size. We will focus mainly on feature-based approaches. The goal in restricting the size of the models is that they generalize better. So we will measure what these models capture in order to assess the granularity of their "understanding" of the language. Also, by analyzing the gaps of controllable size models, we will be able to highlight what more complex models have captured. To carry out our study, we evaluate ourselves here on SQuAD: a popular data set offered by Standford University.
Keywords: Question-Answering, SQuAD

Contents

List of tables

List of figures

Remerciements

"C'est par le travail que l'homme se transforme"[1]. C'est au terme de ces années de travail que je réalise à quel point elles m'ont été formatrices. Après avoir abordé timidement cette recherche, j'ai acquis au fil des mois grâce à de nombreuses personnes, diverses méthodes de travail et d'analyse celles-ci m'ont permis de prendre confiance en mon travail.

Je souhaite avant tout remercier mon directeur Philippe Langlais, pour le temps qu'il a consacré à m'apporter les outils méthodologiques indispensables à la conduite de cette recherche. Son exigence dans les moments forts et moins forts de ma recherche m'a grandement stimulé. Je voudrais exprimer ma reconnaissance envers mes parents qui m'ont apporté leur soutien moral tout au long de ma démarche. Enfin, je tiens à témoigner toute ma gratitude à Fabrizio ainsi qu'à l'ensemble des membres du RALI pour leur soutiens et leurs conseils. En particulier j'aimerai remercier David, Fredéric, Khalil, Louis et Olivier pour les nombreuses conversations qui m'ont permis de m'enrichir intellectuellement.

[1]Citation de Louis Aragon

Chapter 1

Introduction

On évalue souvent l'intelligence d'une personne en fonction de sa capacité de réponse à des questions plus ou moins difficiles. C'est sans doute pour cette raison, et pour améliorer l'interaction homme-machine, que très tôt la communauté de Traitement Automatique du Langage Naturel (TALN) s'est intéressé à la tâche de Question-Réponse (*Question-Answering*, QA). Relevant à la fois du domaine du Traitement Automatique des Langues Naturelles et de la Recherche d'Informations (RI), le Question-Réponse vise à développer un système fournissant une réponse exacte à une question posée par l'utilisateur en langage naturel. Du point de vue de la recherche d'informations le QA est "une forme sophistiquée de recherche d'informations caractérisée par un besoin d'informations au moins partiellement exprimées en langage naturel" [Kolomiyets and Moens, 2011]. En revanche selon la communauté du Traitement Automatique des Langues Naturelles, le QA est "la technologie qui localise, extrait et représente une réponse spécifique à une question posée par l'utilisateur en langage naturel" [Barskar et al., 2012].

Au quotidien, il existe une grande variété de types de questions qui sont posées:

- Les questions factuelles portant sur la connaissance du monde en général. Celles-ci permettent d'effectuer des recherches à propos de faits réels. Étant donné que ces questions viennent généralement avec des réponses standards, il est possible de juger de la véracité des réponses produites. Voici quelques exemples de questions factuelles:
  - Quel âge à Chilly Gonzales? (une courte phrase suffit pour répondre)
  - Qui étaient les 10 derniers rois de France? (on attend ici un court texte)

  - Qu'est ce que l'informatique? (une définition est nécessaire pour y répondre)
  - Pourquoi le ciel est bleu? (réponse nécessitant une explication)
- Les questions d'opinion et de croyances subjectives. Habituellement, ces questions n'ont pas de réponses définitives, car elles varient en fonction des individus. Cependant, il est possible d'évaluer la qualité des réponses en les jugeant comme pertinentes, acceptables, non pertinentes ou inacceptables. Par exemple:
  - Quelle est la plus belle peinture de René Magritte? (un avis est attendu)
  - Quelle robe préfères tu? (un autre avis potentiellement conflictuel est attendu)
  - Que se passe-t-il si le chat traverse la route? (ici la réponse est une hypothèse)
  - Comment faire pousser un potager? (réponse nécessitant un cheminement évolutif)

Il est possible de mesurer à l'aide de métriques d'évaluation, la véracité d'une réponse à une question factuelle de manière quantitative (voir la section 3.2). Dans le cas des questions d'opinion et de croyances subjectives, étant donné qu'il n'existe pas de réponses standard, on utilise souvent la stratégie du vote d'utilisateurs pour classifier et évaluer l'acceptabilité et la pertinence des réponses. Contrairement aux réponses de questions factuelles, qui souvent peuvent être extraites automatiquement dans des sources existantes, les réponses de questions d'opinion et de croyances subjectives sont la plupart du temps produites à partir de zéro ou agrégées.

1.1. Historique

Les systèmes de Question-Réponse existent depuis assez longtemps et ont été largement utilisés en raison de leurs applications et de leurs résultats prometteurs. En présentant directement à l'utilisateur une réponse à sa question, ce dernier a le sentiment de dialoguer directement avec le système. C'est sans doute pour développer cela qu'il y a eu et qu'il y a encore un aussi grand nombre de travaux et d'applications de QA. On distingue généralement les systèmes de QA en deux catégories: ceux à domaine ouvert et ceux à domaine fermé. Le Question-Réponse à domaine ouvert inclut des questions sur presque tout, alors que dans le cas d'un système à domaine fermé, les questions traitent d'un domaine spécifique (ex : médical, sportif, ...). La tâche de Question-Réponse est à priori

plus facile sur un domaine fermé que sur un domaine ouvert. En effet, dans un domaine fermé, les systèmes peuvent utiliser des connaissances spécifiques au domaine pour répondre à l'ensemble limité de questions. En revanche, du fait de l'infinité et la diversité des questions, les systèmes à domaine ouvert ont besoin de beaucoup plus d'informations pour extraire les réponses.

BASEBALL [Green Jr et al., 1961] a été le premier système de Question-Réponse développé. Bien que primitif, BASEBALL est un système à domaine fermé qui permet de répondre à beaucoup de questions simples en langage naturel sur les matchs de baseball d'une saison de la ligue américaine. Étant donné qu'il a été élaboré dans les années 1960, et que le développement du TALN était encore limité à cette époque, les questions étaient posées sous forme de cartes perforées, avec un "trou" pour l'élément recherché. Il suffisait ensuite au système d'aller à partir de sa représentation interne des connaissances trouver la réponse. Pour garder une certaine simplicité dans le questionnement, les questions avec plusieurs clauses dépendantes ou la logique booléenne (et, ou, ...) n'étaient pas permises. On présente un exemple du fonctionnement de BASEBALL à la figure 1.1. Il est remarquable que le fonctionnement de ce système précurseur n'est pas éloigné de systèmes plus récents que nous décrirons.

Fig. 1.1. Exemple de fonctionnement de BASEBALL [Green Jr et al., 1961]

Soit la question: `Where did the Red Sox play on July 7?`

Après traitement on obtient la structure syntaxique suivante:

`[Where] did [the Red Sox] play (on [July 7])?`

Avec les groupes nominaux entre crochets et les locutions prépositionnelles entre parenthèses. En effectuant une recherche dans une table de conversion on peut convertir cela en paire "attribut = valeur", la question devient donc:

```
Place = ?
Team = Red Sox
Month = July
Day = 7
```

Il reste ensuite au système à trouver l'élément manquant dans sa base de connaissances.

BASEBALL présente des limites au niveau de son adaptabilité et de son maintien dû à sa représentation interne des connaissances. Il est facile de réaliser une analyse sémantique sur un très petit domaine, cependant cela n'est pas encore applicable à grande échelle. C'est pourquoi très vite, la communauté du TALN s'est intéressée aux systèmes à domaine ouvert.

LUNAR [Woods, 1973] est un système conçu pour répondre aux questions de géologues lunaires concernant l'analyse et la composition chimique des roches et du sol lunaire obtenus lors des missions Apollo. Des questions telles que *What is the average concentration of aluminum in high-alkali rocks?* sont d'abord analysées à l'aide de règles syntaxiques (une grammaire). Ensuite à l'aide de règles de dérivation sémantiques on interprète le sens de la question [Woods, 1978]. Finalement, on interroge la base de données afin de récupérer la réponse souhaitée. Le système LUNAR possède dans sa base de données 13 000 entrées. L'évaluation a montré que 78% des questions de test reçoivent une réponse correcte. LUNAR intègre dans sa représentation langagière les quantificateurs (par exemple des opérateurs pour les mots "pour tous" ou "en moyenne"), cela ressemble à la logique de premier ordre utilisée de nos jours. Ainsi BASEBALL et LUNAR ont ouvert la voie à de nombreuses recherches pour développer des bases de connaissance pour des domaines très spécifiques, entraînant l'apparition des systèmes experts. Ressemblant beaucoup aux systèmes de QA modernes, les systèmes experts sont des programmes informatiques qui essaient d'imiter les décisions que peut avoir un expert humain. La principale composante des systèmes experts est la base de connaissances qui consiste en un ou plusieurs documents préparés et rassemblés soigneusement par un ou plusieurs experts du domaine.

D'autre part la communauté TALN s'est intéressée à construire des systèmes visant spécifiquement à améliorer l'interaction homme-machine. Bien que ce ne sont pas directement des systèmes de QA à proprement dit, ils ont influencé la communauté de TALN et par la même occasion le Question-Réponse. ELIZA [Weizenbaum et al., 1966] développé dans les laboratoires du Massachusetts Institute of Technology (MIT) est le premier exemple de traitement primitif du langage naturel. C'est le premier système conçu pour permettre la communication linguistique entre l'homme et l'ordinateur, en simulant l'interaction que

pourrait avoir un patient humain avec son psychothérapeute (ici ELIZA). De nombreux systèmes visant à l'amélioration de la communication personne-machine ont par la suite été développés. S'apparentant plus à un système de QA, SHRDLU [Winograd, 1971] est un système capable d'avoir des dialogues simples (possiblement interrogatifs) avec l'utilisateur sur un petit monde d'objets. Une caractéristique notable de ce système est qu'il inclut un bloc de mémoire de base fournissant le contexte.

Ainsi, au fil du temps la communauté TALN développe des systèmes de QA intégrant de plus en plus la dimension de l'interaction personne-machine. Contrairement aux systèmes de QA primitifs, les questions posées et les réponses retournées vont progressivement prendre "une forme humaine".

START [Katz, 1997] (*SynTactic Analysis using Reversible Transformations*) est le premier système de Question-Réponse à domaine ouvert disponible en ligne. Pour pouvoir répondre aux questions, START utilise un ensemble d'informations structurées, semi structurées ainsi que non structurées. START peut répondre à des millions de questions dans les domaines des lieux, des films, des personnes, des définitions de dictionnaire, etc. Pour pouvoir gérer la très grande diversité d'informations dont il a besoin, START utilise des annotations paramétrées (< sujet, relation, objet >) pour stocker les données. Aussi, il utilise un processus semi-automatisé de génération d'annotations en langage naturel [Katz et al., 2006]. Finalement, START utilise Omnibase [Katz et al., 2002] et IMPACT [Borchardt, 1992] pour stocker et interroger des données de manière efficace. Ainsi, en combinant les annotations en langage naturel et le traitement de la langue au niveau des phrases, START obtient de bonnes performances.

Développé par Wolfram Research, le moteur de recherche Wolfram Alpha [Wolfram, 2009] est un autre outil très utilisé intégrant une composante de Question-Réponse à domaine ouvert. Wolfram Alpha répond aux questions factuelles en calculant la réponse ou en la cherchant dans une vaste base de données organisée. Les données sont recueillies auprès de diverses sources, telles que le World Factbook, le United States Geographical Survey et le World Wide Web, etc. Derrière le système Wolfram Alpha, il y a le moteur Mathematica [Wolfram, 2007] développé également par Wolfram Research. Mathematica

est un programme utilisant les calculs mathématiques symboliques pour effectuer des calculs techniques complexes. Wolfram Alpha repose principalement sur une base de données organisée pour calculer ou rechercher des réponses. Comme beaucoup d'informations sont facilement disponibles sur internet, Wolfram Alpha à l'aide de Mathematica, implémente des méthodes et des algorithmes pour gérer cette quantité de données et d'ainsi les rendre utilisables.

Les deux derniers systèmes sont dans la mouvance du très populaire système WATSON [Ferrucci et al., 2010]. Le succès du système Watson à mis en lumière les progrès du Question-Réponse, ou plus généralement de l'Intelligence Artificielle (IA), du TALN et de la RI. En gagnant en 2011 au célèbre quiz télévisé de culture générale "Jeopardy!", le système d'IBM marque un tournant dans la recherche en Question-Réponse sans pour autant la freiner. La clé du succès de Watson vient du fait qu'il analyse dans un premier temps la question afin de déterminer son sujet à l'aide du système de règles *Slot Grammar* [McCord et al., 2012] qui permet de mettre en relation les mots afin d'isoler le sujet de la question. Une fois la question interprétée, Watson recherche des réponses dans son large ensemble de sources (Wikipédia, diverses presses, etc). Après avoir généré un ensemble de réponses potentielles, Watson attribue un score de confiance à chacune des réponses candidates à l'aide d'éléments de preuve récupérés dans ses diverses sources avant de choisir celle avec le score le plus élevé. Watson utilise plusieurs modèles, chacun ciblant un ou plusieurs types de questions de type "Jeopardy!". Par exemple, il y a un extracteur pour répondre aux questions sur les présidents américains, un détecteur de jeux de mots pour les faits que les humains considèrent comme intéressants. Les questions de style "Jeopardy!" sont posées au travers de phrases déclaratives: "les indices". Les candidats doivent ensuite déterminer l'élément manquant afin de formuler une question pour laquelle l'indice est la réponse. Par exemple pour la question "`In 1903, with presidential permission, Morris Michtom began marketing` *`these`* `toys.`" les candidats doivent répondre "`What are Teddy Bears?`". Le système Watson est un système hautement optimisé pour répondre aux questions dans le temps imparti et exceller dans le jeu "Jeopardy!". Grâce au succès de Watson a "Jeopardy!", IBM a obtenu un contrat pour adapter Watson au secteur de

la santé pour le gouvernement américain et a produit en avril 2015 le système Watson Health.

De nos jours, avec les récents progrès de l'apprentissage profond [Goodfellow et al., 2016] la plupart des systèmes développés utilisent les réseaux neuronaux. C'est le cas de nombreux assistants intelligents qui ont été développés et ne cessent d'être améliorés. On note que ces systèmes sont également basés sur des méthodes plus traditionnelles. En effet, SIRI par exemple utilise le moteur de recherche précédemment décrit: Wolfram Alpha. Ainsi, en plus de répondre aux questions comme le fait un système classique de Question-Réponse, Siri (Apple), Cortana (Windows), Alexa (Amazon), Google Now (Google), intègrent la reconnaissance vocale. Ceci donne à l'utilisateur l'impression d'interagir avec la machine. On peut désormais poser oralement n'importe quelle question à ces systèmes (ex: "Quel temps fait-il?", "Qui a gagné la dernière Ligue 1?") et obtenir une réponse orale ou écrite dépendamment du choix de l'utilisateur. Dans le but, de remplir pleinement le rôle d'assistant, ils intègrent également d'autres fonctionnalités connexes, par exemple la possibilité sur ordre vocale d'ajouter automatiquement un rendez-vous à l'agenda de l'utilisateur. Cependant ces systèmes sont encore imparfaits, et la technologie qu'ils utilisent, bien qu'elle offre de très bons résultats, complique la compréhension de ce qu'ils capturent réellement. Aussi, l'utilisation de réseaux de neurones oblige à avoir suffisamment de données et de ressources pour développer de tels systèmes.

Bien que la tâche de Question-Réponse a été longuement étudiée, on voit au travers de son évolution et de ses applications que celle-ci est encore d'actualité. Malgré le fait que les modèles neuronaux offrent de bonnes solutions à de nombreux problèmes de QA, ce sont des modèles complexes. Dans ce mémoire nous souhaitons construire un système de Question-Réponse non neuronal. Ainsi, en produisant un système à taille contrôlable de QA et en analysant ses lacunes, nous pouvons mieux mettre en valeur ce que des modèles neuronaux sont capables de faire.

1.2. *Benchmarks* disponibles

Au fil du temps, de nombreux jeux de données (des *benchmarks*) ont étés rendus disponibles afin qu'on puisse développer, évaluer et comparer les systèmes. Il y a eu une floraison de *benchmarks* durant la dernière décennie grâce à l'engouement autour de l'apprentissage profond. L'intérêt d'avoir de nombreux jeux de données (*datasets*) est de pouvoir développer des systèmes applicables à différent domaines. On distingue chaque *dataset* par sa tâche, par son ou ses domaines de questionnement ainsi que par le niveau de difficulté des questions posées. Afin de choisir le *dataset* d'étude, il est important de mieux connaître les principaux *benchmarks*. Du au fait que la très grande majorité des *benchmarks* portent sur des questions factuelles de compréhension de lecture, les autres *benchmarks* ne seront pas étudiés ici. On appelle *benchmarks* de compréhension de lecture l'ensemble de *benchmarks* composés de questions posées sur un ou plusieurs documents ou paragraphes que le système doit généralement "comprendre" afin d'en extraire ou de générer une ou plusieurs phrases ou étendues de texte (*span*) pour répondre.

Crée par Microsoft Research, MCTest [Richardson et al., 2013] est un *dataset* contenant 500 histoires fictives, chacune d'elle associée à un ensemble de questions à choix multiples. Ainsi, le système doit déterminer l'unique bonne réponse parmi quatre réponses candidates proposées. Pour construire ce *dataset*, on a demandé à un panel de personnes de rédiger des histoires (de 150 à 300 mots) avec trois réponses candidates plausibles et une seule réponse attendue. Bien que les histoires et les questions n'ont pas de sujet imposé, elles sont cependant destinées à des enfants du primaire ce qui restreint le niveau de difficulté. Un exemple d'histoire et de questions est présenté à la figure 1.2. Bien que les données de ce *dataset* sont fiables, les 500 textes et les 2000 questions de MCTest n'est pas suffisant. En effet cela rend plus difficiles la mise au point d'approches neuronales si l'on ne fait usage que de ces ressources pour l'entraîner ou s'ajuster.

Sur le même modèle que MCTest, le jeu de données RACE [Lai et al., 2017] est un jeu de données de compréhension de lecture d'examens d'anglais destinés à des collégiens et lycéens chinois. RACE est bien plus grand que MCTest, puisqu'il contient plus de 28 000 passages et près de 100 000 questions à choix multiples (4 choix possibles). La qualité des

Fig. 1.2. Exemples de questions-réponses sur MCTest [Richardson et al., 2013]

Histoire: James the Turtle was always getting in trouble. Sometimes he'd reach into the freezer and empty out all the food. Other times he'd sled on the deck and get a splinter. His aunt Jane tried as hard as she could to keep him out of trouble, but he was sneaky and got into lots of trouble behind her back.
One day, James thought he would go into town and see what kind of trouble he could get into. He went to the grocery store and pulled all the pudding off the shelves and ate two jars. Then he walked to the fast food restaurant and ordered 15 bags of fries. He didn't pay, and instead headed home.
His aunt was waiting for him in his room. She told James that she loved him, but he would have to start acting like a well-behaved turtle.
After about a month, and after getting into lots of trouble, James finally made up his mind to be a better turtle.

Question 1 : What is the name of the trouble making turtle?
A: Fries **B:** Pudding **C:** James **D:** Jane

Question 2 : What is the name of the trouble making turtle?
A: pudding **B:** fries **C:** food **D:** splinters

Question 3 : What is the name of the trouble making turtle?
A: his deck **B:** his freezer **C:** a fast food restaurant **D:** his room

Question 4 : What is the name of the trouble making turtle?
A: went to the grocery store **B:** went home without paying **C:** ate them
D: made up his mind to be a better turtle

modèles est évaluée en fonction de la précision des réponses retournées sur l'ensemble des examens de collège (RACE-m), l'ensemble des examens de lycée (RACE-h) et sur l'ensemble total (RACE). On présente en figure 1.3 un exemple de question posée sur Race.

Fig. 1.3. Exemple de question-réponse sur RACE [Lai et al., 2017]

Passage : Do you love holidays but hate gaining weight? You are not alone.
Holidays are times for celebrating. Many people are worried about their weight.
With proper planning, though, it is possible to keep normal weight during the holidays. The idea is to enjoy the holidays but not to eat too much. You don't have to turn away from the foods that you enjoy.
Here are some tips for preventing weight gain and maintaining physical fitness:
Don't skip meals. Before you leave home, have a small, low-fat meal or snack. This may help to avoid getting too excited before delicious foods.
Control the amount of food. Use a small plate that may encourage you to ''load up''. You should be most comfortable eating an amount of food about the size of your fist.
Begin with soup and fruit or vegetables. Fill up beforehand on water-based soup and raw fruit or vegetables, or drink a large glass of water before you eat to help you to feel full.
Avoid high-fat foods. Dishes that look oily or creamy may have large amount of fat. Choose lean meat. Fill your plate with salad and green vegetables. Use lemon juice instead of creamy food.
Stick to physical activity. Don't let exercise take a break during the holidays. A 20-minute walk helps to burn off extra calories.

Question: Which of the following statements is WRONG according to the passage?

A: You should never eat delicious foods.

B: Drinking some water or soup before eating helps you to eat less.

C: Holidays are happy days but they may bring you weight problems.

D: Physical exercise can reduce the chance of putting on weight.

Les *benchmarks* de style Cloze sont une variante de la tâche de compréhension de lecture. Ici, on présente un texte avec un ou plusieurs mots manquants afin que le système puisse les prédire en fonction de leur contexte et du reste du texte. Le Story Cloze test [Mostafazadeh et al., 2017] est un ensemble de données composé d'environ 4000 histoires mesurant en moyenne quatre phrases. Le but est de proposer au système pour chaque histoire deux fins

Fig. 1.4. Exemples d'histoires du Story Cloze test [Mostafazadeh et al., 2017]

Histoire: `Karen was assigned a roommate her first year of college. Her roommate asked her to go to a nearby city for a concert. Karen agreed happily. The show was absolutely exhilarating.`

Bonne fin: `Karen became good friends with her roommate.`

Mauvaise fin: `Karen hated her roommate.`

Histoire: `Gina misplaced her phone at her grandparents. It wasn't anywhere in the living room. She realized she was in the car before. She grabbed her dad's keys and ran outside.`

Bonne fin: `She found her phone in the car.`

Mauvaise fin: `She didn't want her phone anymore.`

possibles. En fonction de l'histoire, il doit déterminer laquelle des deux fins est la bonne. On présente en figure 1.4 des exemples issus du Story Cloze test.

Publié en 2018, CNN / Daily Mail [Hermann et al., 2015] est un autre *dataset* de style Cloze créé à partir d'articles de nouvelles de CNN et du Daily Mail. Ainsi pour un passage donné, on a remplacé les entités coréférentes par un unique marqueur d'entité @entityn où n est un index distinct. Les questions sont posées sous forme de phrases déclaratives, dans lesquelles on a omis l'entité à retrouver. Le système est donc chargé d'inférer l'entité manquante en fonction du contenu de l'article correspondant. On présente en figure 1.5 un exemple issu de CNN / Daily Mail.

Il y a d'autres *benchmarks* de style Cloze, par exemple CliCR [Suster and Daelemans, 2018] pour lequel il faut compléter des phrases en s'appuyant sur le contenu de rapports cliniques. Bien que les *benchmarks* de style Cloze permettent de mesurer la compréhension contextuelle des systèmes, ils ne concentrent majoritairement pas l'attention de la communauté intéressée au QA. Du à la spécificité de la tâche, les systèmes ne sont pas facilement généralisables et utilisables pour les besoins communs.

Fig. 1.5. Exemple de question-réponse sur CNN / Daily Mail [Hermann et al., 2015]

Passage: `(@entity4) if you feel a ripple in the force today , it may be the news that the official @entity6 is getting its first gay character . according to the sci-fi website @entity9 , the upcoming novel ‘‘ @entity11 ‘‘ will feature a capable but flawed @entity13 official named @entity14 who ‘‘ also happens to be a lesbian . ‘‘ the character is the first gay figure in the official @entity6 - the movies , television shows , comics and books approved by @entity6 franchise owner @entity22 - according to @entity24 , editor of ‘‘ @entity6 ‘‘ books at @entity28 imprint @entity26 .`

Question : `characters in " @placeholder " movies have gradually become more diverse`

Réponse: `@entity6`

NewsQA [Trischler et al., 2016] est un jeu de données de compréhension de lecture composé de plus de 100 000 paires de questions-réponses générées par un panel d'humain sur plus de 10 000 articles de nouvelles de CNN. Les réponses sont des *spans* issus de leurs articles correspondants. Elles peuvent donc être de taille arbitraire. Pour certaines questions, il n'est pas possible d'identifier la réponse dans l'article associé, car il n'y a pas l'information demandée. Un exemple d'une question issue de NewsQA est présenté en figure 1.6.

Publié en 2016 par l'université Stanford, la version 1.1 de SQuAD [Rajpurkar et al., 2016] est un jeu de données composé de plus de 100 000 questions réalisées par un panel d'humains sur un ensemble de plus de 10 000 articles de Wikipédia. Les réponses correspondent à des segments de texte (des étendues ou *spans*) identifiés par un panel d'humain. Elles sont extraites des articles sur lesquels les questions sont posées. SQuAD est divisé en trois parties: "train", "dev" et "test". Les tranches "train" et "dev" sont utilisées lors de la construction des systèmes. Non disponible et réservée à l'évaluation officielle des systèmes, la tranche "test" est accessible uniquement lorsque le système est soumis à la plateforme dédiée[1]. Contrairement à "train" qui dispose d'une seule réponse pour chacune des questions

[1]https://rajpurkar.github.io/SQuAD-explorer/

Fig. 1.6. Exemple de question réponse sur NewsQA [Trischler et al., 2016]

Article: `MOSCOW, Russia (CNN) - Russian space officials say the crew of the Soyuz space ship is resting after a rough ride back to Earth. A South Korean bioengineer was one of three people on board the Soyuz capsule. The craft carrying South Korea's first astronaut landed in northern Kazakhstan on Saturday, 260 miles (418 kilometers) off its mark, they said. Mission Control spokesman Valery Lyndin said the condition of the crew - South Korean bioengineer Yi So-yeon, American astronaut Peggy Whitson and Russian flight engineer Yuri Malenchenko - was satisfactory, though the three had been subjected to severe G-forces during the re-entry. [...]`

Question : `Where did the Soyuz capsule land?`

Réponse: `northern Kazakhstan`

posées, les tranches "dev" et "test" en contiennent au moins trois. Ceci permet d'avoir une meilleure précision lors de l'évaluation, car il se peut que la réponse apparaisse à divers endroits ou de diverses façons dans le texte. C'est pourquoi il est préférable d'avoir un ensemble de spans-réponses pour l'évaluation des systèmes. À la figure 1.7 on présente un exemple de questions-réponses issues de la tranche "dev" de SQuAD version 1.1.

Depuis 2018, la version 2.0 de SQuAD [Rajpurkar et al., 2018] est disponible. Celle-ci est un élargissement de la version 1.1 de SQuAD avec un ajout de 50 000 questions plausibles dont on ne peut répondre avec le texte associé. Le système doit dans ce cas détecter qu'il n'est pas en mesure de répondre.

Ainsi, le très grand nombre de *datasets* et de systèmes proposés témoigne du réel engouement autour du Question-Réponse. Hormis pour MCTest, la tendance actuelle pour les autres *datasets* présentés est d'utiliser des méthodes neuronales pour résoudre leur tâche respective. Si ces systèmes connaissent un succès impressionnant, il n'en reste pas moins qu'il n'est pas facile à comprendre comment une réponse a été fournie. De fait, de récentes études [Jia and Liang, 2017] mettent en avant le fait que les réseaux neuronaux ont une compréhension superficielle des textes. En particulier sur SQuAD, leur étude montre qu'il

Fig. 1.7. Exemple de questions réponses sur SQuAD v. 1.1

Paragraphe: The Broncos took an early lead in Super Bowl 50 and never trailed. Newton was limited by Denver's defense, which sacked him seven times and forced him into three turnovers, including a fumble which they recovered for a touchdown. Denver linebacker Von Miller was named Super Bowl MVP, recording five solo tackles, $2\frac{1}{2}$ sacks, and two forced fumbles.

Question 1: How many tackles did Von Miller accomlish by himself in the game?
Réponses: five, five, five
Question 2: Who was limited by Denver's defense?
Réponses: Newton was limited by Denver's defense, Newton, Newton
Question 3: Who was the Most Valuable Player of Super Bowl 50?
Réponses: Von Miller, Von Miller, Von Miller

est facile de tromper ces systèmes lorsqu'on ajoute dans les paragraphes une phrase fictive proche de la question sans en changer le sens global du texte. Dès lors, les performances de ces systèmes sont fortement diminuées.

C'est dans ce contexte que nous allons tenter de construire un système de Question-Réponse basé sur les *features* sur SQuAD version 1.1. Le seul système basé sur les *features* réalisé sur SQuAD version 1.1 est le modèle de régression logistique proposé par [Rajpurkar et al., 2016] expliqué à la section 3.4.2. Bien qu'il sera utilisé comme référence dans notre étude, ce système est critiquable car il utilise 180 millions de *features* hautement lexicalisées ce qui nécessite d'importantes ressources ainsi que du temps pour l'entraîner. Aussi, le modèle proposé par [Rajpurkar et al., 2016] obtient des performances nettement inférieur à l'état de l'art. On va donc vouloir ici avec nos modèles dépasser les performances obtenues par les auteurs de SQuAD. Cependant, dans le but d'obtenir un modèle généralisable à d'autres datasets, nous allons restreindre la quantité de *features* utilisées ainsi que le niveau de lexicalisation du système.

L'intérêt de prendre SQuAD v1.1 comme *datasets* d'études est qu'il s'agit d'un des *datasets* les plus étudiés en Question-Réponse. Il sera donc facile de comparer nos approches aux approches état de l'art.

Chapter 2

Notions théoriques

Ce chapitre est un rappel d'éléments théoriques utilisés afin de permettre une meilleure compréhension du reste du mémoire. On décrit dans un premier temps les méthodes de prétraitement du texte. Ensuite, on décrit des méthodes visant à récupérer de l'information au niveau des phrases. Enfin, on décrira les modèles utilisés.

2.1. Tokenization des textes

Pour pouvoir travailler sur les textes, il nous faut les découper. Ainsi, un texte va être découpé en phrases; une phrase va être découpée en mots appelés *tokens*. Dans le cas de l'anglais, pour réaliser ce découpage on utilise principalement les espaces et la ponctuation. Pour permettre une meilleure compréhension du mémoire l'utilisation du mot "mot" fait référence à *token*.

2.2. Normalisation des mots

La lemmatisation et le stemming sont deux techniques permettant de normaliser les mots en les réduisant à des formes plus simples. Cette réduction de dimensionnalité des mots est souvent très utilisée.

2.2.1. Le stemming

Le *stemming* (ou racinisation en français) est un processus qui vise à ramener les mots à une forme "racine". Pour cela, on retire des mots tous les accords, déclinaisons ou dérivations en éliminant la plus part du temps les suffixes ou les préfixes des mots. Par

exemple, play est le stem associé aux mots "playing", "plays" et "played". Même si cela fonctionne bien il y a cependant une limite aux stemming. Étant donné que l'analyse est basée sur les préfixes et suffixes, des mots comme "studies" et "studying" seront ramenés à deux formes différentes: "studi" et "study".

2.2.2. La lemmatisation

Contrairement au stemming, la lemmatisation prend en compte l'analyse morphologique des mots. Pour pouvoir relier un mot à son lemme, on va donc se baser sur des dictionnaires. Dans le cas où le mot n'appartient pas au dictionnaire de lemmes, celui-ci sera inchangé. Ainsi, "am", "are" et "is" ont pour lemme "be" tandis que "studies" et "studying" ont eux "study" comme lemme.

2.2.3. Vectorisation des mots

Le texte ne peut pas être donné directement aux algorithmes d'apprentissage machine traditionnels. En effet, étant donné que ces algorithmes fonctionnent sur des espaces vectoriels il est nécessaire de transformer les mots en vecteur. Chaque dimension du vecteur est appelée trait ou feature.
Les plongements de mots (*words-embeddings*) [Mikolov et al., 2013] est une méthode d'apprentissage de représentation des mots utilisée en autre en TALN. Cette technique permet de représenter chaque mot d'un dictionnaire par un vecteur de nombres réels encodant le mot et le contexte dans lequel il apparaît. Cela permet de capturer de l'information sémantique au niveau des mots. Ainsi les mots apparaissant dans des contextes similaires possèdent des vecteurs relativement proches. On peut donc s'attendre à ce que les mots chien et chat soient représentés par des vecteurs relativement proches dans leur espace de définition.
En plus de résoudre partiellement le problème de dimensionnalité, les words-embedings permettent de comparer et d'effectuer des opérations sur les mots. En effet, en plus de pouvoir mesurer la similarité entre deux mots, on peut maintenant les additionner ou les soustraire. Ainsi l'équation de vecteurs suivante: roi - homme + femme doit retourner le vecteur correspondant au mot reine. Aussi (paris - france + italie) $\rightarrow$ rome.

2.3. Mesure TF-IDF

La mesure TF-IDF (*term frequency-inverse document frequency*) est une méthode de pondération fréquemment utilisée en RI et en particulier pour la fouille de textes. Cette mesure statistique permet d'évaluer l'importance d'un terme contenu dans un document, relativement à une collection ou un corpus. Le poids d'un mot est proportionnel au nombre d'occurrences de celui-ci dans le document et à sa fréquence dans l'ensemble du corpus. Bien que non utilisée dans ce mémoire, on présente ici la méthode TF-IDF utilisée par notre modèle de référence [Rajpurkar et al., 2016] (présenté en section 3.4.2).

2.4. Mesure de similarité

Dès lors que les mots ont été vectorisés, il est possible de manipuler ces vecteurs afin de les comparer et d'étudier leur proximité. Étant donné qu'il existe de nombreuses mesures pour comparer deux vecteurs, on décrit uniquement les plus utilisées par la communauté TALN.

La distance euclidienne (également appelée distance L^2) entre deux points dans l'espace de définition permet de mesurer leur proximité. Soit $\vec{x}, \vec{y}$ deux vecteurs de dimension n, on calcule leur distance euclidienne comme:

$$d(\vec{x},\vec{y}) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

Ainsi, la similarité euclidienne entre $\vec{x}$ et $\vec{y}$ est obtenue en faisant le ratio: $\frac{1}{d(\vec{x},\vec{y})}$.

La similarité cosinus permet de mesurer la similarité entre $\vec{x}$,$\vec{y}$ deux vecteurs de dimension n, en calculant le cosinus de l'angle θ qu'ils forment. Cette métrique est fréquemment utilisée en recherche d'information et en particulier dans la fouille de document.

$$cos\_sim(\vec{x},\vec{y}) = cos(\theta) = \frac{\vec{x} \cdot \vec{y}}{\|\vec{x}\| \cdot \|\vec{y}\|}$$

Le coefficient de Dice (également appelé Sørensen-Dice) mesure la similarité entre deux vecteurs $\vec{x}$, $\vec{y}$ de dimension n.

$$dice\_coef(\vec{x},\vec{y}) = \frac{2 \times (\vec{x} \cdot \vec{y})}{\|\vec{x}\|^2 \cdot \|\vec{y}\|^2}$$

La similarité de Jaccard est également souvent utilisée en recherche d'information; voici comment on la calcule:

$$Jaccard\_coef(\vec{x},\vec{y}) = \frac{(\vec{x} \cdot \vec{y})}{\|\vec{x}\|^2 \cdot \|\vec{y}\|^2 - (\vec{x} \cdot \vec{y})}$$

2.5. Part-of-Speech

L'étiquetage grammatical (PoS-tagging) correspond à associer une étiquette grammaticale à chaque mot d'une phrase. Par ce procédé, on distingue ainsi dans chaque phrase les verbes, les adjectifs, les pronoms, etc. Un exemple de PoS-tagging est présenté en table 2.1 .

Tab. 2.1. Exemple de PoS-tagging de la phrase : "And now for something completely different"

mots	**PoS-tag**	**description**
And	CC	conjonction de coordination
now	RB	adverbe
for	IN	préposition ou conjonction de subordination
something	NN	nom, singulier ou pluriel
completely	RB	adverbe
different	JJ	adjectif

De nombreux systèmes et corpus utilisent les étiquettes du Penn TreeBank [Marcus et al., 1994]. Dans ce mémoire, on utilise le PoS-tagger de SpaCy[1] qui embarque un modèle entraîné sur le corpus OntoNotes 5 [Weischedel et al., 2013].

[1] `https://spacy.io/`

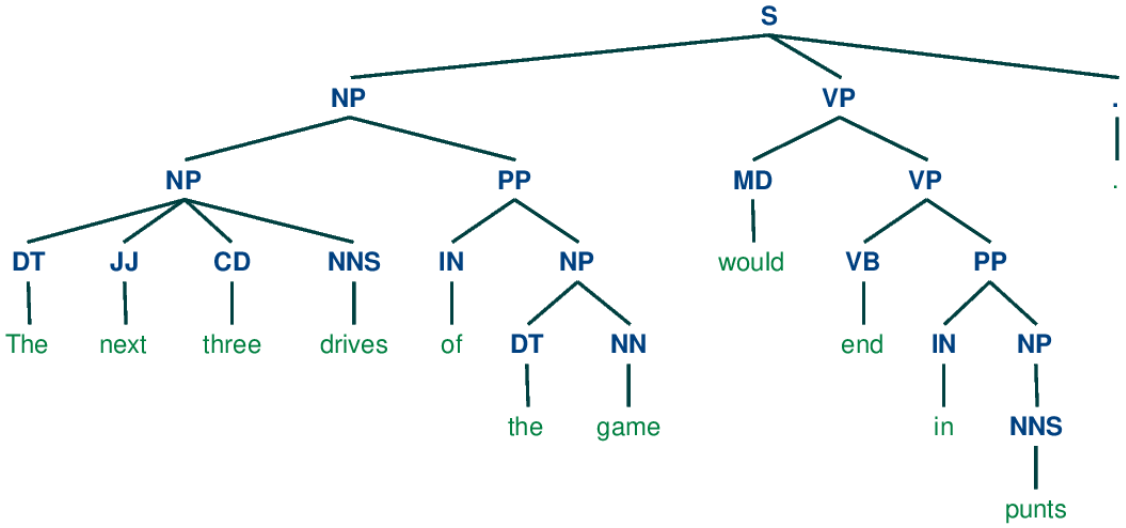
2.6. Le *parsing*

L'analyse syntaxique ou *parsing* d'une phrase consiste à mettre en évidence sa structure syntaxique. On présente ici deux types d'analyses populaires.

2.6.1. L'analyse en constituants

Un arbre d'analyse en constituant divise un texte en syntagmes. Les noeuds non terminaux de l'arbre identifient des syntagmes, les noeuds terminaux sont les mots de la phrase. On présente en figure 2.1 un exemple d'arbre en constituants.

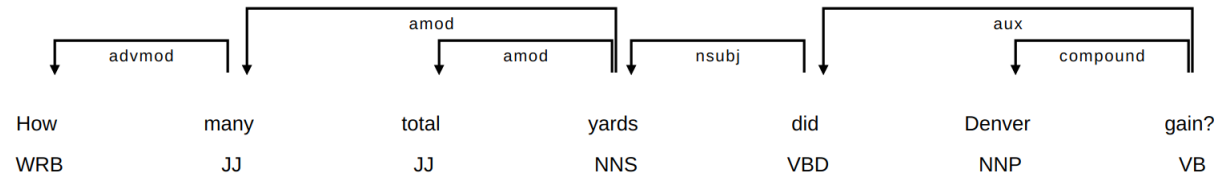
Fig. 2.1. Exemple d'arbre en constituants pour la phrase "The next three drives of the game would end in punts.", produit par l'analyseur de NLTK (question prise de SQuAD)



2.6.2. L'analyse en dépendances

Une analyse en dépendances connecte les mots en fonction des relations syntaxiques qu'ils entretiennent. On représente les relations entre les mots avec des arcs dirigés étiquetés du type de la relation. On présente en figure 2.2 un exemple d'analyse en dépendances ainsi que la description des relations identifiées en table 2.2.

Fig. 2.2. Exemple d'analyse en dépendance de la phrase "How many total yards did Denver gain?", produit par l'analyseur de SpaCy (question prise de SQuAD)



Tab. 2.2. Liste et description des relations identifiées dans la phrase de l'exemple 2.2

étiquette de la relation	**description**
advmod	relation identifiant un adverbe qui modifie le sens d'un mot
amod	relation identifiant une syntagme adjectival qui modifie le sens d'un syntagme nominal
nsubj	relation identifiant un syntagme nominal sujet syntaxique d'une clause
aux	relation identifiant l'auxiliaire d'une clause
compound	relation identifiant les mots composés

2.7. Extraction des entités nommées

L'identification d'entités nommées (ou *Named Entity Recognition* NER) est une tâche issue de la recherche d'information. Dans un texte, on identifie les éléments appartenant à des catégories d'intérêt tels que les noms, les organisations, les lieux, etc. Par exemple dans la phrase: "Genghis Khan came to power by uniting many of the nomadic tribes of Northeast Asia.", on identifie "Genghis Khan" comme une personne et "Northeast Asia" comme un lieu.

Dans ce mémoire on utilise le système identificateur d'entités nommées proposé par SpaCy entraîné sur OntoNotes 5. On présente en table 2.3, l'ensemble des types d'entités nommées reconnues par ce système ainsi que leur description.

Tab. 2.3. Liste des catégories reconnues par le système de reconnaissance d'entités nommées de SpaCy (entraîné sur OntoNotes 5)

Type	Description
PERSON	Les personnes, y compris les personnes fictives
NORP	Nationalités ou groupes religieux ou politiques.
FAC	Bâtiments, aéroports, autoroutes, ponts, etc.
ORG	Entreprises, agences, institutions, etc.
GPE	Pays, villes, états.
LOC	Emplacements hors GPE, chaînes de montagnes, plans d'eau.
PRODUCT	Objets, véhicules, aliments, etc. (pas les services.)
EVENT	Ouragans nommés, batailles, guerres, événements sportifs, etc.
WORK_OF_ART	Titres de livres, chansons, etc.
LAW	Documents nommés transformés en lois.
LANGUAGE	N'importe quelle langue nommée.
DATE	Dates ou périodes absolues ou relatives.
TIME	Durée inférieure à une journée.
PERCENT	Pourcentage, y compris "%".
MONEY	Valeurs monétaires, y compris l'unité.
QUANTITY	Mesures (poids ou distances)
ORDINAL	"Premier", "deuxième", etc.
CARDINAL	Les chiffres qui ne relèvent pas d'un autre type.

2.8. Régression logistique

La régression linaire est un modèle d'apprentissage statistique visant à prédire, à l'aide d'une fonction linéaire, la classe y d'un élément en fonction de son vecteur de paramètres $X \in \mathbb{R}^d$. Pour cela on calcule à partir de données d'entraînement les paramètres $\theta = \{W,b\}$, où $W \in \mathbb{R}^d$ est le vecteur de poids et $b \in \mathbb{R}$ le biais. Ainsi la fonction à déterminer est de la forme:

$$y = f_\theta(X) = W^T X + b$$

Appelée risque empirique, la perte totale sur l'ensemble d'entraînement est calculée comme la somme des pertes sur chacun des éléments de l'ensemble d'entraînement D_n (de taille n):

$$\hat{R} = \sum_{i=1}^{n} L(f_\theta(X^{(i)}),y^{(i)})$$

$L(y,t)$ est la fonction de coût qui permet de pénaliser le modèle proportionnellement à l'écart entre y la prédiction prédite et t la cible à prédire. On utilise généralement l'erreur quadratique :

$$L(y,t) = (y-t)^2$$

Il est souvent nécessaire d'induire une "préférence" pour certaines valeurs des paramètres plutôt que d'autres pour éviter le surapprentissage. On utilise une fonction de régularisation Ω qui pénalise plus ou moins certaines valeurs de paramètres et $\lambda \geq 0$ pour contrôler l'importance de cette régularisation. Le risque empirique régularisé est défini par:

$$\hat{R}_\lambda = \hat{R} + \lambda\Omega(\theta) = \sum_{i=1}^{n} L(f_\theta(X^{(i)}),y^{(i)}) + \lambda\Omega(\theta)$$

Il existe diverses fonctions de régularisation. On utilise généralement la régularisation L_2:

$$\Omega(\theta) = \Omega(W,b) = \sum_{j=1}^{d} w_j^2$$

Avec w_j le j$^{\text{ième}}$ coefficient de W. L'apprentissage revient donc à déterminer θ^* tel que:

$$\theta^* = \underset{\theta}{\operatorname{argmin}}\, \hat{R}_\lambda(f_\theta,D_n)$$

Le modèle de régression logistique correspond a une régression binomiale. Contrairement à la régression linéaire, ce modèle prédit la probabilité de l'évènement y en fonction d'un vecteur de paramètres X ($y \in \{0,1\}, X \in \mathbb{R}^d$):

$$P(y|X) = \frac{1}{1+e^{-W^T X+b}}$$

2.9. Machines à vecteurs de support

Les machines à vecteurs de support ou SVM (Support Vector Machine) sont des algorithmes qui consistent à trouver un hyperplan dans l'espace vectoriel d'entrée afin de pouvoir distinguer deux classes. Ainsi sur les données d'entraînement, un modèle SVM apprend une fonction linéaire afin de retourner une classe.

On définit un hyperplan par l'ensemble des points de coordonnées $\vec{x}$ satisfaisant l'équation suivante:

$$\vec{w} \cdot \vec{x} + b = 0$$

avec $\vec{w}$ un vecteur normal de l'hyperplan et b le biais correspondant au déplacement de l'hyperplan depuis l'origine de l'espace vectoriel.

Une marge entre un hyperplan (une droite en deux dimensions) et un ensemble de points correspond à la distance entre l'hyperplan et le plus proche point. On souhaite optimiser cette marge afin que celle-ci soit la plus grande. Étant donné deux classes linéairement séparables:

$$\exists w,b \,|\, \forall i \in \{1,\cdots,n\}\,, sign(w^T x_i + b) = t_i$$

où $t_i \in \{-1,1\}$ correspond à l'étiquette respective à chaque classe. Ce problème correspond donc une minimisation de $||\vec{w}||$ sous la contrainte que les points soient correctement classifiés. Plus petit est $||\vec{w}||$, plus la marge sera grande, améliorant les performances de l'algorithme.

Cependant dans la plupart des cas, les données ne sont pas linéairement séparables. Il faut donc utiliser une fonction de perte dans le but de minimiser le nombre d'erreurs que l'on tolère. La fonction de perte est définie comme :

$$max(0,1 - t_i(\vec{w} \cdot \vec{x_i} + b))$$

On pénalise une bonne classification par 0 et une mauvaise par $1 - t_i * (\vec{w} \cdot \vec{x_i} + b)$. De cette façon la pénalité attribuée à une mauvaise classification est proportionnelle à la distance entre le point et l'hyperplan. Le problème devient donc la minimisation de:

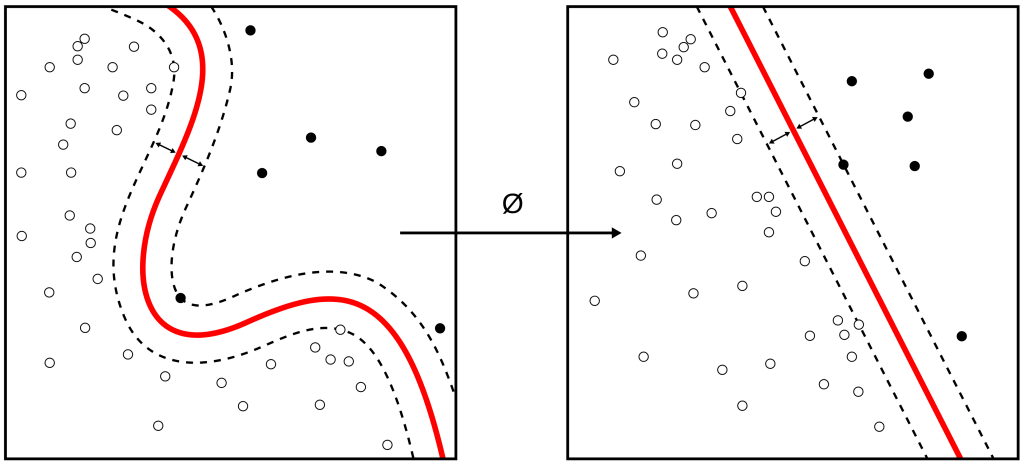
$$\frac{1}{n}\sum_{i=1}^{n}(max(0,1 - t_i(\vec{w} \cdot \vec{x_i} + b))) + \lambda||\vec{w}||^2$$

avec λ le paramètre déterminant "la souplesse" de la marge.

Pour contourner les contraintes de linéarité des modèles SVM, on utilise fréquemment l'astuce du noyau. En appliquant à l'aide d'une transformation non linéaire ϕ on projette l'espace de définition d'entrée dans un espace de plus haute dimension. Dans ce nouvel espace, l'algorithme peut trouver des corrélations non linaires dans l'espace d'entrée. Nous

savons par les propriétés mathématiques de projection d'espaces, qu'il est possible de directement calculer le produit scalaire dans l'espace transformé. Dès lors puisqu'il n'est plus nécessaire de calculer le mapping explicitement, on réduit la complexité des calculs. La figure 2.3 illustre un exemple de mapping ϕ.

Fig. 2.3. Exemple d'un mapping ϕ (image tirée de Wikipedia)



Il existe de multiples noyaux. Dans ce travail, on utilise un noyau populaire, le noyau RBF (appelé aussi Gaussien) défini par la formule suivante:

$$K_\sigma(a,b) = e^{-\frac{1}{2}\frac{||a-b||^2}{\sigma^2}} \Leftrightarrow K_\gamma(a,b) = \exp(-\gamma||a-b||^2)$$

avec $\sigma,\gamma > 0$ des hyperparamètres choisis pour ajuster la loi gaussienne (σ étant l'écart type).

2.10. Forêt aléatoire

L'algorithme de forêt aléatoire (*Random Forest*) construit un ensemble de prédicteurs A (classifieur ou régresseur) à partir de l'ensemble d'entraînement D_n. On fait ensuite voter l'ensemble de ces prédicteurs afin de retourner le vote majoritaire.

À partir de D_n on applique la méthode du bagging [Breiman, 1996] où l'on génère aléatoirement T sous-ensembles de données: $D_1,...,D_T$ de taille m à partir de D_n. Sur chaque sous-ensemble D_t on entraine un prédicteur $h_t = A(D_t)$. Ainsi y la prédiction du modèle ($y \in \{-1,1\}$) est définie par:

$$y = f(X) = \frac{1}{T}\sum_{t=1}^{T} h_t(X)$$

2.11. Réseau neuronal

Bien que celles-ci ne soient pas directement utilisées dans ce mémoire, il nous faut décrire brièvement l'architecture et les principes fondamentaux des méthodes neuronales pour une meilleure compréhension de certains systèmes. Un réseau de neurones consiste généralement en une couche d'entrées composées d'unités représentant les éléments d'entrée du problème (ici les mots). Ensuite, il y a une série de couches cachées composées d'unités appelées neurones. Plus un réseau à de couches cachées plus il est profond. Chacun des neurones est plus ou moins interconnecté aux autres dans le but de pouvoir dans la couche finale produire la sortie espérée. On présente en figure 2.4 le schéma d'un réseau neuronal classique.

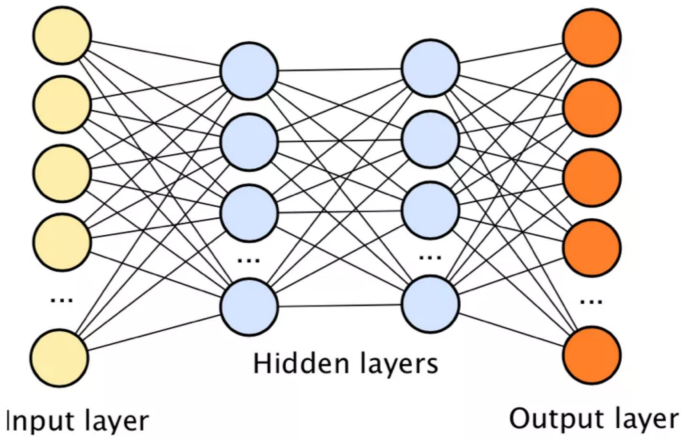


Fig. 2.4. Réseau neuronal classique

(source : `https://blog.eduonix.com/artificial-intelligence/deep-neural-networks-keras/`)

Pour entraîner le modèle afin que celui-ci retourne de bonnes prédictions, durant la phase d'entraînement, on utilise un algorithme de rétropropagation (appelé également descente de gradient). Ceci permet d'ajuster les poids des connexions entre les neurones. Ainsi les sorties retournées par le réseau vont tendre vers les sorties attendues pour les données d'entraînement.

Chapter 3

SQuAD

3.1. Étude préliminaire de SQuAD

Avant de pouvoir construire des modèles de Question-Réponse sur SQuAD il nous faut mieux le connaître. En particulier il faut nous intéresser aux questions, spans-réponses ainsi qu'aux paragraphes de SQuAD.

En moyenne, chaque paragraphe contient 735.8 caractères sur le train et 774.3 sur le dev. Le nombre moyen de mots par paragraphe est de 134.2 pour le train et 140.6 pour le dev. Dans chaque paragraphe il y a en moyenne 4.9 phrases pour le train et 5.0 pour dev. Les phrases du train contiennent en moyenne 27.1 et mesurent 147.7 caractères. Les phrases du dev ont elles 27.8 mots et 152.2 caractères. Ces informations sont obtenues grâce à la résultante d'un découpage réalisé à l'aide de la librairie TextBlob[1].

Il est important de connaître le type des questions que contient SQuAD. En effet, intuitivement certains types de question appellent certains types de réponses. Ainsi, connaître le type de question permet souvent de pouvoir établir des a priori sur le type de réponse attendu. Basée sur CoreNLP[2], la segmentation de questions proposée par [Rajpurkar et al., 2016] permet de classer automatiquement toutes les questions en huit types à l'aide de leur pronom interrogatif. Lorsque le système ne parvient pas à identifier le type d'une question

[1] `https://textblob.readthedocs.io/en/dev/`

[2] `https://stanfordnlp.github.io/CoreNLP`

on lui associe le type "*Other*". Voici le type ainsi que le détail explicatif de chacun de ces types :

- *What / Which NN[*]?*: questions avec le pronom interrogatif "quoi" ou "lequel/laquelle" portant sur l'ensemble des noms possibles (singulier, pluriel, propre).
  E.g., "Which NFL team represented the NFC at Super Bowl 50?", "What American actor is also a university graduate?".

- *What VB[*]?*: questions avec le pronom interrogatif "quoi" ou "lequel/laquelle" portant sur l'ensemble des verbes possibles (passé, gérondif, 3è forme, ...).
  E.g., "What covered the new field at Levi's Stadium?", "What happened to Dane?".

- *What name / is called?*: les questions portant sur la dénomination d'un sujet (une personne, un évènement...).
  E.g., "What is the name of a Bodhisattva vow?", "What is the jelly-like substance called?".

- *Who?*: les questions portant sur une personne.
  E.g., "Who was Robert's son?", "Who is responsible for axiomatic complexity theory?".

- *When / What year?*: questionnement sur une date ou évènement.
  E.g., "In what year did Harvey Martin die?", "When is the Wianki festival held?".

- *How much / many?*: les questions portant sur une durée ou une quantité.
  E.g., "How many balls did Josh Norman intercept?", "How much does a Probationer earn, initially?".

- *Where?* : les questions portant sur les lieux.
  E.g., "Where was Super Bowl 50 held?", "Where was Melanchthon at the time?".

- *How?*: les questions portant sur la manière.
  E.g., "How many turnovers did Cam Newton have?", "How far is Warsaw from the Baltic Sea?".

- *Other* : le reste des questions non identifiées.

  E.g.,"What is the AFC short for?", "In what city's Marriott did the Broncos stay?".

On présente en table 3.1 la statistique des divers types de questions dans chaque tranche du jeu de données (train, dev). On observe des distributions sur train et dev similaires.

Tab. 3.1. Statistique des divers types de question

	Train		**Dev**		**Train+Dev**	
Type de question	compte	%	compte	%	compte	%
What / Which NN[*]?	34215	39.1%	3976	37.6%	38191	38.9%
What VB[*]?	9801	11.2%	1367	12.9%	11168	11.4%
Who?	9428	10.8%	1185	11.2%	10613	10.8%
When / What year?	8528	9.7%	943	8.9%	9471	9.6%
How much / many?	5986	6.8%	757	7.2%	6743	6.9%
Where?	3714	4.2%	477	4.5%	4191	4.3%
How?	3399	3.9%	468	4.4%	3867	3.9%
What name / is called?	2820	3.2%	325	3.1%	3145	3.2%
Other	9708	11.1%	1072	10.1%	10780	11.0%
Total	87599		10570		98169	

En s'intéressant à la taille des spans-réponses on observe qu'ils sont courts. En effet en utilisant la librairie NLTK[3], on mesure la taille de ces spans en tokens. En moyenne les réponses mesurent 3.37 tokens pour le train et 3.59 pour le dev. On présente dans la figure 3.1 une représentation graphique de la distribution des tailles des spans-réponse. En plus de remarquer que les distributions sur le train et le dev sont similaires, on note sur le train que:

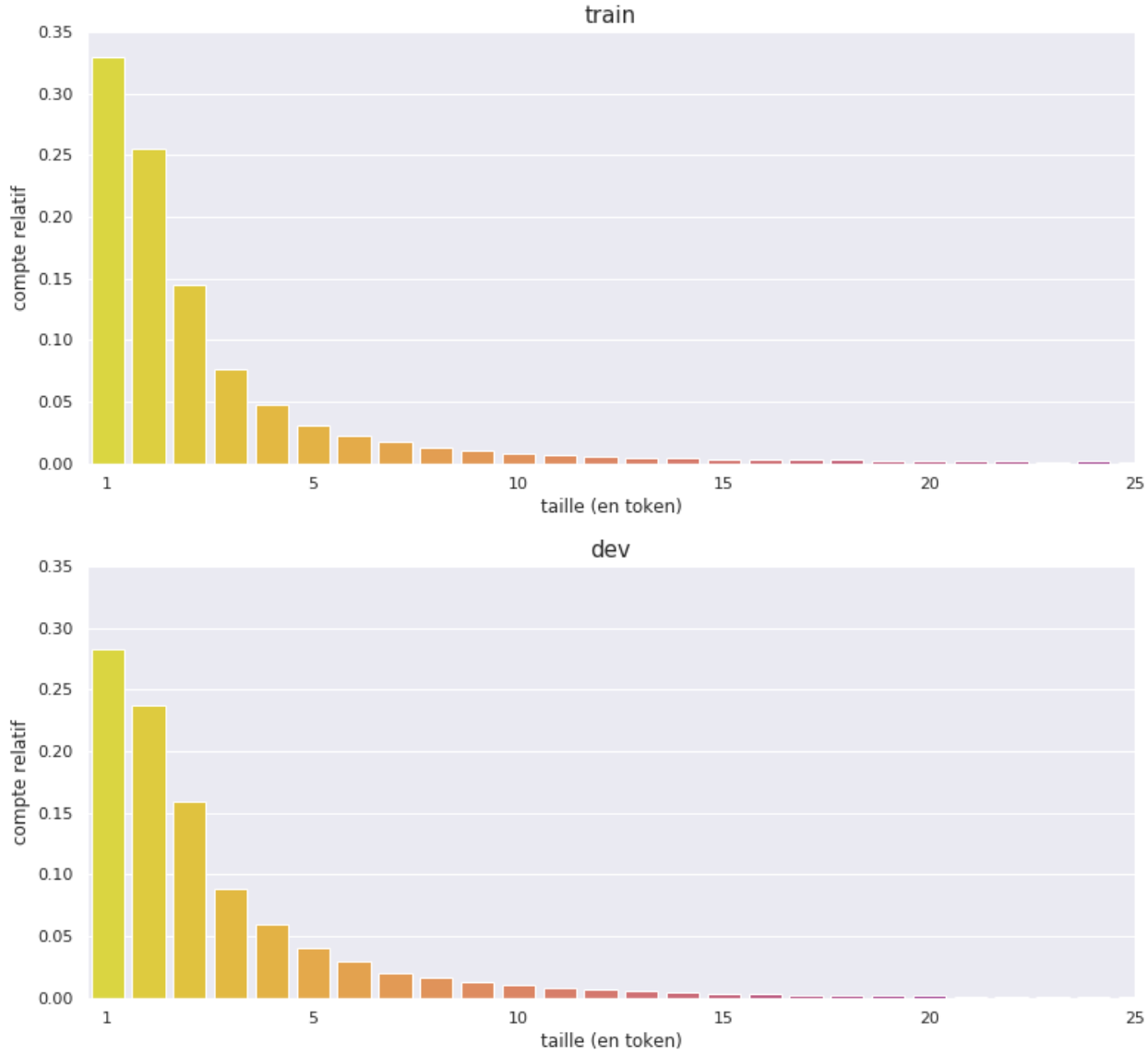
- 33 % des spans sont de taille 1
- 59 % des spans sont de taille 2 et moins
- 73 % des spans sont de taille 3 et moins
- 81 % des spans sont de taille 4 et moins

[3] `https://www.nltk.org/`

- 85 % des spans sont de taille 5 et moins

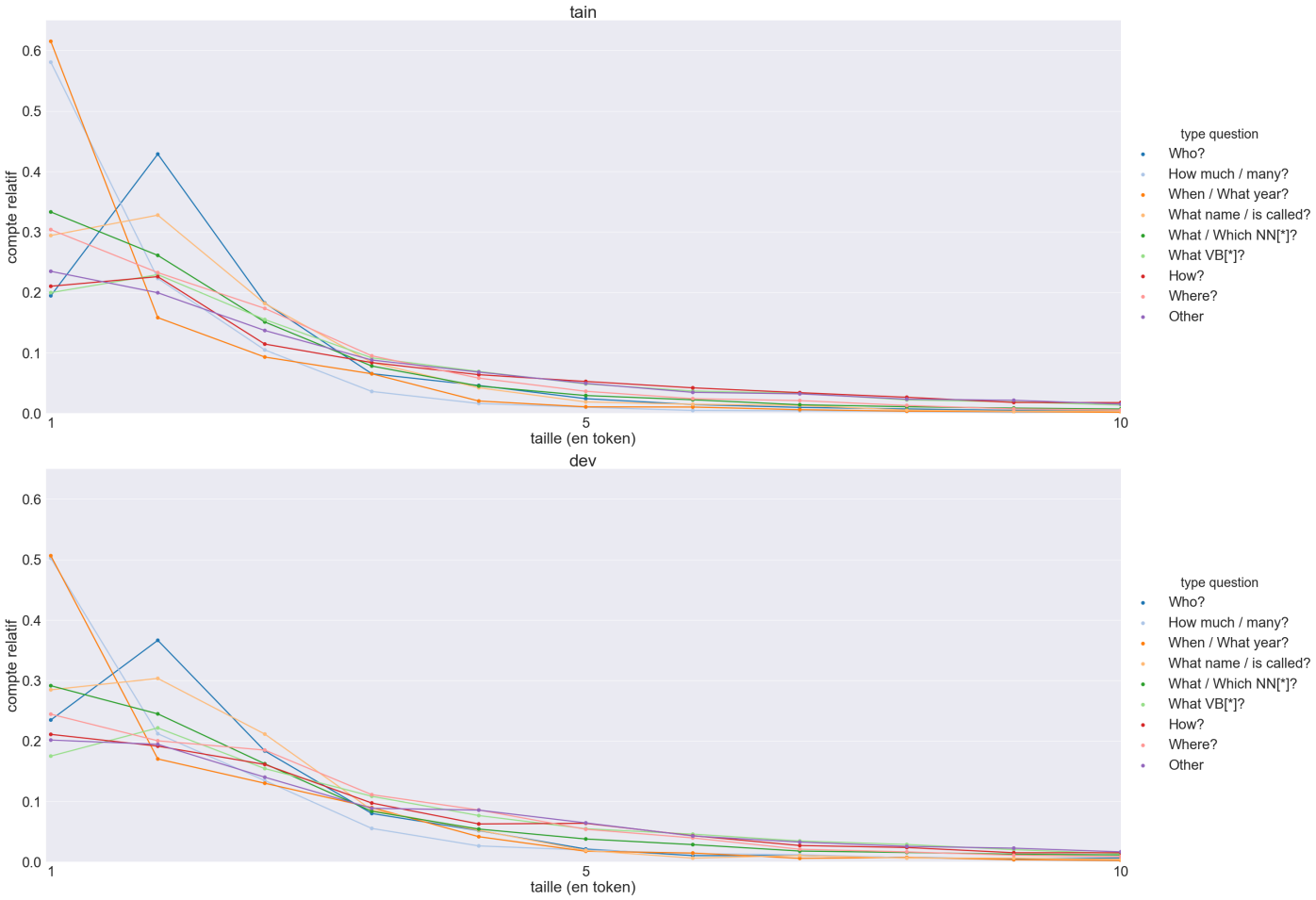
Il est donc plus payant de construire des modèles préférant les spans courts aux spans contenant beaucoup de tokens.

Fig. 3.1. Distribution de la taille des spans en tokens



Vu que les réponses fournies varient en fonction du type de question posée, on présente en figure 3.2 la distribution des tailles des spans-réponses en fonction du type de question. Au vu de ces résultats, les types de questions *How much/many?* et *When/What year?* se démarquent légèrement dans leur distribution. Cela coïncide avec les connaissances

Fig. 3.2. Distributions des tailles des réponses en fonction du type de question



linguistiques de base. En effet, très souvent on peut répondre à ce type de questions à l'aide d'un simple nombre (une quantité, une année). Mis à part pour ces deux types de questions, on ne peut pas réellement déterminer la taille attendue en fonction d'un type de question. Il faut donc trouver d'autres caractéristiques (features) plus discriminantes afin de sélectionner le bon span réponse.

On s'intéresse ensuite à calculer la position des spans-réponses dans le paragraphe . Pour pouvoir comparer les positions des spans indépendamment de la taille des paragraphes desquels ils sont issus, on va calculer une position relative (en caractères, en tokens ou en phrases). Les résultats sont présentés à la figure 3.3.

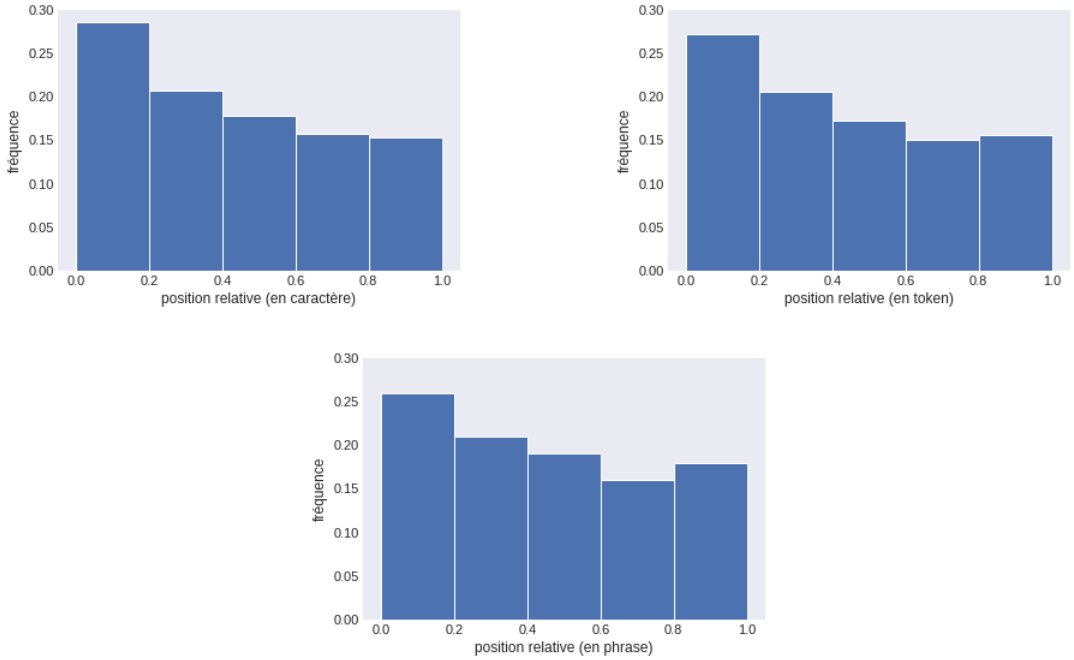


Fig. 3.3. Position relative au paragraphe de la phrase contenant la span

Ainsi, d'après les graphiques on observe ici qu'il est possible d'établir un *feature* sur la position du span dans le paragraphe.

3.2. Metrique d'évaluation

Afin de comparer les performances des systèmes, les auteurs de SQuAD proposent l'utilisation de deux métriques: le score de correspondances exactes (ou *exact match*) et le score F1. Chacune de ces deux métriques utilise la normalisation des spans décrite à la section 3.3 avant de réaliser les comparaisons.

Le score d'*exact match* est une métrique correspondant au pourcentage de prédictions valides sur le nombre total de prédictions réalisées. Une prédiction est comptée comme valide si elle se trouve dans l'ensemble des réponses de référence (annotées par des humains), plus spécifiquement si la prédiction est identique au caractère près à l'une des réponses de référence. De manière formelle, on peut définir le score d'*exact match* entre la chaîne x et

l'ensemble de référence Y comme:

$$exact\_match(x,Y) = \begin{cases} 1, & \text{si } \exists z \in Y | \ x \text{ et } z \text{ sont identiques.} \\ 0, & \text{sinon.} \end{cases}$$

Le score F1 est une mesure d'évaluation qui compare deux séquences de mots en évaluant leur chevauchement. Dans le cas de SQuAD, on considère chacune des réponses et des prédictions comme une suite de tokens. De manière formelle, on peut définir le score F1 entre une prédiction et une réponse par:

$$F_1 = 2 * \frac{\text{précision} * \text{rappel}}{\text{précision} + \text{rappel}}$$

avec:

$$\text{précision} = \frac{\#\text{ tokens communs entre la prédiction et la réponse}}{\#\text{ tokens de la prédiction}}$$

$$\text{rappel} = \frac{\#\text{ tokens communs entre la prédiction et la réponse}}{\#\text{ tokens de la réponse}}$$

Étant donné que dans dev et test il y a plusieurs réponses de référence, le score attribué à chaque prédiction correspond au maximum des scores F1 obtenus entre la prédiction et chacun des spans-réponses de référence. Pour évaluer un modèle, on réalise ensuite la moyenne des scores F1 de ses prédictions. L'avantage du score F1 est que sa conception permet de prendre en compte les réponses partielles car il est basé sur les métriques de précision et de rappel. Celles-ci sont des bons indicateurs pour évaluer la qualité des prédictions d'un système. La précision évalue la qualité des tokens retournés. Le rappel, permet d'évaluer le nombre de tokens pertinents retrouvés. Si une prédiction correspond exactement a un span-réponse référence, son score F1 sera de 1.

Dans le but de mieux comprendre les distinctions entre le score d'*exact match* et le score F1, on présente en table 3.2 quelque exemples de ces scores pour des prédictions et des réponses de référence issues de SQuAD.

Tab. 3.2. Exemple de scores d'*exact match* et de score F1 pour des prédictions sur SQuAD

Span prédit	**Spans-réponses de référence**	***Ex. match***	**F1**
1944	august 1944	0	**0.66**
	1 august 1944	0	0.5
	red army was nearing city	0	0
between rival lighting systems	competition between rival lighting systems	0	**0.88**
	lighting systems	0	0.66
	electrical distribution	0	0
transplastomic	genetically modified plants	0	0
	genetically modified crops	0	0
	transplastomic	**1**	**1**

3.3. Normalisation de l'écriture des spans-réponse

Lors de l'évaluation des systèmes et afin de pouvoir comparer les sorties obtenues avec les vrais spans-réponses il est nécessaire de normaliser l'écriture de l'ensemble des spans. En particulier, la mesure d'exact match (décrite à la section 3.2) compare deux séquences sur leur forme, deux spans ayant les mêmes mots sont notés comme différents si ils diffèrent, par exemple d'une majuscule. Pour résoudre, conformément à ce qui est fait dans la littérature, dans le script d'évaluation de SQuAD on réalise au préalable une normalisation d'écriture des spans:

- Dans un premier temps les articles "a", "an", "the" sont enlevés. Les réponses des systèmes peuvent diverger au niveau des articles, étant donné que ces derniers n'apportent pas ou peu d'information sémantique. Les éliminer permet donc de rendre l'évaluation plus précise. Aussi il n'est pas pertinent de pénaliser un système vis à vis d'un autre si leurs réponses diffèrent d'un article puisque là n'est pas l'intérêt de la tâche.
- Les paragraphes contenant parfois plusieurs espaces qui se suivent, il faut uniformiser l'écriture des span-réponse en éliminant le surplus d'espaces.
- En observant les réponses identifiées par le panel, on s'aperçoit que parfois, certains spans sont accolés à des ponctuations. On élimine donc des spans l'ensemble des ponctuations.

- Finalement, les spans sont convertis en minuscule.

Ainsi, dans le script officiel d'évaluation de SQuAD, on normalise d'abord l'ensemble des spans retournés par le système ainsi que l'ensemble des spans-réponses de référence avant de les comparer à l'aide des divers métriques (voir section 3.2). Notons que l'on utilise également cette même normalisation au sein de nos modèles pour prédire un span.

3.4. État de l'art sur SQuAD

Le premier système proposé par les auteurs de SQuAD dans [Rajpurkar et al., 2016] est une régression logistique basée sur un ensemble de traits (*features*). Ce modèle, décrit à la section 3.4.2, identifie correctement 40.0% des spans-réponses sur l'ensemble "dev". Comme dit en fin de section 1.2, il y a eu ensuite une longue série de systèmes basés sur des réseaux neuronaux. Les scores d'exact match obtenus par ces modèles varient entre 53 et 83. Depuis fin 2018, l'usage de BERT [Devlin et al., 2019] a permis d'améliorer grandement les performances. Les premières performances de BERT sont de 85.1 (exact match). Même si de plus en plus de récents modèles sont compétitif vis à vis de BERT, il reste une référence, car en plus d'être utilisé sur des problèmes de Question-Réponse, il peut être entraîné sur de nombreux autres problèmes de TALN. C'est pourquoi on décrit ici BERT [Devlin et al., 2019] brièvement dans la section 3.4.1. On présente en table 3.3 les résultats des meilleurs systèmes réalisés sur SQuAD.

3.4.1. Système BERT

Comparativement aux performances état de l'art en début mémoire, il y a une nette progression des systèmes de Question-Réponse notamment grâce à BERT. Sorti au cours de nôtre mémoire, le modèle *Bidirectional Encoder Representations from Transformers* ou communément appelé BERT est un système neuronal (voir section 2.11) crée par GOOGLE AI fin 2018. Ce système est aujourd'hui très populaire dans la communauté TALN et notamment sur SQuAD. En particulier, à sa sortie, BERT a permit d'augmenter de plus de 5% les performances état de l'art en terme d'exact match (métrique expliquée à la section 3.2) sur SQuAD. Contrairement aux modèles directionnels, qui lisent l'entrée de texte séquentiellement (de gauche à droite ou de droite à gauche) BERT lit en une fois l'ensemble des mots. Cette caractéristique permet au modèle d'apprendre le contexte d'un mot en fonction de

Tab. 3.3. État de l'art sur SQuAD (mesures réalisées sur la tranche de données test)

rang	**model**	**ex. match**	**F1**
	performance Humaine *Stanford University* [Rajpurkar et al., 2016]	82.3	91.2
1 (21 mai 2019)	XLNet (single model*****) Google Brain & CMU	89.9	95.1
2 (11 août 2019)	XLNET-123 (single model*****) MST/EOI	89.6	94.9
3 (21 juillet 2019)	SpanBERT (single model*****) FAIR & UW	88.8	94.6
4 (03 juillet 2019)	BERT+WWM+MT (single model*****) Xiaoi Research	88.7	94.4
5 (21 juillet 2019)	Tuned BERT-1seq Large Cased (single model*****) FAIR & UW	87.5	93.3
10 (05 octobre 2018)	BERT (single model*****) Google AI Language	85.1	91.8
	Régression logistique [Rajpurkar et al., 2016]	40.4	51.0

*****système basé sur un unique modèle

tout son environnement (à gauche et à droite du mot). La force de BERT vient du fait qu'on l'oblige à capturer les liens entre les mots. Ainsi à l'aide de larges corpus d'entraînements (BooksCorpus (800 millions de mots) [Zhu et al., 2015] et Wikipédia Anglais (2,500 million de mots)), le système est préentraîné en deux étapes:

- On masque une partie des mots afin que le système apprenne à correctement les prédire.

- On entraîne le système à prédire pour une paire de phrases donnée, si la seconde phrase est la suite logique de la première dans le document d'origine.

Ce préentraînement permet à BERT d'être applicable à une grande variété de tâches linguistiques. Après le préentraînement, on utilise les paramètres obtenus pour adapter le plus possible le modèle à la tâche souhaitée (*fine-tuning*). Dans le cas du Question-Réponse, en particulier de SQuAD v.1.1, on introduit le vecteur "debut" S et le vecteur "fin" E (S,$E \in \mathbb{R}^H$). L'intuition derrière ces vecteurs et de capturer des caractéristiques exclusivement sur les mots de début et de fin des spans-réponses. Ils vont donc être utilisés par le modèle pour identifier la position de début et de fin du span-réponse. La phase d'entraînement vise donc à apprendre S et E. Ainsi, on donne au système l'ensemble des mots (vectorisés) de la question et du paragraphe. Le système préentraîné permet de représenter le $i^{\text{ème}}$ token du paragraphe(de taille n) en un vecteur T_i de dimension H ($0 < i < n$). La probabilité P_i que le $i^{\text{ème}}$ *token* du paragraphe corresponde au début du span-réponse est calculé à l'aide du softmax du produit scalaire entre T_i et S:

$$P_i = \frac{e^{S \cdot T_i}}{\sum_{j=1}^{n} e^{S \cdot T_j}}$$

Une formule analogue avec E est utilisée pour calculer la probabilité que le $i^{\text{ème}}$ *token* du paragraphe corresponde à la fin du span-réponse. On attribue un score à chaque span candidat, en particulier le score attribué au span allant du $i^{\text{ème}}$ au $j^{\text{ème}}$ token du paragraphe correspond à $S \cdot T_i + E \cdot T_j$ (avec $i < j$). Finalement, le système sélectionne le span ayant le meilleur score.

3.4.2. Système basé sur des *features*

Dans leur article [Rajpurkar et al., 2016] présentent la seule approche basée sur des *features* testée sur SQuAD. Pour leur régression logistique, les auteurs ont extrait divers types de *features* pour chacun des spans-candidats (les spans possibles contenus dans chacune des phrases du paragraphe). Bien que lacunaire, on utilise l'information fournie dans [Rajpurkar et al., 2016] afin d'expliquer plus en détail ces traits. On illustrera nos explications à l'aide de l'exemple présenté en figure 3.4. Les arbres présentés aux figures 3.5, et 3.6 permettent une meilleure compréhension de certains *features*.

Fig. 3.4. Exemple de question-réponse (sur SQuAD v.1.1) utilisée par les auteurs pour présenter leurs *features*. On dénotera par Q la question , par P la phrase contenant le span-candidat, et par s le span-candidat "gravity" choisi en exemple. Les mots communs entre Q et P sont mis en italique.

Paragraphe: `In meteorology, precipitation is any product of the condensation of atmospheric water vapor that falls under gravity. The main forms of precipitation include drizzle, rain, sleet, snow, graupel and hail... Precipitation forms as smaller droplets coalesce via collision with other rain drops or ice crystals within a cloud. Short, intense periods of rain in scattered locations are called ‘‘showers’’.`

Question ("Q"): `What causes precipitation to fall?`
Réponse et span-candidat ("s"): gravity
Phrase contenant le span-candidat ("P"): `In meteorology, precipitation is any product of the condensation of atmospheric water vapor that falls under gravity.`

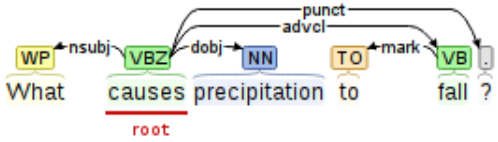


Fig. 3.5. Analyse en dépendance de "Q", produite par CoreNLP. On souligne en rouge le mot placé à la racine de l'arbre en dépendance obtenu.

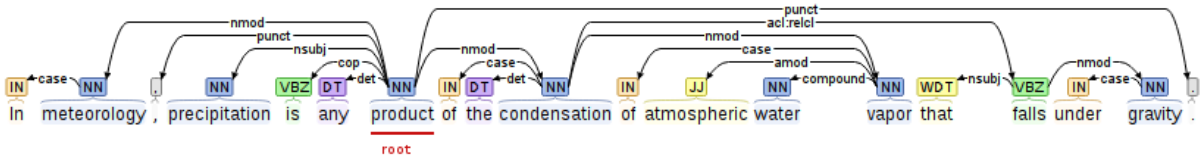


Fig. 3.6. Analyse en dépendance de "P", produite par CoreNLP. On souligne en rouge le mot placé à la racine de l'arbre en dépendance obtenu.

Avant de débuter l'explication, il faut noter que les auteurs de SQuAD ont discrétisé en dix catégories chaque *feature* continue. Voici les détails des familles de traits utilisés par la régression logistique de [Rajpurkar et al., 2016]:

(1) **Correspondance de fréquence de mots:** Famille composée de quatre sommes de TF-IDF[4] (voir section 2.3) des mots apparaissant à la fois dans la question et dans:

- le span-candidat.

  E.g., la somme "sum" des TF-IDF des mots communs à Q et s est de 0 puisqu'ils n'en ont pas. Après discrétisation cette somme est classée dans la catégorie: $[0 \leq sum < 0.01]$.

- la partie de la phrase à gauche du span-candidat

  E.g., Q et la partie de P à gauche de s ont deux mots communs: "precipitation" et "fall". Les auteurs rapportent que la somme des TF-IDF obtenue appartient à: $[7.9, 10.7]$.

- la partie de la phrase à droite du span-candidat.

  E.g., n'ayant pas de mots à droite de s la somme des TF-IDF obtenue est nulle.

- la totalité de la phrase du span-candidat.

  E.g., la somme obtenue est la même que celle calculée avec la partie de P à gauche de s.

(2) **Correspondance de fréquence de bigramme (2-grams):** Similaire à la famille de *features* précédente sauf qu'on utilise des bigrammes. Généralisation de la méthode TF-IDF de [Shirakawa et al., 2015].

E.g., N'ayant aucun bigramme en commun à Q et s, la somme des TF-IDF obtenue est nulle et appartient à: $[0, 2.4]$. Aussi il n'y a aucun bigramme en commun à Q et P, les auteurs rapportent que la somme de TF-IDF obtenue appartient à: $[0, 2.7]$.

(3) **Correspondance des racines:** Après avoir obtenu "AQ" l'arbre en dépendance de la question et "AP" celui de la phrase du span-candidat, on indique, à l'aide de booléens si:

- le mot placé à la racine de "AQ" correspond au mot placé à la racine de "AP".
- le mot placé à la racine de "AQ" est inclus dans la phrase span-candidat.
- le mot placé à la racine de "AP" est inclus dans la question.

[4][Rajpurkar et al., 2016] ne précisent pas les détails de calcul de leurs TF-IDF.

E.g., Après analyse en dépendance de Q et P, on observe que "*causes*" et "*product*" sont les mots placés respectivement à la racine des deux arbres en dépendance. Comme on peut le voir, aucun des trois cas étudiés n'est vrai, les trois booléens sont donc tous faux.

(4) **Taille:** On calcule, en mots, la taille du span-candidat ainsi que:

- la partie de la phrase à gauche du span-candidat,
- la partie de la phrase à droite du span-candidat,
- la totalité de la phrase du span-candidat.

E.g., La taille "*num*" de s étant de 1, celle-ci appartient donc à la catégorie: $[1 \leq num < 2]$. Concernant la taille de la partie de la phrase à gauche de s, celle-ci appartient à la catégorie: $[15 \leq num < 19]$.

(5) **Fréquence des mots du span:** Somme des TF-IDF des mots du span-candidat, indépendamment du fait qu'ils apparaissent dans la question.

E.g., Les auteurs rapportent qu'après discrétisation, la somme des TF-IDF des mots de s appartient à: $[5.2, 6.9]$.

(6) **Étiquettes des constituants:** Étiquettes des mots du span-candidat obtenues à l'aide de l'analyse en constituant, optionnellement combinées avec avec le type de la question.

E.g., s contient uniquement le syntagme nominal "*gravity*" dont l'étiquette est NP. Il est possible de combiner cette étiquette avec le type de Q: "what".

(7) **Étiquettes POS span:** Séquence des étiquettes grammaticales des mots du span optionnellement combinée avec le type de la question.

E.g., étant donné que s ne contient qu'un seul nom: "*gravity*", le premier trait correspondant à la séquence d'étiquettes grammaticales des mots de s est [NN]. Le second trait obtenu en combinant la séquence d'étiquettes grammaticales des mots de s avec le type de Q est [NN]-what.

(8) ***Features* lexicalisées:** Lemmes des mots de la question combinés avec les lemmes des mots de la phrase se trouvant dans une fenêtre de deux mots autour du span-candidat dans l'arbre de dépendance. Aussi, on conserve les relations de l'arbre de dépendance. De manière séparée, les lemmes des mots de la question sont associés aux lemmes des mots du span.

E.g., les auteurs rapportent qu'en combinant les lemmes de Q avec les lemmes des mots de P se trouvant dans une fenêtre de deux mots autour de s, ils ont extrait le trait suivant: Q:"cause", P:"under"$\xleftarrow{cause}$. En combinant les lemmes des mots de Q et de s ils obtiennent: Q:"fall", s:"gravity".

(9) **Analyse en dépendance:** Pour chaque lemme t apparaissant à la fois dans la question et dans la phrase du span-candidat, on extrait dans l'arbre en dépendance de la phrase, le chemin allant de t jusqu'au span-candidat. Chacun des chemins trouvés précédemment peut être optionnellement combiné avec le chemin allant du pronom interrogatif à t dans l'arbre en dépendance de la question. On conserve dans les chemins les étiquettes grammaticales.

E.g., "*fall*" est le seul lemme partagé par Q et P ainsi, comme on peut le voir à l'aide de la figure 3.6, on extrait dans l'arbre de dépendance de P: VBZ $\xrightarrow{nmod}$ NN. En le combinant avec le chemin dans l'arbre en dépendance de "Q" (voir 3.5) on obtient : what $\xleftarrow{nsubj}$ VBZ$\xrightarrow{advcl}$ + VBZ $\xrightarrow{nmod}$NN.

Au total, leur modèle est construit sur plus de 180 millions de *features*, la plupart d'entre eux sont lexicalisés ou issus de l'analyse en dépendance. Le modèle résultant peut dans 40.4% des cas déterminer le bon span-réponse.

L'une des limites majeures de ce modèle est qu'il repose en grande partie sur des *features* lexicalisés, ce qui implique donc de devoir faire du "cas par cas". Afin de mesurer l'importance de chaque famille de *features*, on présente en table 3.4 les performances rapportées par les auteurs de SQuAD de leur régression logistique lorsqu'on élimine tour à tour chacune des familles de *features*.

Comme on peut le voir au travers des résultats présentés en table 3.4, les *features* les plus importantes sont les *features* lexicalisées et les *features* d'analyse en dépendances. En comparant les performances obtenues sur le train et sur le dev, on note que ce modèle surapprend les données d'entraînements et généralise donc mal. En effet, mis à part lorsque les *features* lexicalisés et les *features* d'analyse en dépendances sont retirées, les scores F1 obtenus sont d'environ 91% sur train alors qu'ils sont d'environ 50% pour ces mêmes familles sur dev. Cela nous indique donc que le modèle apprend très bien de l'ensemble d'entraînement mais que lorsqu'il observe de nouveau cas issus de dev, le modèle a de la difficulté à réaliser des prédictions. Pour respecter notre volonté de construire un modèle à taille contrôlé, on

Tab. 3.4. Performances du modèle de regression logistique de [Rajpurkar et al., 2016] en réalisant de l'ablation de *features* (source: [Rajpurkar et al., 2016]).

	F1	
Familles de *features* sélectionnées pour la régression logistique	train	dev
Toutes	91.7%	51.0%
Toutes - Lexalisés et Analyse en dépendance	33.9%	35.8%
Toutes - Lexalisés	53.5%	45.4%
Toutes - Annalyse en Dépendance	91.4%	46.4%
Toutes - Correspondance des fréquences de mots	91.7%	48.1%
Toutes - Étiquettes POS span	91.7%	49.1%
Toutes - Correspondance fréq. bigrams	91.7%	50.3%
Toutes - Étiquettes des constituants	91.7%	50.4%
Toutes - Tailles	91.8%	50.5%
Toutes - Fréq. mots du span	91.7%	50.5%
Toutes - Correspondance des racines	91.7%	50.6%

développe dans ce mémoire des modèles basés sur des features non lexicalisées. Bien que dans l'absolu on souhaite battre les performances de [Rajpurkar et al., 2016] (51% en score F1 sur dev), notre véritable point de comparaison est 45.4% en terme de score F1 sur dev, puisque ces résultats correspondent aux performances de la régression logistique de [Rajpurkar et al., 2016] sans *features* lexicalisées.

Chapter 4

Détection de la phrase-réponse

Après avoir réalisé les expériences préliminaires nécessaires à la meilleure connaissance de SQuAD, nous avons cherché à construire des systèmes nous permettant de détecter la phrase-réponse, c'est à dire la phrase dans laquelle se trouve le span-réponse. Étant donné que chacun des spans se trouve dans une phrase, il semble en effet intéressant d'identifier d'abord la phrase-réponse avant d'identifier le span-réponse lui-même.

Fig. 4.1. Exemple d'identification attendue d'une phrase-réponse (en jaune).

paragraphe: According to the same statistics, the average age of people living in Newcastle is 37.8 (the national average being 38.6). Many people in the city have Scottish or Irish ancestors. There is a strong presence of Border Reiver surnames, such as Armstrong, Charlton, Elliot, Johnstone, Kerr, Hall, Nixon, Little and Robson. There are also small but significant Chinese, Jewish and Eastern European (Polish, Czech Roma) populations. There are also estimated to be between 500 and 2,000 Bolivians in Newcastle, forming up to 1% of the population-the largest such percentage of any UK city.

question: What is the average age of people who live in Newcastle?
réponse: 37.8

Il y a plusieurs raisons à vouloir identifier la phrase-réponse ou la phrase d'intérêt et particulièrement en recherche d'information. En effet, cela permet notamment à un lecteur d'obtenir la réponse à sa question ou requête tout en conservant le contexte de celle-ci.

Aussi, en sélectionnant préalablement la phrase-réponse au span-réponse lui-même, on réduit l'espace de recherche pouvant amener à un gain de temps, voire en performance. Enfin, il semble potentiellement plus simple à un modèle de sélectionner un span parmi un ensemble restreint de spans plutôt qu'avec la totalité des spans du paragraphe.
Comme décrit en section 3.4, dans leur article [Rajpurkar et al., 2016], les auteurs de SQuAD présentent pour identifier les spans-réponses, une régression logistique donnant des performances en *exact match* de 40.4% et un score F1 de 51.0% sur le corpus test. Ils notent a posteriori que dans 79.3% des cas, le span identifié par leur système se trouve dans la bonne phrase-réponse. Les modèles neuronaux bidirectionnels de [Wang et al., 2018] et [Shen et al., 2018] développés pour l'identification des phrases-réponses donnent des performances respectives de 79.8% et de 68.38% sur SQuAD. Nous utiliserons ces performances comme base de comparaison.

Dans le cas de la régression logistique de [Rajpurkar et al., 2016], la comparaison des 79.3% obtenus a posteriori de l'identification des spans-réponses avec les 40.4% obtenus pour l'identification des spans-réponses, semble nous indiquer que trouver la phrase-réponse est plus facile. Les performances à 79.8% pour la tâche d'identification des phrases-réponses obtenues par [Wang et al., 2018] semblent également confirmer cette observation. Cependant les 68.38% de phrases-réponses correctement identifiées par [Shen et al., 2018] nuance cela et montre une certaine difficulté à identifier les phrases-réponses. Après analyse du contenu des paragraphes de SQuAD, la difficulté de la tâche d'identification des phrases-réponses peut être expliquée par le fait qu'étant donné que l'ensemble des phrases d'un paragraphe partagent un même sujet, leur sens global est proche. Ainsi, il est plus difficile d'étudier les phrases dans leur globalité dans le but d'en distinguer une, que d'étudier et distinguer des spans avec moins de contenu. Notons qu'il est difficile de comparer les tâches d'identification des phrases-réponses et des spans-réponses puisque que mis à part [Rajpurkar et al., 2016], aucune autre étude, même a posteriori de l'identification des spans-réponses, ne s'est intéressée à mesurer la performance des systèmes pour l'identification des phrases-réponses.
Il serait naturel de supposer, qu'une fois le bon span-réponse trouvé, il suffit de retourner la phrase qui le contient afin d'obtenir la phrase-réponse. Ainsi basé sur cette supposition, on pourrait borner inférieurement les performances de sélection des phrases-réponses après

identification des spans-réponses, par les performances de sélection des spans-réponses. Cependant il arrive fréquemment qu'un span-réponse apparaissent dans d'autres phrases que celle permettant de répondre à la question. Les métriques d'évaluations de SQuAD ne vérifient pas si un span retourné est bien celui identifié, elles s'intéressent juste à la forme des spans. Ainsi il n'est pas garanti que les performances qu'obtient un système à sélectionner les phrases-réponse a posteriori des spans-réponses soient toujours supérieur à celles qu'il obtient sur la sélection des spans-réponses.

Dans ce chapitre, on présente, dans un premier temps, la méthodologie nécessaire et utilisée dans la segmentation des paragraphes en phrases avant de décrire les quatre grandes familles de modèles étudiées et d'analyser les résultats obtenus.

4.1. Segmentation des paragraphes en phrases

Après s'être familiarisé avec le formatage du texte, on utilise les librairies NLTK et SpaCy afin de comparer leurs performances dans la segmentation des phrases sur SQuAD. Ne pouvant segmenter manuellement chacun des paragraphes en phrases afin d'obtenir un élément de référence pour la segmentation attendue, les performances sont mesurées en calculant le nombre de span-réponse chevauchant deux phrases identifiées. Ainsi un paragraphe est noté comme mal segmenté si l'un des spans-réponse qu'il contient chevauche deux phrases. Étant donné qu'un span-réponse est obligatoirement inclus dans une phrase, si celui-ci chevauche plusieurs phrases, il sera impossible de pouvoir le retrouver par la suite.

Pour la segmentation en phrases NLTK utilise des expressions régulières alors que SpaCy fait usage d'un analyseur en dépendance neuronal. Pour les deux librairies, on observe de très bons résultats puisque leurs taux d'erreur (sur le train) est de 0.25% pour NLTK et 0.16% pour SpaCy. Dans le but d'obtenir les meilleurs résultats possible, on utilise le découpage de SpaCy.

On observe que la source majeure d'erreur de mauvaise segmentation est la ponctuation. En effet, on utilise souvent des points dans les abréviations de noms ou de titres, ceci peut dérouter les systèmes lors de la segmentation des paragraphes. On illustre en figure 4.2 un exemple de mauvaise segmentation de phrase.

Fig. 4.2. Exemple de mauvaise segmentation de phrases par SpaCy avec les spans-réponses en couleur.

---

phrase originale : `On May 19 thousands of students massed downtown protesting the Klavern, and only the arrival of college president Fr. Matthew Walsh prevented any further clashes.`
segmentation réalisée: `["On May 19 thousands of students massed downtown protesting the Klavern, and only the arrival of college president Fr.", "Matthew Walsh prevented any further clashes."]`

phrase originale : `At a subsequent eolomelodicon concert on 10 June 1825, Chopin performed his Rondo Op. 1.`
segmentation réalisée: `["At a subsequent eolomelodicon concert on 10 June 1825, Chopin performed his Rondo Op.", "1."]`

4.2. Compte des tokens partagés

Le premier système réalisé "`mutuals_words`" est un système naïf. Pour que le système sélectionne la phrase-réponse, on attribue à chaque phrase un score correspondant au ratio du nombre de tokens partagés avec la question sur le nombre total de tokens dans la phrase. La phrase ayant le plus haut score est sélectionnée. Avant de comptabiliser le nombre de tokens communs entre la question et chaque phrase, il est préférable de normaliser l'écriture des mots et d'en réduire la diversité en les lemmatisant ou en les stemant (à l'aide du PorterStemmer de NLTK). En pratique la lemmatisation permet la plus grande amélioration des résultats. En conséquence, les résultats présentés pour "`mutuals_words`" sont obtenus en utilisant la lemmatisation.

Aussi, à l'aide d'une liste de 179 mots vides (*stop words*) proposés par NLTK, nous avons éliminé cet ensemble de mots avant de réaliser le compte des tokens communs entre les phrases et la question. On rapporte une légère baisse des performances après cette élimination. On explique cela par le fait que de nombreuses questions sont des reformulations partielles ou totales des phrases-réponses, ainsi, le fait d'enlever du compte ces mots vides pauvres en intérêt sémantique affecte les performances de notre système.

Notons que ce système présente une limite majeure. En effet, étant donné qu'il ne s'intéresse qu'à la forme des mots, deux mots ayant la même écriture de sens différents sont comptabilisés comme identiques. À contrario, deux mots de même sens se distinguant dans leur forme sont eux différenciés. Des exemples de problèmes sont illustrés en figure 4.3.

Fig. 4.3. Exemples de scores attribués avec "`mutuals_words`", on trie les phrases par ordres décroissant en fonction de leur score. (les spans-réponses sont mis en couleur).

---

phrase 1 : `Super Bowl 50 was an American football game to determine the champion of the National Football League (NFL) for the 2015 season.`

phrase 2 : `The American Football Conference (AFC) champion Denver Broncos defeated the National Football Conference (NFC) champion Carolina Panthers 24-10 to earn their third Super Bowl title.`

phrase 3 : `The game was played on February 7, 2016, at Levi's Stadium in the San Francisco Bay Area at Santa Clara, California.`

phrase 4 : `As this was the 50th Super Bowl, the league emphasized the "golden anniversary" with various gold-themed initiatives, as well as temporarily suspending the tradition of naming each Super Bowl game with Roman numerals (under which the game would have been known as "Super Bowl L"), so that the logo could prominently feature the Arabic numerals 50.`

Question 1 : `Which NFL team won Super Bowl 50? Denver Broncos`

score phrase 1 : 4 / 22 (4 tokens communs: NFL, Super, Bowl, 50)

score phrase 4 : 4 / 57 (4 tokens communs: Which, Super, Bowl, 50)

score phrase 2 : 2 / 25 (2 tokens communs: Super, Bowl)

score phrase 3 : 0 / 21 (0 tokens communs)

Question 2 : `What venue did Super Bowl 50 take place in? Levi's Stadium in the San Francisco Bay Area at Santa Clara`

score phrase 1 : 3 / 22 (3 tokens communs: Super, Bowl, 50)

score phrase 3 : 1 / 21 (1 token commun: in)

score phrase 2 : 2 / 25 (2 tokens communs: Super, Bowl)

score phrase 4 : 3 / 57 (3 tokens communs: Super, Bowl, 50)

En particulier avec l'exemple de la question 1, on voit que le système sélectionne la première phrase alors qu'avec une meilleure compréhension sémantique au niveau des mots il aurait pu sélectionner la deuxième phrase. En remarquant les liens sémantiques suivants entre *"won"* et *"deafeated"*, *"NFL"* et *"Football"*, *"won"* et *"title"*, *"won"* et *"champion"* il aurait sans doute pu sélectionner la bonne phrase. Aussi, dans l'exemple 2, en comprenant que *"take place in"* faisait référence à un lieu il aurait été aisé au système de retourner la troisième phrase.

4.3. *Embeddings* des phrases et des questions

Afin de résoudre les limites de `"mutuals_words"` on construit une série de systèmes utilisant les *embeddings* des mots dans le but d'obtenir une représentation sémantique de chaque phrase/question pondérée sur le nombre de mots quelles contiennent. On nomme chacun des systèmes de cette série par `"emb_sent_sim"` où sim, la similarité utilisée appartient à l'ensemble [euc, cos, dice, jacc] correspondant aux similarités euclidienne, cosinus, dice et jaccard décrites à la section 2.4 .

Dans un premier temps on définit l'*embedding* $e(S)$ d'une phrase $S \equiv w_1 \cdots w_l$ en faisant la moyenne de l'*embedding* de ses mots $e(w_i)$ ($i \in [1,\, l]$) (voir section 2.2.3):

$$e(S) = \frac{1}{l} \sum_{i=1}^{l} e(w_i) \tag{4.3.1}$$

De manière analogue, on calcule les *embedding*s de la question Q: $e(Q)$ ainsi que ceux de chacune des m phrases du paragraphe: $e(S_i)$ (avec S_i la $i^{\text{ème}}$ phrase du paragraphe). Puis, à l'aide de la mesure de similarité choisie: *sim*, on détermine "I" l'index de la phrase sélectionnée ($I \in [0,m-1]$) en prenant la phrase ayant le plus haut score de similarité avec la question:

$$I = \underset{0 \leq i < m}{\operatorname{argmax}}(sim(e(S_i), e(Q))) \tag{4.3.2}$$

Lors de nos expériences, on compare deux systèmes d'*embeddings* populaires: GloVe[1] [Pennington et al., 2014] et fastText[2] [Bojanowski et al., 2017].

Entraîné sur Wikipédia, GloVe est un système qui associe à chaque mot du corpus d'entraînement un *embedding*. L'inconvénient de GloVe est qu'il ne génère pas de vecteur pour des

[1] https://nlp.stanford.edu/projects/glove/

[2] https://fasttext.cc/

mots non vus à l'entraînement.

Avec Wikipédia et Common Crawl[3] comme corpus d'entrainement, fastText peut contrairement à GloVe retourner un vecteur pour n'importe quel mot. En effet, grâce à la prise en compte des ngrams de caractères au sein des mots de la construction des *word-embeddings*, fastText est capable de retourner un vecteur pour tout mot même si celui-ci n'a pas été observé lors de la phase d'entraînement.

En pratique, les *embeddings* préentraînés de fastText donnent de meilleurs performances comparativement aux *embeddings* préentraînés de GloVe. En plus, de par le fait que fastText est entraîné sur un plus gros corpus on explique cette différence de performances du fait qu'environ 15% des mots de SQuAD sont inconnus de GloVe, notre système doit donc les ignorer durant le calcul des *embeddings* des phrases et questions.

Notons que nous avons essayé à l'aide de fastText et des paragraphes du train de SQuAD comme corpus d'entraînement, de générer nos propres *embeddings*, cependant les performances étaient diminuées d'environ 5% comparativement aux *embeddings* préentraînés sur Wikipédia et Common Crawl. Cette baisse de performances provient sans doute du fait que le corpus d'entraînement utilisé n'est pas suffisamment grand.

Ainsi les résultats présentés à la section 4.7 pour le modèles `"emb_sent_sim"` sont obtenus avec les *embeddings* préentrainés de fastText.

4.4. Utilisation d'*embeddings* de n-grams

Dans le but d'améliorer nos modèles et de les rendre plus précis, plutôt que de toujours étudier chacune des phrases dans sa globalité, on fragmente celle-ci en un ensemble de n-grams de mots. L'idée avec ce partitionnement est d'obtenir des segments de phrase plus spécifiques sémantiquement et d'ainsi sélectionner la phrase possédant le n-gram le plus similaire à la question. Dérivés de la section 4.3 chacun des modèles de cette famille de modèles est dénommé par "k-`gram_sim`" où k, la taille des n-grams considérés, est un entier et *sim* est la similarité appartenant à l'ensemble [cos, dice] correspondant aux similarités cosinus et dice. Notons qu'on nomme par "[a-b]-`gram_sim`" les modèles considérant les k-grams pour tout $k \in [a, b]$.

À l'aide de fastText, similairement à $e(S)$ (voir équation 4.3.1), $e(s)$ l'*embedding* d'un n-gram

[3] https://commoncrawl.org/

s est obtenu en moyennant les *embeddings* de ses mots w_i $(0 \leq i < k)$. Dès lors, pour une similarité choisit sim, on peut calculer $sim(e(s),\ e(Q))$: la similarité entre le n-gram s et la question Q. Le score d'une phrase S contenant v n-grams s_j $(0 \leq j < v)$ est alors définit par:

$$score(S) = \max_{0 \leq j < v} sim(e(s_j), e(Q)) \tag{4.4.1}$$

Ainsi, après avoir attribué à chacune des m phrases du paragraphe un score en fonction de son ensemble de n-grams (obtenu avec NLTK), on retourne de manière analogue à `"emb_sent_sim"` la phrase ayant le plus haut score (voir équation 4.3.2).

4.5. Utilisation des *embeddings* des n-grams et des phrases

On réalise une famille de système correspondant à la combinaison de systèmes `emb_sent_sim` et `k-gram_sim` (voir les sections 4.3 , 4.4). Ainsi, le score attribué à chaque phrase correspond au maximum des scores retournés par les modèles `emb_sent_sim` et `k-gram_sim`. La phrase ayant le meilleur score est sélectionnée. On nomme chacun de ces systèmes par `"k-gram_sent_sim"` où est k un entier ou un ensemble d'entiers, et sim $\in$ [cos, dice] la mesure de similarité utilisée.

4.6. Comparaison avec les n-grams de la question

De manière analogue à ce qui est décrit dans la section 4.4, en plus d'évaluer la similarité avec la totalité de la question Q, on va également segmenter celle-ci en n-grams afin de calculer un score de similarité avec chacun d'entre eux. Ainsi, pour chaque question Q et pour un n fixé, on obtient un ensemble de n-grams q_i $(0 < i \leq u)$ avec leur *embedding* respectif: $e(q_i)$. En plus des similarités calculées avec $e(Q)$ (voir équation 4.3.2 et 4.4.1) on calcule maintenant aussi la similarité avec chacun des $e(q_i)$. Enfin, on prend le maximum de l'ensemble des scores de similarité obtenus.

En pratique cela ne marche pas si l'on considère tous les n-grams possibles de chaque question. Après étude, on observe que les résultats sont faussés dès lors que la question et la phrase partagent un même n-gram même si celui-ci n'a pas d'intérêt sémantique (*on*, *was*, ...). Ce phénomène est particulièrement visible dans le cas des modèles de la section 4.4.

Après expériences, on constate qu'en utilisant une segmentation en 3-grams et plus des questions permet d'améliorer légèrement les performances comparativement aux modèles

Fig. 4.4. Illustration des limites du système n-grams sur les questions.

phrase 1 : `John Paul II's visits to his native country in 1979 and 1983 brought support to the budding solidarity movement and encouraged the growing anti-communist fervor there.`

phrase 2 : `In 1979, less than a year after becoming pope, John Paul celebrated Mass in Victory Square in Warsaw and ended his sermon with a call to "renew the face" of Poland: Let Thy Spirit descend!`

phrase 3 : `Let Thy Spirit descend and renew the face of the land!`

phrase 4 : `This land!`

phrase 5 : `These words were very meaningful for the Polish citizens who understood them as the incentive for the democratic changes.`

Dès lors que le système s'intéresse aux similarités des tokens de la phrase et de la question (1-gram) on obtient les résultats de similarités suivants:

Question: `What pope as a native of Poland? John Paul II`

score de la phrase 1 : 1 (1 token commun: native)

score de la phrase 2 : 1 (3 tokens commun: pope, a, of)

score de la phrase 3 : 1 (1 token commun: of)

score de la phrase 5 : 1 (1 token commun: as)

score de la phrase 4 : 0

classiques qui utilisent toujours l'entièreté de la question dans le calcul des similarités. Aussi, l'élimination des pronoms interrogatifs des questions, ainsi que la normalisation d'écriture (décrite en 3.3) des phrases et des questions avant le calcul des similarités permet de gagner quelques pour cent au niveau des performances. En effet, en éliminant les tokens sémantiquement faibles on améliore nos systèmes.

Ainsi, à la section 4.7 les résultats présentés pour les modèles `emb_sent_sim`, k-`gram_sim` et `k-gram_sent_sim` des sections 4.3, 4.4 et 4.5 sont obtenus en utilisant un système qui normalise les phrases/questions et qui utilise pour le calcul des similarités les 3-grams et plus des questions ôtées de leur pronom interrogatif.

4.7. Résultats

Dans cette section, on présente les performances des différents modèles réalisés sur la tâche de détection des phrases réponses.
On présente en table 4.1 les résultats des systèmes qui étudient les phrases dans leurs intégralités afin de sélectionner celles ayant le meilleur score (présentés dans les sections 4.2 et 4.3).

Tab. 4.1. Pourcentage de phrases-réponses correctement détectées sur dev.

modèle	
`mutuals_words`	63.31%
`emb_sent_euc`	66.04%
`emb_sent_jacc`	66.67%
`emb_sent_dice`	67.10%
[Shen et al., 2018]	68.38%
`emb_sent_cos`	68.50%
[Rajpurkar et al., 2016]*	79.3%
[Wang et al., 2018]	79.8%

*résultats obtenus a posteriori de la présélection des spans-réponses

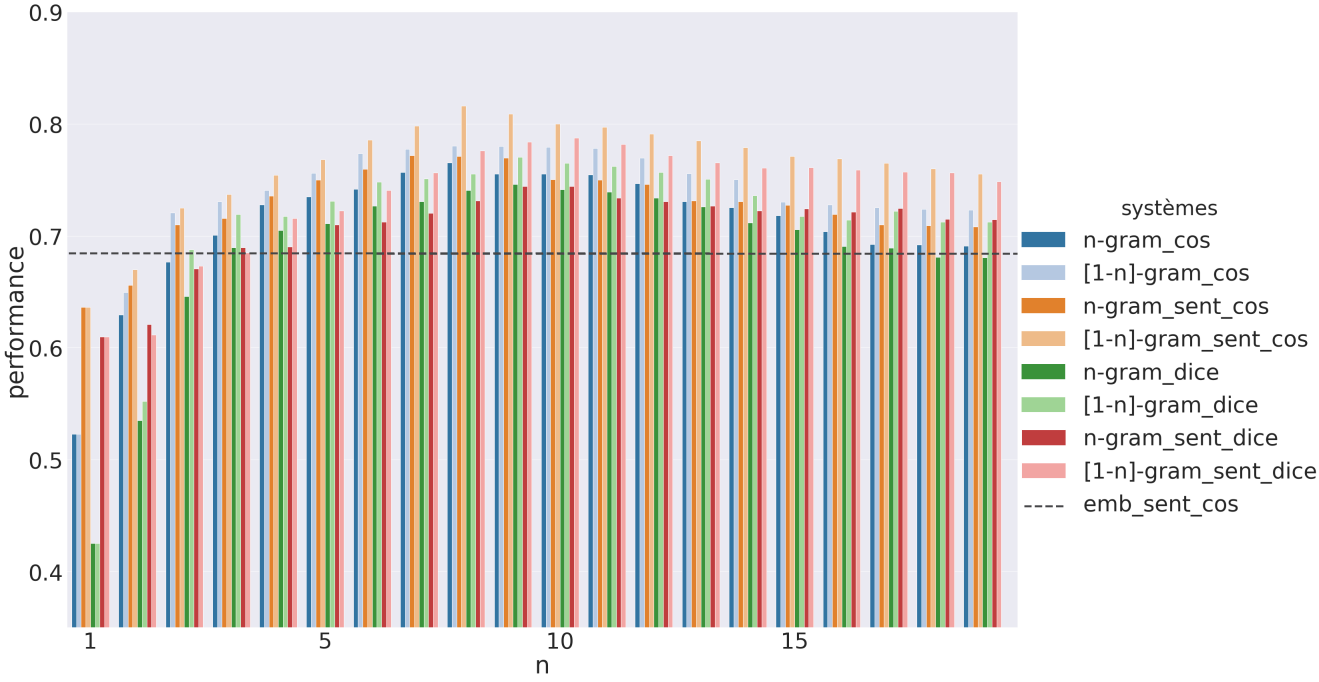
Au vu des résultats de la table 4.1, les systèmes utilisant les *embeddings* de phrases sont meilleurs que `mutuals_words`. Cela confirme que les modèles utilisant les *embeddings* capturent plus d'information sémantique sur les mots. Communément utilisée, la similarité cosinus permet à `emb_sent_euc` d'obtenir les meilleurs résultats (68.50%). Remarquons cependant que l'ensemble des systèmes encodant les phrases dans leur intégralité capturent nettement moins d'informations que la régression logistique de [Rajpurkar et al., 2016] qui, avec ses nombreux *features*, sélectionne un span dans la bonne phrase-réponse dans 79.3% des cas.

L'ensemble des systèmes "`emb_sent_sim`" (avec sim $\in$ [euc, cos, dice, jacc]) et le système neuronal bidirectionnel de [Shen et al., 2018] se ressemblent dans le fait que tous encodent l'intégralité des phrases et des questions afin de les comparés pour déterminé la phrase-réponse. Malgré la simplicité de notre système `emb_sent_euc` nous arrivons à

dépasser légèrement les performances de [Shen et al., 2018] (68.38%).

Étudions et analysons les résultats obtenus par les familles de systèmes qui segmentent les phrases en n-grams avant de les comparés avec chaque 3-grams et plus de la question dans le but de retourner la phrase contenant le n-gram le plus plus proche de la question (voir sections 4.4, 4.5, 4.6). La figure 4.5 rapporte les performances obtenues par l'ensemble de ces systèmes.

Fig. 4.5. Performances des modèles dans la détection des phrases-réponses en utilisant les *embeddings* des n-grams en fonction de n.



On observe qu'avec un n petit les performances des approches basées sur les *embeddings* de n-grams sont inférieures à celles basées sur les *embeddings* de phrases (`emb_sent_cos`). Ceci est du au phénomène décrit à la section 4.6. À l'inverse, lorsque n est trop grand les performances sont proches de celles obtenues en comparant la question à chacune des phrases du paragraphe dans leur totalité (voir table 4.1). Ces résultats montrent que fragmenter les phrases en {1-9}grams, permet d'obtenir une bonne segmentation des éléments comparés à la question (fragmentée) sans trop y inclure de mots non pertinents pour la comparaison. Bien qu'il soit basique, avec 81% de phrases-réponses correctement identifiées, `[1-9]-gram_sent_cos` est le meilleur système obtenu. Ce score est cette fois-ci

supérieur à l'approche de [Rajpurkar et al., 2016] bien que laquelle soit plus simple. On présente dans la table 4.2 le détail des performances obtenues en fonction du type de question.

Tab. 4.2. Performances dans la détection des phrases réponses du système `[1-9]-gram_sent_cos` en fonction du type de questions.

	dev		
Type de question	#	bonne sélection	%
What / Which NN[*]?	3976	3279	82.46%
Who?	1185	922	77.80%
What VB[*]?	1367	1106	80.90%
When / What year?	943	807	85.57%
How much / many?	757	598	78.99%
Where?	477	371	77.77%
How?	468	363	77.56%
What name / is called?	325	257	79.07%
Other	1072	859	80.13%
Total	10570	8562	81.00%

Les résultats du meilleur système (table 4.2) montrent qu'avec 85.57%, les questions de type "When / What year ?" sont les plus faciles dans la détection de phrase-réponse. Après étude, on observe que souvent, la question paraphrase la phrase-réponse. La question et la phrase réponse ont donc en commun une étendue de mots. Ainsi lors des comparaisons entre les spans des phrases et des questions, le système peut plus facilement retourner la bonne phrase puisque les phrases réponses et la question partagent un même span. Illustré à la figure 4.6, ce même phénomène est observable également pour les questions de type "What / Which NN [*]?".

Ainsi, la segmentation des phrases en n-grams permet l'obtention de performances honorables (81.00 %) dans la détection des phrases-réponses. Les résultats du modèle `[1-9]-gram_sent_cos` dépassent les 68.38% du modèle proposé dans [Shen et al., 2018] et les 79.8% obtenu par [Wang et al., 2018]. Il dépasse également les résultats (79.3%) obtenus

Fig. 4.6. Illustration de questions qui paraphrasent les phrases-réponses

Afin qu'elles soient plus visibles, nous avons mis en italique les étendus de mots partagées entre les questions et les phrases réponses.

Question: When did the *1973 oil crisis begin*?
Réponse: October 1973
Phrase réponse : The *1973 oil crisis began* in October 1973 when the members of the Organization of Arab Petroleum Exporting Countries (OAPEC, consisting of the Arab members of OPEC plus Egypt and Syria) proclaimed an oil embargo.

Question: When was the *second oil crisis* ?
Réponse: 1979
Phrase réponse : It was later called the "first oil shock", followed by the 1979 oil crisis, termed the " *second oil crisis*".

Question: What *is the Republic of Kenya named after* ?
Réponse: Mount Kenya
Phrase réponse: *The Republic of Kenya is named after* Mount Kenya.

par [Rajpurkar et al., 2016], qui déterminent la phrase-réponse en identifiant d'abord le span-réponse. Notons que les performance état de l'art pour cette tâche dépassent les 85%. Du fait des resultats de nos systèmes et de l'état de l'art pour la sélection des phrases-réponses, on conclut que la difficulté se situe (un peu étonnamment) au niveau de l'identification du span-réponse.

Chapter 5

Identification du span-réponse

Nous nous concentrons dans ce chapitre sur la tache initiale de détection des spans-réponses. Dans cette partie on décrit dans un premier temps les expériences réalisées afin d'orienter nos systèmes dans la sélection des spans-réponses. On présente ensuite les systèmes finaux ainsi que les résultats obtenus pour la sélection des spans-réponses.

5.1. Détection du span à l'aide des *embeddings*

On s'intéresse en premier lieu, à déterminer s'il est possible d'identifier directement le span-réponse à partir des mots du paragraphe. On conjecture ici que, basé sur la similarité entre les mots de la question et du paragraphe, il est possible d'obtenir un signal discriminant le span-réponse. La supposition ici est que le signal va distinctement croître au niveau du span-réponse. Nous allons donc étudier la similarité entre les questions et le contexte des spans-réponses.

Pour chaque paire (question, span-réponse) du train, soit t la taille (en mot) du span-réponse et i sa position de début (en mot) dans le paragraphe. Après avoir récupéré t et i, on récupère tous les spans de taille t dans la fenêtre de mots allant des positions $i-10$ à $i+t+9$, ces spans sont appelés "spans-contextes". Pour permettre une meilleure compréhension, on présente en figure 5.1 un exemple d'extraction de l'ensemble des spans-contextes.

Similairement à $e(S)$ (voir 4.3.1), on commence par calculer l'*embedding* de la question et de chaque span-contexte à l'aide de fastText. On calcule ensuite la distance cosinus entre chaque span-contexte et sa question correspondante. Finalement, en fonction de la position

Fig. 5.1. Exemple de spans-contextes. (Le span-réponse est coloré en bleu)

Paragraphe: `The Broncos took an early lead in Super Bowl 50 and never trailed. Newton was limited by Denver's defense, which sacked him seven times and forced him into three turnovers, including a fumble which they recovered for a touchdown. Denver linebacker Von Miller was named Super Bowl MVP, recording five solo tackles, 2½ sacks, and two forced fumbles.`

Spans-contextes droits: `{"Miller was", "was named", ···,"tackles 2½"}`

Spans-contextes gauches:" `{"linebacker Von", "Denver linebacker",···,"a fumble"}`

des spans-contextes, on segmente en 21 sous-ensembles l'ensemble des similarités cosinus obtenues. Pour chacun d'eux, on calcule la moyenne de ses similarités cosinus. On présente en figure 5.2 les résultats obtenus.

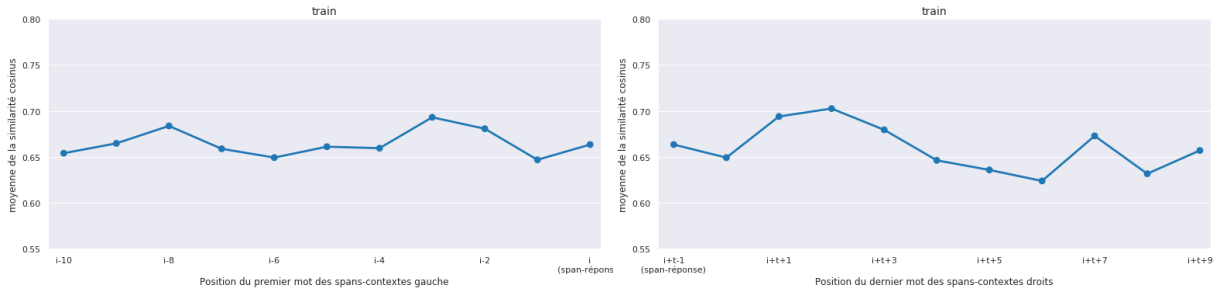


Fig. 5.2. Moyenne des similarités cosinus entre les questions et les spans-contexts (gauche et droite)

Nous souhaitions observer une distinction nette dans la courbe de signal au niveau du span-réponse, cependant cela n'est pas le cas. On constate, qu'il est difficile de distinguer le span-réponse de son contexte en utilisant les *embeddings*. Nous avons utilisé Dice et Jacquard comme mesures de similarité, cependant cela ne donne rien de mieux.

Après observation du contexte des spans-réponse, il apparaît que le contexte du span-réponse contient des mots sémantiquement reliés aux mots du sujet d'intérêt (la question). On présente en figure 5.3 des exemples de spans-réponses.

Fig. 5.3. Exemples de spans-réponses dont les mots voisins sont sémantiquement proches des mots de la question. (Les spans-réponses ont étés mis en bleu et les mots voisins du span-réponses communs à la question ont étés mis en italique. Par soucis de simplicité, nous présentons dans cette exemple uniquement les phrases-réponses, et non l'intégralité des paragraphes.)

Question: How many Pro Bowlers were on the Panthers offense?
Phrase-réponse: The Panthers offense, which led the NFL in scoring (500 points), was loaded with talent, boasting six *Pro* Bowl selections.

Question: When are the new Sky Q products going to be available?
Phrase-réponse: On 18 November 2015, Sky announced Sky Q, a range of *products* and services *to be available* in 2016.

Comme on peut le voir dans les mots autour des spans-réponses sont proches des mots de la questions. Ainsi, lorsque l'on va comparer à la question ces spans-contextes qui contiennent ces mots, on obtient un bon score de similarité ce qui ne permet pas de discriminer le span-réponse de tout ses spans-contextes. Au vu de ces premiers résultats, on voit qu'il n'est pas trivial d'identifier directement le span-réponse, du pas en considérant la question dans sa globalité.

5.2. Identification des spans candidats à l'aide des entités nommées

On s'intéresse à savoir s'il est possible de réduire l'espace de spans-candidats considérés aux seules entités nommées identifiées dans le texte. On mesure pour cela la perte qu'entraîne ce type de présélection. De plus, à l'aide d'a priori et de connaissances linguistiques, on identifie lorsque cela est possible une série d'heuristiques basées sur le type des questions. Intuitivement, certaines questions attendent des types précis de réponses. Par exemple, si une question est de type "Where?" il est probable que la réponse attendue soit un lieu. Néanmoins, d'autres types de questions comme "How?" n'attendent pas de type précis de réponse. Dans ces cas-là, on ne peut établir d'heuristiques de présélection. On présente en table 5.1 la liste de questions pour lesquelles on a identifié des catégories d'entités nommées à présélectionner.

Tab. 5.1. Type d'entitées nommées séléctionées en fonction de certains types de questions

Type de question	**Catégorie d'entités nommées à conserver**
Where?	GPE, LOC, FAC, ORG
How much / many?	MONEY, QUANTITY, PERCENT, CARDINAL, TIME, DATE, ORDINAL
What name / is called?	PERSON, ORG, GPE, LOC, PRODUCT, EVENT, WORK_OF_ART, LAW, LANGUAGE, FAC
Who?	PERSON, ORG, NORP, GPE, PRODUCT
When / What year?	TIME, DATE, EVENT

Les détails de chacune des catégories d'entités nommées, utilisées ici ont été présentés à la section 2.7. On évalue la pertinence de cette présélection en comparant chaque span-réponse à l'ensemble de NER obtenu avec à l'aide de l'exact match et du score F1 (décrites à la section 3.2). On calcule également le nombre moyen de spans-candidats pré-sélectionnés: "size_set". Les résultats pour les types de questions ayant des heuristiques sont présentés en table 5.2.

Tab. 5.2. Résultats d'une présélection réalisée avec les heuristiques décrites en table 5.1

	Train		
Type de question	ex. match	f1	size_set
When / What year?	79.0%	80.4%	3.7
How much / many?	74.6%	77.9%	7.9
Who?	66.2%	70.2%	8.9
What name / is called?	54.2%	59.9%	8.1
Where?	47.0%	54.7%	5.9
Ensemble des 5 types ci-dessus	68.0%	71.7%	6.8

Selon les résultats présentés dans la table 5.2, cette pré-sélection permet de retrouver 68.0% des spans-réponses de l'ensemble des questions sélectionnées. Pour certains types de questions comme "When / What year?", on est capable d'en retrouver 79% ce qui reste

inférieur aux performances à l'état de l'art (89.9%). Après observation des sorties, diverses sources d'erreurs expliquent les limites de cette présélection:

- Les abréviations utilisant des points dans l'écriture des noms, prénoms, titres, etc compliquent la détection puisque parfois le système considère une fin d'entité.
- Un mauvais découpage des éléments temporels, comme par exemple avec le span-réponse "last weekend of september" qui est découpé par le système en deux entités: "last week end" et "September". En comptant tous les spans-réponses commençant et finissant par une entité nommée, on comptabilise que ce mauvais découpage arrive pour 2.18% des spans-réponses du train.
- Souvent les spans-réponses varient des entités nommées de quelques mots comme par exemple avec le span-réponse: "early 2012" alors que le système identifiant les entités nommées sélectionne seulement "2012".
- Bien qu'on soit capable d'identifier le type des questions, certaines questions ont un sens plus subtil que cela. Ainsi, le type de réponse ne correspond pas toujours à ce qui serait normalement attendu. Par exemple dans le cas de "Where does Warsaw rank in terms of population in the EU?", il est évident qu'il est inutile de pré-sélectionner des lieux pour cette question, pourtant de type "Where?".

Bien que cette méthode n'identifie pas directement le span-réponse, elle peut être utilisée comme critère d'attention du système sur certains éléments du texte. Aussi , cette présélection permet de restreindre l'espace de recherche à environ 7 spans-candidats ce qui est non négligeable si on cherche à réduire le temps de calculs des systèmes. Cependant, il faut nuancer cela car l'ensemble de questions étudiées ici ne représente qu'environ 35% du dataset total.

Pour pouvoir confirmer les heuristiques présentées en table 5.1 et étudier les limites d'une présélection par NER, on mesure les résultats maximum qu'il est possible d'obtenir avec cette présélection. Pour cela, on va retirer les restrictions au niveau des types de questions et d'entités nommés, que l'on s'est imposés précédemment (voir table 5.1). Ainsi, pour chacun des types de question identifiable (voir section 3.1), on conserve toutes les entités nommées identifiées par le système (voir table 2.7). On présente ces résultats en table 5.3.

Tab. 5.3. Résultats d'une présélection conservant, pour chaque type de question identifiable, la totalité des entités nommées (voir table 2.7).

	Train		
Type de question	ex. match	f1	size_set
Tous sauf "What VB[*]?"	49.39%	55.24%	12.8
Tous sauf "Other"	48.29%	54.49%	12.6
Tous sauf "What / Which NN[*]?"	47.81%	53.32%	12.5
Tous sauf "How?"	45.89%	51.93%	12.4
Tous sauf "What name / is called?"	44.87%	50.98%	12.4
Tous sauf "Where?"	44.83%	50.96%	12.4
Tous sauf "How much / many?"	43.04%	49.25%	12.0
Tous sauf "Who?"	42.58%	48.89%	12.2
Tous sauf "When / What year?"	41.33%	47.84%	12.1
Tous	45.23%	51.34%	12.4

Les mauvais résultats rapportés dans la table 5.3 confirment le fait qu'il n'est pas possible de répondre à toutes les questions à l'aide d'entités nommées. Cela est particulièrement visible dans le cas des questions de type "What VB[*]?". Dès lors qu'on ignore ce type de questions on observe que les performances de la présélection par entités nommées sont améliorées.

La comparaison des résultats de la table 5.3 avec ceux affichés en table 5.2 valide les heuristiques choisies. En effet, en éliminant les types de questions avec heuristique (voir table 5.1) on diminue les performances globales de cette présélection. Ainsi d'après ces résultats on note qu'il est important de conserver l'information apportée par le système détectant les entités nommées lors la sélection du span-réponse.

Étant donnée qu'une présélection par entités nommées est trop restrictive entraînant ainsi trop perte, dans le reste de notre étude, après avoir découpé le paragraphe en phrases, notre système va considérer tous les spans-candidats possibles à l'intérieur de chaque phrase.

5.3. Construction d'exemples négatifs

On cherche à construire des systèmes qui puissent reconnaître les span-réponses des autres en apprenant ce qui distingue un bon span-candidat d'un mauvais. Ainsi, il nous faut des exemples négatifs. C'est à dire des exemples de span qui ne sont pas des spans-réponses. Étant donné que SQuAD ne contient que les spans-réponses, il nous faut générer des exemples négatifs. Pour cela on combine deux méthodes de sélection:

- **Génération avec les entités nommées**: Pour chaque span-réponse on sélectionne aléatoirement au plus (dépendamment de la disponibilité), dix entités nommées dans le texte qui ne correspondent pas au span-réponse.
- **Génération avec les n-grams**: En excluant les entités nommées, pour chaque span-réponse on sélectionne aléatoirement au plus (dépendamment de la disponibilité), dix mauvais n-grams ($n < k$, où k est la taille de la phrase dans laquelle on extrait le n-gram) dans le paragraphe.

On présente en table 5.4 le nombre d'exemples utilisés pour l'entraînement de nos modèles.

Tab. 5.4. Statistiques des exemples d'apprentissage. Dans le cas des exemples positifs, on a distingué ceux qui coïncident avec des entités nommées des autres.

type d'exemple	NER	n-grams	**Total**
positifs	39 621	47 978	87 599
négatif	719 176	875 990	1 595 166
Total	758 797	923 968	1 682 765

Étant donné que les exemples positifs ne représentant qu'environ que 5% des données d'entraînements, ce nouvel ensemble est débalancé. Dans la section 5.5.1, nous étudierons l'influence du nombre et du type des exemples négatifs sur les performances de nos modèles. Basé sur ce nouvel ensemble de données, on en extrait un ensemble de caractéristiques (voir section 5.4) afin que nos modèles puissent déterminer si un span est un span-réponse ou non.

5.4. Famille de features

On présente ci-dessous l'ensemble des *features* extraits afin que les modèles réalisés puissent distinguer les spans-réponses des autres spans. Pour illustrer l'ensemble des familles de *features* sur lesquelles sont basés nos modèles, similairement à [Rajpurkar et al., 2016] on utilise le même exemple, que l'on présente en figure 5.4 . La figure A.1 permet une meilleure compréhension de certains *features*.

Fig. 5.4. Exemple de question réponse (sur SQuAD v.1.1) utilisé par les auteurs pour présenter leurs *features*. On dénotera par "Q" la question , par "P" la phrase contenant le span-candidat, et par "s" le span-candidat "gravity" choisit en exemple.

Paragraphe: `In meteorology, precipitation is any product of the condensation of atmospheric water vapor that falls under gravity. The main forms of precipitation include drizzle, rain, sleet, snow, graupel and hail... Precipitation forms as smaller droplets coalesce via collision with other rain drops or ice crystals within a cloud. Short, intense periods of rain in scattered locations are called ‘‘showers’’.`

Question ("Q"): `What causes precipitation to fall?`
Réponse et span-candidat ("s"): `gravity`
Phrase contenant le span-candidat ("P"): `In meteorology, precipitation is any product of the condensation of atmospheric water vapor that falls under gravity.`

Voici la liste des familles *features* que l'on utilise:

(1) **Forme du span, de la phrase et de la question:** Famille de **36** *features* de base. Celle-ci comprend la taille (en caractères et en tokens) du span-candidat, ainsi que des booléens de présence de chiffres et de majuscules avec leurs comptes relatifs dans le span-candidat. Les mêmes *features* sont calculés de façon analogue pour le span-candidat normalisé (voir section 3.3), la phrase et la question.

E.g., "s" contient sept caractères et est composé d'un mot. Il ne comporte pas de chiffre ni de majuscule.

(2) **Position:** Famille de **6** *features* comprenant la position relative du span-candidat (en caractères et en mots) dans la phrase et dans le paragraphe, ainsi que la position relative de la phrase (en caractères et en mots) dans le paragraphe.

E.g., les positions relatives en caractères et en mots de "s" dans "P" sont de 0.93 et 0.94 . "P" contient 116 caractères et 18 mots et "s" apparaît aux 109 caractères et aux 17 mots de "P".

(3) **Similarités embeddings:** Cette famille comprend **20** mesures de similarités cosinus entre le span-candidat, son contexte et la question. Pour un span-candidat donné, soit p sa position de début (en mot) dans la phrase et t sa taille (en mots). En plus du span-candidat, on extrait huit span-contextes allant des positions $p - i$ à $p + t - 1 + j$ (avec i,$j \in [0,2]$ et $i + j > 0$). On calcule ensuite la similarité cosinus entre la question et chacun de ces spans. On mesure également la similarité entre la phrase et la question.

Afin d'extraire l'essentiel du questionnement on obtient à l'aide de CoreNLP[1] une forme réduite de la question. Cette dernière est composée du pronom interrogatif de la question ainsi que de l'ensemble des mots ayant une relation grammaticale directe avec le pronom interrogatif. De manière analogue on réalise les mêmes calculs de similarités avec la forme réduite de la question.

E.g., La similarité cosinus obtenue entre "s" et "Q" est de 0.39, la similarité obtenue entre le span-contexte "falls under gravity" et "Q" est de: 0.54 .

(4) **Similarités lemmes: 2** features comptant le nombre de lemmes commun au span-candidat et la question sur la taille en mots du span-candidat et sur la taille en mots

[1] `https://stanfordnlp.github.io/CoreNLP`

de la question.

E.g., les deux comptes des lemmes partagés sont nuls car "s" et "Q" n'ont aucun mot commun.

(5) **PoS-taging unigrams et bigrams:** À l'aide des 53 étiquettes grammaticales spécifiques à l'anglais (issues de OntoNotes 5) que propose le PoS-tagger de SpaCy, on identifie les catégories grammaticales des unigrams. De manière analogue on réalise cela pour les bigrams du span-candidat en utilisant les 18 étiquettes grammaticales issues de l'Universal Dependencies scheme[2]. On obtient ainsi un vecteur de **377** booléen.

E.g., la seule étiquette grammaticale de l'unigram contenu dans "s" est NN.

(6) **PoS-tag premier mot:** Vecteur de dimension **53** encodant l'étiquette grammaticale du premier mot du span-candidat.

E.g., l'étiquette grammaticale du premier mot de "s" est NN.

(7) **NER span:** Vecteur de **18** booléens encodant les catégories d'entités nommées présentes dans le span-candidat détectée à l'aide de SpaCy.

E.g., aucune entité nommée n'est présente dans "s".

(8) **Cohésion du span:** vecteur de **3** booléens indiquant si le span-candidat correspond exactement à:

- un groupe nominal
- une entité nommée
- un sous-arbre complet de l'arbre d'analyse en constituant de la phrase.

E.g., comme on peut le voir à la figure A.1, après analyse en constituant de la phrase, on voit que "s" est une feuille de l'arbre résultant, c'est pourquoi le booléeen correspondant à un sous-arbre complet est donc activé.

(9) **Structures fréquentes:** Un vecteur de **10** booléens représentant dix structures fréquemment observées à l'aide de l'analyse en dépendance sur les spans-réponses. Par exemple on active un booléen lorsque le span-candidat contient un groupe nominal précédent/suivant un verbe partagé (après lemmatisation) à la fois par la phrase et par la question.

E.g., il y a une correspondance entre les lemmes de "fall" et "falls" et "gravity"

[2]https://universaldependencies.org/u/pos/

est un NP suivant "falls", on active donc le booléen indiquant que le span-candidat contient un groupe nominal précédent/suivant un verbe partagé entre la phrase et par la question.

(10) **Analyse en dépendance non lexicalisée:** Cette famille contient **1200** booléens d'informations concernant les relations de dépendances grammaticales menant au premier mot des spans-réponses (exemples positifs) du train (voir section 5.5.2 pour le choix de la taille de cette famille). Ainsi à l'aide de l'analyseur en dépendance de CoreNLP[3] , pour chaque span-réponse du train, on extrait tous les chemins, de longueur inférieure à quatre, entrant ou sortant du premier mot du span-réponse. Chaque chemin correspond à l'alternance des étiquettes grammaticales des mots et des relations syntaxiques les reliant. Finalement, chacun des *features* booléens correspond à l'un des chemins les plus fréquents. On présente en table 5.6 les relations les plus fréquemment observées.

E.g., une relation identifiée est: IN $\xrightarrow{pobj}$ NN liant "under" à "gravity". Aussi il y a VBZ $\xrightarrow{prep}$ IN $\xrightarrow{pobj}$ NN liant de gauche à droite "falls" à "under" et à "gravity".

(11) **Analyse en dépendance lexicalisée** Famille de **7000** features similaire à celle décrite précédemment, si ce n'est que l'on ajoute le lemme du premier mot du span-candidat en avant de la relation (voir section 5.5.2 pour le choix de la taille de cette famille). On présente en table 5.7 les relations les plus fréquemment observées.

E.g., aucun *feature* de relation est allumé car les spans-réponses commençant par "gravity" sont trop peu fréquent pour en conserver les relations.

Comparativement au modèle proposée par [Rajpurkar et al., 2016] qui utilise plus de 180 millions de *features*, nos modèles en utilisent seulement 8725, ce qui est nettement moins coûteux. Aussi, un autre avantage de nos *features* est qu'ils sont facilement extractibles automatiquement. On présente dans un premier temps l'ensemble de nos résultats obtenus en utilisant ces features avec la régression logistique afin de les comparer à [Rajpurkar et al., 2016], avant d'étudier les résultats obtenus pour les modèles SVM et les forêts aléatoires.

[3]https://stanfordnlp.github.io/CoreNLP/

5.5. Modèle basé sur la régression logistique

La plupart de nos expériences ont été réalisées avec un système qui sélectionne en premier la phrase-réponse (avec `[1-9]-gram_sent_cos` section 4) avant de déterminer le span-réponse à l'aide d'une régression logistique construite avec scikit-learn[4]. Basé sur les exemples d'apprentissage (voir section 5.3) on entraîne une régression logistique afin qu'elle détermine si un span donné est un span-réponse ou non. Le système utilise ensuite cette régression pour prédire les spans-réponses, c'est à dire tout ceux qui ont respectivement la plus haute probabilité d'être un span-réponse. Après avoir présenté l'ensemble de nos expériences avec cette première architecture de système, en fin de section, on s'intéresse à système analogue qui lui va réaliser le span-réponse directement dans la totalité du paragraphe.

5.5.1. Variation du nombre d'exemples négatifs

Dans le but de pouvoir observer l'influence qu'a le débalancement de nos données d'entraînement, nous faisons varier le nombre d'exemples négatifs considéré. Afin de ne pas biaiser nos résultats en réalisant les observations et l'évaluation sur le corpus dev, on réalise nos observations sur une tranche de train. Pour cela, on segmente aléatoirement train en deux sous-ensembles:

- train1 est un ensemble d'entraînement composé de 77 599 questions avec leur ensemble d'exemples (positifs et négatifs),
- similaire à dev en nombre de questions, dev1 un ensemble de 10 000 questions pour s'évaluer.

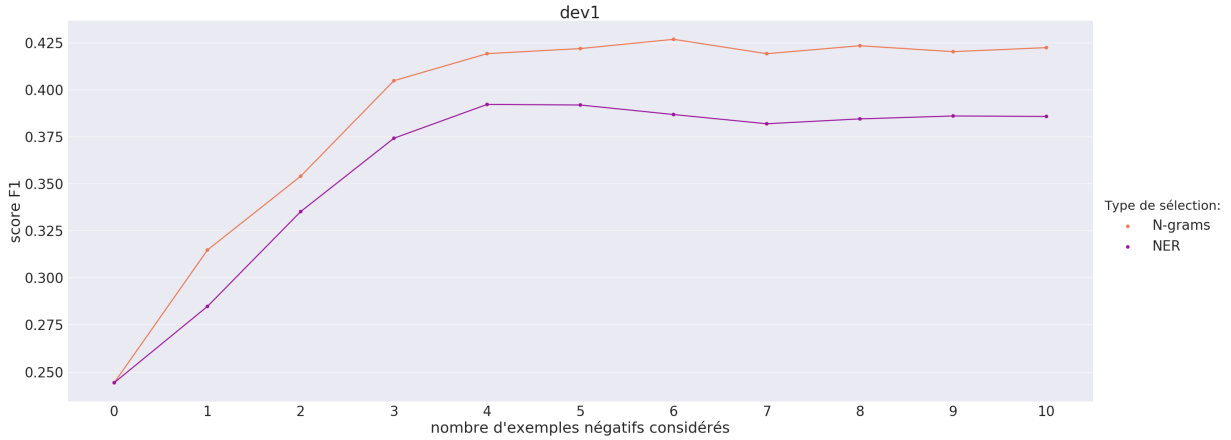
De manière séparée pour le corpus train1, on fait varier d,$e \in [0,10]$ où d correspond au nombre d'exemples négatifs générés par n-grams et e le nombre d'exemples négatifs générés par NER. Ainsi pour chaque d et e, on entraîne sur le corpus train1 un modèle de régression logistique. N'ayant pas encore fixé la taille des deux dernières familles de *features* elles sont omisent durant la réalisation de nos mesures. On présente en figure 5.5 les performances obtenues sur dev1 en fonction des paramètres d et e.

D'après les résultats présentés, il ressort que les meilleures performances sont obtenues en prenant pour chaque span-réponse au maximum:

- six exemples négatifs issus de la génération par n-grams,

[4]https://scikit-learn.org/stable/index.html

Fig. 5.5. Performances obtenues en fonction du nombre d'exemples négatifs considérés sur dev1.



- quatre exemples négatifs issus de la génération par entités nommées.

On explique la différence de performances observée sur les deux courbes de la figure 5.5 par le fait qu'étant donné que la très grande majorité des spans-candidats sont des n-grams, la prise en compte de n-grams en tant qu'exemples négatifs permet d'obtenir de meilleurs performances. Durant nos expériences on constate que l'ajout d'exemples négatifs de type NER diminue les performances à distinguer les exemples positifs des négatifs sur train1. Cela s'explique par le fait que les entités nommées ont plus de chances de correspondre à un span-réponse qu'un n-gram tiré aléatoirement (voir section 5.2). Étant donné que les entités nommées sont le résultat d'une analyse fine des textes, celles-ci sont toujours censées contrairement à un n-gram tiré aléatoirement qui lui peut être une suite illogique de mots. L'intérêt d'ajouter quelques NER en exemples négatifs permet de dérouter légèrement le modèle avec des réponses plausibles et d'éviter ainsi le sur-apprentissage sur les données d'entraînement. On présente en table 5.5 le nombre final d'exemples utilisés par nos modèles.

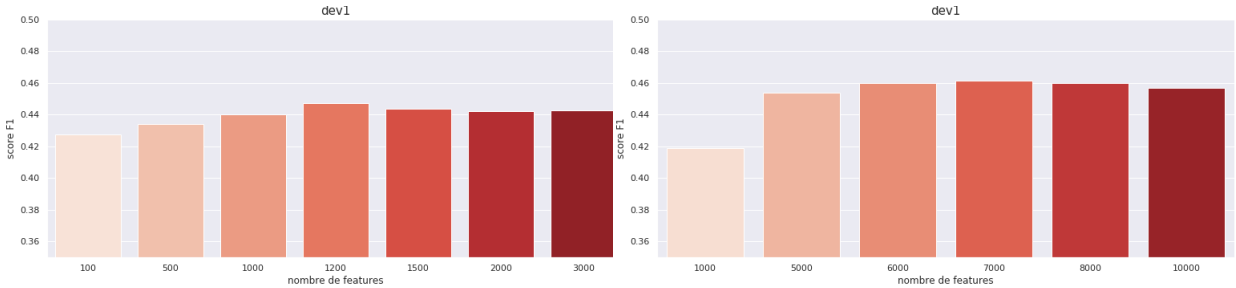
Tab. 5.5. Statistiques des exemples d'apprentissage

type d'exemple	NER	n-grams	**Total**
positifs	39 621	47 978	87 599
négatif	324 457	525 594	850 051
Total	364 078	573 572	937 650

5.5.2. Tailles des deux dernières familles de *features*

Afin de maximiser les résultats en utilisant le minimum de *features*, on doit déterminer la taille des deux dernières familles de *features*. Ainsi, sur le corpus train nous avons identifié 6 481 relations pour la famille "Analyse en dépendance non lexicalisée" après avoir éliminé toutes celles qui n'apparaissent qu'une fois on en conserve 3 979. De manière analogue on réalise cela pour l'"Analyse en dépendance lexicalisée". Après avoir collecté et lemmatisé l'ensemble des premiers mots des spans-réponses, on obtient un ensemble de 23 157 lemmes. Par la suite on s'intéresse uniquement aux 8 902 lemmes apparaissant plus d'une fois. Ainsi, on comptabilise au total 36 045 relations desquelles sont conservées 21 023 relations qui apparaissent plus d'une fois. En se basant sur les fréquences des relations obtenues sur le corpus train1, pour chacune de ces deux familles, on construit une régression logistique basée sur tous les autres *features* plus la famille d'intérêt. Nous avons fait varier la taille de chacune de ces familles, on présente en figures 5.6 les résultats sur dev1 de ces variations.

Fig. 5.6. Score F1 obtenu en faisant varier la taille des deux dernières familles de *features* (à gauche: Analyse en dépendance non lexicalisée, à droite: Analyse en dépendance lexicalisée)



Au vu de ces résultats, on conserve du corpus train les 1200 relations les plus fréquentes pour la famille "Analyse en dépendance non lexicalisée" et les 7000 plus fréquents pour "Analyse en dépendance lexicalisée". Afin d'illustrer quelques relations conservées, on présente en figure 5.6 et en figure 5.7 les relations les plus fréquemment observées pour chacune des deux familles.

Tab. 5.6. Relations le plus fréquemment observées pour l'"Analyse en dépendance non lexicalisée"

Relations identifiées	**Informations capturées**
NNS $\xleftarrow{compound}$ NNS	relation liant deux noms pluriels complémentaires.
DT $\xleftarrow{det}$ NN	relation liant un déterminant à un nom.
NN $\xleftarrow{compound}$ NN	relation liant deux noms singuliers complémentaires.
JJ $\xleftarrow{amod}$ NNS	relation liant un ajectif à un nom pluriel.
IN $\xleftarrow{case}$ NN	relation liant une préposition ou une conjonction de subordination à un nom.

Tab. 5.7. Relations le plus fréquemment observées pour l"Analyse en dépendance lexicalisée"

Relations identifiées	**Informations capturées**
the : DT $\xleftarrow{det}$ NNP	relation liant *the* à un nom propre.
the : DT $\xleftarrow{det}$ NN	relation liant *the* à un nom singulier.
the : DT $\xleftarrow{det}$ NN $\xleftarrow{compound}$ NN	relation liant *the* à deux noms singuliers complémentaires.
a : DT $\xleftarrow{det}$ NN	relation liant *a* à un nom singulier.
the: DT $\xleftarrow{det}$ NN $\xleftarrow{nmod}$ VBN	relation liant *the* à un nom lui même relié à un verbe conjugé à la 3ème personne du singulier.

5.5.3. Modèle unique versus multi-modèles

On s'est ensuite intéressé à déterminer s'il est préférable d'utiliser un unique modèle général pour l'ensemble des questions ou d'avoir un modèle spécifique à chacun des neuf types de questions (voir section 3.1). Dans le cas du multi-modèles, on a entraîné sur le corpus train1 et testé sur dev1 une régression logistique pour chaque type de question. Ainsi pour réaliser une prédiction, on identifie, lorsque cela est possible le type de la question,

sinon on lui attribue le type "*Other*" puis on utilise la régression logistique associée au type identifié. Afin de pouvoir se comparer à [Rajpurkar et al., 2016] et étant donné que les observations réalisées sur le corpus dev1 sont similaires, on présente en table 5.8 uniquement les résultats obtenus après un entraînement sur train et une évaluation sur dev.

Tab. 5.8. Comparaison des résultats entre un unique modèle versus un modèle pour chaque type de question.

	dev	
régression logistique	ex. match	F1
model unique	41.23%	50.37
multi-modèles	46.94%	55.68
[Rajpurkar et al., 2016]	40.00%	51.00

Au travers de ces premiers résultats, il est visible que les *features* sur lesquels se base notre régression logistique permettent de capturer plus d'informations que les *features* de [Rajpurkar et al., 2016]. Un deuxième fait notable est que le fait de construire une régression logistique pour chacun des neuf types de questions permet d'améliorer les performances. Cela s'explique par le fait que chaque type de questions a ses spécificités. Ainsi, en construisant une régression logistique pour chaque type de question on capture mieux l'ensemble des spécificités de chaque type de question, ce qui permet donc d'obtenir de meilleurs résultats lors de la sélection des spans-réponses. Dans le reste de cette section, les résultats présentés sont obtenus avec des modèles composés de neuf régressions logistiques (car il y a neuf types de questions possibles en comptant "*Other*").

5.5.4. Sélection de *features*

On s'intéresse maintenant à comprendre l'utilité de chaque famille de *features*. Soit U l'ensemble des familles de *features* sélectionnées, et F l'ensemble des familles de *features* restantes. On initialise U à l'ensemble vide. On ajoute alors de manière itérative la meilleure famille de F et ce jusqu'à ne plus parvenir à augmenter les performances mesurées sur dev1, voir à les dégrader. À l'aide d'identificateurs significatifs on présente ci-dessous la liste des famille de *features* sélectionnées classées dans l'ordre de sélection, le numéro de la famille est rappelé entre parenthèses (voir section 5.4):

NER: NER Span (7)

SE: Similarités embeddings (3)

SL: Similarités lemmes (4)

DNP: Analyse en dépendance non lexicalisée (10)

DL: Analyse en dépendance lexicalisée (11)

PoS: PoS-taging unigrams et bigrams (5)

Fo: Forme du span, de la phrase et de la question (1)

Po: Position (2)

De manière analogue aux résultats présentés à la table 5.8, bien que nos observations on étés réalisées à l'aide des corpus train1 et dev1, étant donné que celles-ci sont similaires à celles réaliser sur dev et pour pouvoir se comparer avec [Rajpurkar et al., 2016], on présente en table 5.9 les performances obtenues à chaque étape de sélection sur dev.

Tab. 5.9. Résultats obtenus à chaque incrément de la sélection de *features*

	dev	
Familles sélectionnées	ex. match	F1
NER	15.16%	21.32
NER + **SE**	26.78%	31.14
NER + **SE** + **SL**	30.27%	35.92
NER + **SE** + **SL** + **DNL**	36.45%	41.09
NER + **SE** + **SL** + **DNL** + **DL**	39.91%	46.18
NER + **SE** + **SL** + **DNL** + **DL** + **PoS**	43.54%	51.70
NER + **SE** + **SL** + **DNL** + **DL** + **PoS** + **Fo**	45.78%	54.11
NER + **SE** + **SL** + **DNL** + **DL** + **PoS** + **Fo** + **Po**	46.01%	54.99
Toutes les familles	46.94%	55.68
[Rajpurkar et al., 2016]	40.00%	51.00

Au travers de ces résultats, on constate que la sélection de familles de *features* ne permet pas d'améliorer les performances de notre modèle. En effet, avec sélection de *features*, on obtient des performances en termes d'exact match de 46.01% contre 46.94% si aucune sélection n'est opérée. Ainsi pour les modèles suivants, nous allons utiliser toutes les familles de

features.
Un autre fait notable qui vient confirmer les observations réalisées à la section 5.2 est qu'au premier abord, il y a beaucoup d'informations à capturer aux niveaux des entités nommées puisqu'ici la famille "NER Span" est sélectionnée en premier et permet d'obtenir à elle seule 15.16% d'exact match. Enfin on remarquera les familles composées de *features* plus précises comme "Analyse en dépendance non lexicalisée" et "Analyse en dépendance lexicalisée" sont dépendantes de l'information collectée par d'autres familles de *features* plus généraux.

5.5.5. Ajustement des métaparamètres

Pour pouvoir déterminer les meilleurs métaparamètres de notre régression logistique parmi ceux disponibles, on les fait varier. Par défaut, scikit-learn utilise la pénalité L2 et l'algorithme "lbfgs" comme solveur (algorithme permettant l'optimisation des paramètres). À l'aide de l'algorithme de *grid search* de scikit-learn basé sur diverses combinaisons de métaparamètres, on entraîne sur train1 des régressions logistiques afin de determiner quelle combinaison donne les meilleurs résultats sur dev1. On essaye chacune des régularisations suivantes: {L1, L2, elasticnet, aucune} combinée (lorsque cela est possible) avec chacun des solveurs suivants: {newton-cg, lbfgs, sag, saga}. Il résulte des nos expériences que les meilleurs résultats sont obtenus à l'aide de la pénalité L2 et le solveur "lbfgs". En conséquence, on conserve les métaparamètres pour l'obtention de nos résultats.

5.5.6. Comparaison de deux architectures

Comme expliqué en début de section, nous avons réalisé deux architectures de modèle. La première `"modele_phrase_span"` sélectionne la phrase-réponse avant d'y sélectionner le span-réponse. `"modele_span"` cherche directement le span-réponse dans le paragraphe. Bien que nos observations on été réalisées sur dev1, étant donné que celles-ci sont similaire à celles obtenues sur dev on présente en table 5.10 les résultats que l'on obtient avec les deux architectures uniquement sur dev afin de pouvoir comparer tout les modèles entre eux et avec celui de [Rajpurkar et al., 2016].

Tab. 5.10. Comparaison des resultats entre un unique modèle versus un modèle pour chaque type de question.

	dev	
regression logistique	ex. match	F1
`modele_phrase_span`	46.94%	55.68
`modele_span`	48.47%	57.50
[Rajpurkar et al., 2016]	40.00%	51.00

Les résultats confirment les observations réalisées au chapitre 4. En effet, ils montrent que "`modele_span`" qui extrait directement le span-réponse sur l'entièreté du paragraphe donne de meilleurs résultats que "`modele_phrase_span`". Avec ces résultats, on peut affirmer qu'il n'est pas nécessaire de présélectionner la phrase-réponse avant d'y sélectionner le span-réponse. En effet, étant donné qu'il y a une perte au niveau de la séléction des phrases-réponses cela va restreindre les performances du système. Il vaut donc mieux lui laisser seul reconnaître les éléments du paragraphe qui sont des spans-réponses. En conséquence, l'ensemble de nos modèles suivants travaille directement sur l'ensemble du paragraphe.

5.6. Détection à l'aide des forêts aléatoires

Basé sur les *features* décrit en section 5.4 et sélectionnés en section 5.5.4, on entraîne un modèle de forêt aléatoire. Comme pour la régression logistique, on entraîne sur le corpus train1 un modèle propre à chaque type de question. Notre système détermine donc dans un premier temps le type de la question afin de sélectionner le modèle correspondant pour réaliser la prédiction du span-réponse.

De manière analogue à la régression logistique on optimise les méta-paramètres de notre système parmi ceux disponibles et testés sur le corpus dev1. Ainsi, on fait varier le nombre d'arbres avec les valeurs suivantes: {100, 500, 1000, 2500, 5000} combinés avec "gini" ou avec l'"entropie" comme critère mesurant la qualité de chaque sous ensemble de données. Il ressort de nos expériences que la construction de 1000 arbres avec "gini" donne la meilleure performance. On présente en table 5.11 les résultats obtenus par ce modèle.

5.7. Détection à l'aide des SVM

Similairement à ce qu'on a réalisé pour la régression logistique, on construit ensuite un système basé sur des modèles SVM. Ainsi, pour prédire le span-réponse dans l'intégralité du paragraphe, on utilise un modèle en fonction du type de question.

Nous avons essayé divers noyaux: {"polynomial", "rbf", "sigmoïde"}, avec les valeurs de paramètres suivant:

- γ: $\{10^{-2}, 10^{-3}, 5 \times 10^{-4}, 10^{-4}, 5 \times 10^{-5}, 10^{-5}\}$
- d: {3,5,7,10} (pour le noyau polynomial)

Nos expériences montrent que le modèle SVM à noyeau gaussien avec $\gamma = 10^{-4}$ donne les meilleures performances pour la sélection des spans-réponses. Les résultats présentés en table 5.11 sont obtenus avec ces paramètres.

5.8. Résultats des modèles simples

Bien que les observations on été réalisés sur dev1, étant donné que les résultats obtenus sont similaires à ceux obtenus sur dev, on présente en table 5.11 l'ensemble des résultats obtenus avec chaque modèle testé sur dev pour pouvoir se comparer avec le reste des modèles.

Tab. 5.11. Résultats obtenus pour l'ensemble des modèles simples

	dev	
modèle	ex. match	F1
SVM	52.29%	61.42
Forêt aléatoire	50.53%	59.94
Régression logistique	48.47%	57.50
[Rajpurkar et al., 2016]	40.00%	51.00
[Wang and Jiang, 2016]	54.50%	67.74

Les résultats montrent, qu'avec 52.29% d'exact match, le modèle SVM lui dépasse l'ensemble des autres modèles. Ainsi avec des *features* simples extraits automatiquement nous avons réussi à dépasser largement notre modèle de référence [Rajpurkar et al., 2016]. Avec 50.53% en score d'exact match le modèle forêt aléatoire dépasse lui aussi la régression logistique. En effet, en entraînant des arbres sur des sous-ensembles de *features* et sur des

sous-ensembles du dataset, on construit des arbres plus spécifiques. En les combinant, cela permet d'être plus précis au niveau de la sélection du span-réponse. Bien que notre régression logistique donne les moins bons résultats, ses performances sont tout de même honorables puisque elle dépasse d'environ 8% le modèle de référence [Rajpurkar et al., 2016]. On note que les performances obtenues par nos modèles sont de même ordre que celles obtenues par les premiers modèles neuronaux testés sur SQuAD comme par exemple le match-LSTM réalisé par [Wang and Jiang, 2016] qui obtient 54.5% d'exact match.

5.9. Combinaison des modèles

Dans le but de prendre avantage le plus possible des forces de chacun des divers systèmes construits, on tente dans cette section de les combiner. Étant donné que pour chacun de nos systèmes on a construit un modèle spécifique à chaque type de question, on s'intéresse à mesurer les performances des modèles en fonction du type de question. Ainsi, il est ensuite facile de sélectionner pour chaque type de question le modèle donnant les meilleurs résultats en terme d'exact match. Bien que nous avons fait nos observations sur le corpus dev1, étant donné que celles-ci sont similaires a celles obtenues sur le corpus dev et afin d'avoir l'évaluation finale détaillée de nos modèles en fonction des questions, on présente en table 5.12 uniquement les performances obtenues sur dev.

Au vu des résultats obtenus on va utiliser le modèle SVM pour tous les types de questions, exceptées les questions de type "What VB[*]?" et "How much / many?" où l'on utilisera le modèle de foret aléatoire. Aussi on note que les résultats de la table 5.12 concordent avec les résultats et les observations réalisées à la table 5.3. En effet, on peut noter que les questions de types: "What VB[*]?", "How?" et "Other" se distinguent des autres par le fait qu'elles donnent des performances nettement inférieures à celles obtenues pour l'ensemble des autres types de questions. Après avoir observé les spans-réponses attendus pour ces types de questions on explique cette différence de performance par la trop grande diversité des span-réponses attendus pour ces types de question. En effet, comparativement aux autres types de questions, il est difficile d'établir des caractéristiques communes pour les spans-réponses des questions "What VB[*]?", "How?" et "Other". Par exemple, les spans-réponses de type "Where?" ont souvent comme caractéristique d'être des entités nommées de dates ou de contenir des nombres.

Tab. 5.12. Résultats des modèles en fonction du type de question sur dev (la meilleure performance obtenue pour chaque type de question a été mise en gras)

	SVM		**Forêt aléatoire**		**R. logistique**	
Type de question	ex. match	f1	ex. match	f1	ex. match	f1
What / Which NN[*]?	**51.31**	62.63	49.03	60.57	47.44	58.32
Who?	**67.74**	74.36	65.71	72.49	63.2	70.24
What VB[*]?	31.3	41.57	**32.18**	41.24	29.78	38.05
When / What year?	**76.93**	82.33	75.01	80.61	72.95	78.41
How much / many?	75.88	81.53	**76.02**	81.73	72.69	79.2
Where?	**53.71**	61.61	50.69	60.01	48.63	57.51
How?	**34.15**	41.38	32.31	42.12	30.33	39.64
What name / is called?	**44.65**	54.96	44.2	53.5	41.78	51.04
Other	**36.87**	45.97	32.99	43.71	31.07	41.25
Moyenne	52.29	61.42	50.53	59.94	48.47	57.50

On présente en table 5.13 les résultats obtenus lorsque l'on réalise cette combinaison de modèle.

Tab. 5.13. Résultats obtenus pour l'ensemble de tout nos modèles

	dev	
modèle	ex. match	F1
Modèle combiné	52.41%	61.39
SVM	52.29%	61.42
Foret aléatoire	50.53%	59.94
Régression logistique	48.47%	57.50
[Rajpurkar et al., 2016]	40.00%	51.00

En combinant les modèles de SVM et de forêt aléatoire, on arrive à augmenter légèrement les performances en terme d'exact match, le score F1 est cependant pratiquement inchangé.

5.10. Limites des modèles et des *features*

Avant de conclure notre étude, on s'intéresse aux limites de nos modèles. Pour cela, on observe les sorties produites par notre meilleur modèle, celui combinant les modèles de SVM et de forêt aléatoire. Ainsi, on note que dès qu'un span-réponse possède un premier mot peu fréquent, le modèle a de la difficulté à le prédire. En effet, si un span possède un premier mot peu fréquent, aucun *feature* appartenant à la famille "Analyse en dépendance lexicalisée" ne va s'activer, le modèle doit donc s'appuyer sur d'autres *features* plus généraux ce qui peut induire le modèle en erreur. On illustre nos propos avec les exemples présentés en figure 5.7.

Les spans-réponses des deux questions présentées en figure 5.7 ont des lemmes de premier mot peu fréquents dans le corpus train. Dans l'exemple 1 en plus du fait qu'aucun *feature* de la 11ème famille n'est activé, les mots de la phrase dans laquelle apparaît "Bert Bolin" ne présentent pas de grande similitude avec les mots questions, c'est pour cela que le modèle porte son attention sur la première phrase et retourne le span correspondant à une entité nommée de personne: "Hoesung Lee". Contrairement au span retourné dans l'exemple 1 qui n'active aucun *feature* de la 11ème famille, dans l'exemple 2, "market economy" active la *feature* correspondante à "market NN $\xleftarrow{compound}$ NN", ainsi bien les phrases desquelles proviennent "Neoclassical economics" et "market economy" possèdent toutes deux des mots proches de la question, le modèle préfère choisir "market economy". On voit donc ici que nos *features* lexicalisés biaisent le modèle vers ce qu'il a vu durant l'entraînement.

Lors de nos expérimentations (basées sur train1 et dev1), on note en observant les sorties produites par notre modèle combiné que celles-ci sont censées et qu'elles sont fortement basées sur l'information des entités nommées. Après étude on note sur dev1 que 66.17% des spans produits sont composés d'entités nommées. On présente en figure 5.8 un exemple d'erreur réalisée par le modèle qui répond par une entité nommée.
En premier lieu, remarquons que l'exemple en figure 5.8 présente une incomplétude de SQuAD puisque "the Amazon Jungle" semble être une réponse valide à la question posée. Dans le cas de "also known in English as Amazonia or the Amazon Jungle", les *features* de la famille "Analyse en dépendance lexicalisée" tels que "also : RB $\xleftarrow{advmod}$ VBN" et "also : RB $\xleftarrow{advmod}$ VBN $\xrightarrow{nmod}$ NNP" sont activés. Concernant "the Amazon Jungle", ce sont les *features* "the : DT $\xleftarrow{det}$ NN", "the : DT $\xleftarrow{det}$ NN", "the : DT $\xleftarrow{det}$ NN $\xrightarrow{compound}$ NNP", "the :

DT $\xleftarrow{det}$ NN $\xleftarrow{conj}$ NNP" qui sont activés. Ici on voit que le modèle retourne l'entité nommé "the Amazon Jungle" de type "LOC" (voir table 2.3) car même si le span "also known in English as Amazonia or the Amazon Jungle" l'inclut aussi le modèle préfère choisir un span court d'autant plus qu'il possède un fort ratio de mots partagés avec la question (famille "Similarités lemmes"). Les autres spans-réponse "Amazonia or the Amazon Jungle" et "Amazonia" ne sont pas considérés par le modèle car leur lemme de premier mot est peu fréquent dans le corpus train. Ainsi, ces deux spans n'activent aucun *feature* dans la famille "Analyse en dépendance lexicalisée".

On s'interroge sur la pertinence de construire de nombreux *features* à partir du premier mot du span-réponse (voir les familles 10 et 11 de *features* à la section 5.4). Ainsi, de manière analogue, nous avons essayé des *features* basés sur le dernier mot des spans-réponses. Cependant, nous ne les présentons pas ici car leur ajout diminue les performances des modèles. On justifie l'intérêt des *features* construit à partir du premier mot du span-réponse par le fait que les spans-réponse de SQuAD sont courts. En effet, comme dit à la section 3.1, 33% des spans du train contiennent juste un mot, c'est dans ce cas où partir du premier mot prend le plus d'intérêt puisqu'on peut récupérer toutes les relations grammaticales du span.

On étudie également la nature du premier mot de span-réponse. Ainsi en utilisant la liste de mots vides proposés par NLTK on s'aperçoit que dans la famille de *features* "Analyse en dépendance lexicalisée", 31.03% des relations identifiées on pour premier mot un *stop word* (ex: the, an, ...). Ce fort pourcentage peut là aussi mettre en cause la pertinence du premier mot des spans-réponses. Nous avons essayé alors de modifier les *features* afin de conserver uniquement les relations des spans-réponses ayant un premier de mot ne faisant pas parti des mots vides. La aussi les performances des modèles ont été réduites. Après observation sur dev1 on constate que 37% des spans-réponses de SQuAD débutent par un stop-words, ceci témoigne de l'intérêt à conserver les relations des spans-réponses commençant par un mot vide.

Fig. 5.7. Deux exemples, issus du corpus dev, de questions pour lesquelles notre système ne parvient pas à identifier les bons span-réponses. (Les spans-réponses attendus et retournés ont respectivement étés mis en vert et surlignés en rose.)

Exemple 1

Paragraphe: `Korean economist Hoesung Lee is the chair of the IPCC since October 8, 2015, following the election of the new IPCC Bureau. Before this election, the IPCC was led by his vice-Chair Ismail El Gizouli, who was designated acting Chair after the resignation of Rajendra K. Pachauri in February 2015. The previous chairs were Rajendra K. Pachauri, elected in May 2002; Robert Watson in 1997; and Bert Bolin in 1988. The chair is assisted by an elected bureau including vice-chairs, working group co-chairs, and a secretariat.`

Question: `Who was the first chair of the IPCC?`
Span-réponse attendu: `Bert Bolin`
Span-réponse retourné: `Hoesung Lee`

Exemple 2

Paragraphe: `Neoclassical economics views inequalities in the distribution of income as arising from differences in value added by labor, capital and land. Within labor income distribution is due to differences in value added by different classifications of workers. In this perspective, wages and profits are determined by the marginal value added of each economic actor (worker, capitalist/business owner, landlord). Thus, in a market economy, inequality is a reflection of the productivity gap between highly-paid professions and lower-paid professions.`

Question: `What philosophy of thought addresses wealth inequality?`
Span-réponse attendu: `Neoclassical economics`
Span-réponse retourné: `market economy`

Notons que cependant il reste de nombreuses autres *features* lexicaux basées sur les dépendances grammaticale à explorer comme par exemple comptabiliser le nombre de relations d'un certain type...

Fig. 5.8. Exemple issu du corpus dev, de question pour laquelle notre système ne parvient pas à identifier les bons span-réponses et retourne un span basé sur une entité nommée. (Les spans-réponses attendus et retournés ont respectivement étés mis en vert et surlignés en rose.)

Paragraphe: The Amazon rainforest (Portuguese: Floresta Amazônica or Amazônia; Spanish: Selva Amazónica, Amazonía or usually Amazonia; French: Forêt amazonienne; Dutch: Amazoneregenwoud), also known in English as Amazonia or the Amazon Jungle, is a moist broadleaf forest that covers most of the Amazon basin of South America. This basin encompasses 7,000,000 square kilometres (2,700,000 sq mi), of which 5,500,000 square kilometres (2,100,000 sq mi) are covered by the rainforest. This region includes territory belonging to nine nations. The majority of the forest is contained within Brazil, with 60% of the rainforest, followed by Peru with 13%, Colombia with 10%, and with minor amounts in Venezuela, Ecuador, Bolivia, Guyana, Suriname and French Guiana. States or departments in four nations contain "Amazonas" in their names. The Amazon represents over half of the planet's remaining rainforests, and comprises the largest and most biodiverse tract of tropical rainforest in the world, with an estimated 390 billion individual trees divided into 16,000 species.

Question: Which name is also used to describe the Amazon rainforest in English?
Span-réponse attendu: also known in English as Amazonia or the Amazon Jungle, Amazonia or the Amazon Jungle, Amazonia
Span-réponse retourné: the Amazon Jungle

Chapter 6

Conclusion

Dans ce mémoire nous avons exploré plusieurs aspects du Question-Réponse sur le jeu de données SQuAD. En premier lieu, on s'est attelé à chercher la phrase-réponse, avant la recherche du span-réponse.

Dans le cas de l'identification des phrases-réponses, nous avons montré qu'avec les *embeddings* et la segmentation des phrases et des questions, il est possible d'obtenir des résultats honorables (81%) compétitifs et même parfois meilleurs que certains systèmes neuronaux. On explique ces étonnantes performances par le fait que généralement dans SQuAD, les questions sont une reformulation des phrases-réponses, c'est pourquoi la difficulté ne se situe pas dans l'identification des phrases-réponses mais bien au niveau du span-réponse.

Concernant l'identification des spans-réponses, en s'inspirant du modèle de régression logistique de [Rajpurkar et al., 2016], nous avons extrait nos propres *features* avant de construire divers modèles. Nos systèmes sont nettement plus simples puisque comparativement aux 180 millions de *features* qu'utilisent [Rajpurkar et al., 2016], on en extrait ici 8725.

On a montré que les systèmes performent mieux si l'on construit un modèle propre à chaque type de question. Aussi, il résulte de nos expériences qu'identifier directement le span-réponse dans le paragraphe donne de meilleurs performances que si l'on identifie d'abord la phrase-réponse. Nous avons comparé les trois modèles suivants: la régression logistique, la forêt aléatoire, les machines à vecteurs de supports (SVM).
À modèle égal, les performances en terme d'exact match de notre régression logistique (48.47%) sont nettement meilleures que celles obtenues par [Rajpurkar et al., 2016] (40.0%). Avec 52.29% d'exact match, le système SVM est le meilleur des modèles étudiés. Après

avoir observé les performances des différents systèmes construits en fonction de chaque type de question, nous avons combiné le modèle SVM avec celui de forêt aléatoire. On obtient alors un système combiné donnant 52.41% d'exact match. Ce dernier système obtient des performances nettement supérieures à celles de [Rajpurkar et al., 2016] et est compétitif avec les premiers modèles neuronaux proposés sur SQuAD comme par exemple le match-LSTM réalisé par [Wang and Jiang, 2016] qui obtient 54.5% d'exact match. Cependant, nous sommes encore bien loin des modèles BERT (sortis en cours de mémoire) ainsi que des derniers modèles proposés (89.89%). Nous pensions au départ pouvoir développer un modèle compétitif avec l'état de l'art mais cela ne s'est pas avéré le cas et cette importante différence de performances avec l'état de l'art témoigne de la difficulté à construire un modèle basé sur les *features* et appuie la pertinence des approches neuronales profondes pour cette tâche.

Pour poursuivre ce travail, il serait intéressant, dans un premier temps, d'ajouter de nouveaux *features* à nos modèles et d'en mesurer les performances. Même si nous avons convergé vers un ensemble décent de 8725 *features*, notre étude reste partielle. On pourrait élargir les *features* d'analyse en dépendance à la question. Ainsi pour l'ensemble des questions du *dataset* on pourrait collecter toutes les relations de dépendances grammaticales entre les mots avant de conserver uniquement les n plus fréquentes. On pourrait aussi s'intéresser à des *features* booléens qui s'armeraient lorsque certains éléments sont présents à la fois dans la question et dans le span. On pourrait alors trouver des relations liant les questions aux spans-réponses.

On pourrait également réaliser une validation croisée (*cross validation*) pour déterminer la taille de nos deux familles de traits d'analyse en dépendance. En effet, ici nous avons, dans un premier temps, collecté les *features* sur environ 90% du train avant de déterminer sur les 10% restant le nombre f de *features* à conserver. Il convient de souligner que f est variable en fonction de la division réalisée. Cependant, on pourrait obtenir plus de stabilité au niveau de f en réalisant une validation croisée.

Bien que nous avons montré dans ce travail qu'il était possible de capturer de l'information simplement, il convient de souligner que nous utilisons de nombreuses ressources entraînées sur d'autres corpus (parsers, NER, etc). Il serait sans doute possible d'améliorer nos modèles en utilisant des ressources entraînées sur le même *dataset* (données en provenance de Wikipédia) mais cela complexifierait l'approche pour un gain probablement mineur.

Dans un second temps, il serait pertinent de fournir à des modèles avec plus de capacité nos *features* dans l'espoir d'en tirer le meilleur profit. Il serait alors possible de juger de la qualité et de la pertinence de nos *features* comparativement aux *features* utilisées par les modèles à l'état de l'art. Bien qu'ici on réduirait la simplicité et donc l'interprétabilité des modèles, on obtiendrait sans doute des résultats plus compétitifs.

Enfin, l'une des raisons pour lesquelles nous souhaitions restreindre la taille de nos modèles était de construire des modèles qui généralisent bien. Il nous faudrait donc maintenant vérifier si les performances obtenues par nos modèles sur d'autres *datasets* sont similaires à celles obtenues sur SQuAD. Aussi, étant donné que la faible taille de nos modèles est l'une de leurs forces, on pourrait évaluer leur robustesse sur des corpus de petite taille comparativement à des modèles tels que BERT qui eux utilisent 110 ou 340 millions de paramètres. En effet, puisqu'on a peu de coefficients à apprendre ceci peut-être un avantage pour des applications spécifiques ou les données manquent. On pourrait alors regarder la quantité de données nécessaire au fine-tuning de BERT pour qu'il obtienne de bonnes performances et voir si il est possible de faire mieux avec moins de données. L'étude de [Le Berre and Langlais, 2020] à récemment montré sur OpenBookQA (un *dataset* de Question-Réponse) que la prise en compte de la question par un modèle BERT influence peu les performances obtenues pour sélectionner la bonne réponse. En plus de confirmer le doute concernant la réelle compréhension linguistique des réseaux neuronaux cette étude nous conforte dans l'idée qu'il faudrait évaluer nos modèles, qui eux tiennent compte des questions sur d'autres jeu de données.

Bibliography

Raju Barskar, Gulfishan Firdose Ahmed, and Nepal Barskar. An approach for extracting exact answers to question answering (qa) system for english sentences. 2012.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017. ISSN 2307-387X.

Gary C. Borchardt. Understanding causal descriptions of physical systems. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, AAAI'92, pages 2–8. AAAI Press, 1992. ISBN 0-262-51063-4. URL `http://dl.acm.org/citation.cfm?id=1867135.1867136`.

Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*, 2019.

David Ferrucci, Eric Brown, Jennifer Chu-Carroll, James Fan, David Gondek, Aditya A Kalyanpur, Adam Lally, J William Murdock, Eric Nyberg, John Prager, et al. Building watson: An overview of the deepqa project. *AI magazine*, 31(3):59–79, 2010.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

Bert F Green Jr, Alice K Wolf, Carol Chomsky, and Kenneth Laughery. Baseball: an automatic question-answerer. In *Papers presented at the May 9-11, 1961, western joint IRE-AIEE-ACM computer conference*, pages 219–224. ACM, 1961.

Karl Moritz Hermann, Tomás Kociský, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. *CoRR*, abs/1506.03340, 2015. URL `http://arxiv.org/abs/1506.03340`.

Robin Jia and Percy Liang. Adversarial examples for evaluating reading comprehension systems. *CoRR*, abs/1707.07328, 2017. URL http://arxiv.org/abs/1707.07328.

Boris Katz. Annotating the world wide web using natural language. In *Computer-Assisted Information Searching on Internet*, RIAO '97, pages 136–155, Paris, France, France, 1997. LE CENTRE DE HAUTES ETUDES INTERNATIONALES D'INFORMATIQUE DOCUMENTAIRE. URL http://dl.acm.org/citation.cfm?id=2856695.2856709.

Boris Katz, Sue Felshin, Deniz Yuret, Ali Ibrahim, Jimmy Lin, Gregory Marton, Alton McFarland, and Baris Temelkuran. Omnibase: Uniform access to heterogeneous data for question answering. pages 230–234, 06 2002. doi: 10.1007/3-540-36271-1_23.

Boris Katz, Gary C Borchardt, and Sue Felshin. Natural language annotations for question answering. In *FLAIRS Conference*, pages 303–306, 2006.

Oleksandr Kolomiyets and Marie-Francine Moens. A survey on question answering technology from an information retrieval perspective". *Information Sciences*, 181(24): 5412 – 5434, 2011. URL http://www.sciencedirect.com/science/article/pii/S0020025511003860.

Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard H. Hovy. RACE: large-scale reading comprehension dataset from examinations. *CoRR*, abs/1704.04683, 2017. URL http://arxiv.org/abs/1704.04683.

G. Le Berre and P. Langlais. Attending knowledge facts with bert-like models in question-answering: Disappointing results and some explanations. *Canadian AI*, 2020.

Mitchell Marcus, Grace Kim, Mary Ann Marcinkiewicz, Robert MacIntyre, Ann Bies, Mark Ferguson, Karen Katz, and Britta Schasberger. The penn treebank: Annotating predicate argument structure. In *Proceedings of the Workshop on Human Language Technology*, HLT '94, pages 114–119, Stroudsburg, PA, USA, 1994. Association for Computational Linguistics. ISBN 1-55860-357-3. doi: 10.3115/1075812.1075835. URL https://doi.org/10.3115/1075812.1075835.

Michael C. McCord, J. William Murdock, and Branimir Boguraev. Deep parsing in watson. *IBM Journal of Research and Development*, 56:3, 2012.

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*, pages 3111–3119, 2013.

Nasrin Mostafazadeh, Michael Roth, Annie Louis, Nathanael Chambers, and James Allen. LSDSem 2017 shared task: The story cloze test. In *Proceedings of the 2nd Workshop on Linking Models of Lexical, Sentential and Discourse-level Semantics*, pages 46–51, Valencia, Spain, April 2017. Association for Computational Linguistics. doi: 10.18653/v1/W17-0906. URL https://www.aclweb.org/anthology/W17-0906.

Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100, 000+ questions for machine comprehension of text. *CoRR*, abs/1606.05250, 2016. URL http://arxiv.org/abs/1606.05250.

Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for squad. *CoRR*, abs/1806.03822, 2018. URL http://arxiv.org/abs/1806.03822.

Matthew Richardson, Christopher J.C. Burges, and Erin Renshaw. MCTest: A challenge dataset for the open-domain machine comprehension of text. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 193–203, Seattle, Washington, USA, October 2013. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/D13-1020.

Tao Shen, Tianyi Zhou, Guodong Long, Jing Jiang, and Chengqi Zhang. Bi-directional block self-attention for fast and memory-efficient sequence modeling. *CoRR*, abs/1804.00857, 2018. URL http://arxiv.org/abs/1804.00857.

Masumi Shirakawa, Takahiro Hara, and Shojiro Nishio. N-gram idf: A global term weighting scheme based on information distance. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15, pages 960–970, Republic and Canton of Geneva, Switzerland, 2015. International World Wide Web Conferences Steering Committee. ISBN 978-1-4503-3469-3. doi: 10.1145/2736277.2741628. URL https://doi.org/10.1145/2736277.2741628.

Simon Suster and Walter Daelemans. Clicr: A dataset of clinical case reports for machine reading comprehension. *CoRR*, abs/1803.09720, 2018. URL http://arxiv.org/abs/1803.09720.

Adam Trischler, Tong Wang, Xingdi Yuan, Justin Harris, Alessandro Sordoni, Philip Bachman, and Kaheer Suleman. Newsqa: A machine comprehension dataset. *CoRR*, abs/1611.09830, 2016. URL `http://arxiv.org/abs/1611.09830`.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. In *BlackboxNLP@EMNLP*, 2018.

Shuohang Wang and Jing Jiang. Machine comprehension using match-lstm and answer pointer. *ArXiv*, abs/1608.07905, 2016.

Ralph Weischedel, Martha Palmer, Mitchell Marcus, Eduard Hovy, Sameer Pradhan, Lance Ramshaw, Nianwen Xue, Ann Taylor, Jeff Kaufman, Michelle Franchini, et al. Ontonotes release 5.0 ldc2013t19. *Linguistic Data Consortium, Philadelphia, PA*, 23, 2013.

Joseph Weizenbaum et al. Eliza—a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9(1):36–45, 1966.

Terry Winograd. Procedures as a representation for data in a computer program for understanding natural language. 1971. *URL http://hci. stanford. edu/˜ winograd/shrdlu/AITR-235. pdf*, 1971.

Stephen Wolfram. Wolfram| alpha. *On the WWW. URL http://www. wolframalpha. com*, 2007.

Stephen Wolfram. Wolfram| alpha. *On the WWW. URL http://www. wolframalpha. com*, 2009.

W.A. Woods. Semantics and quantification in natural language question answering. volume 17 of *Advances in Computers*, pages 1 – 87. Elsevier, 1978. doi: https://doi.org/10.1016/S0065-2458(08)60390-3. URL `http://www.sciencedirect.com/science/article/pii/S0065245808603903`.

William A Woods. Progress in natural language understanding: an application to lunar geology. In *Proceedings of the June 4-8, 1973, national computer conference and exposition*, pages 441–450. ACM, 1973.

Yukun Zhu, Ryan Kiros, Richard S. Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. *CoRR*, abs/1506.06724, 2015. URL

http://arxiv.org/abs/1506.06724.

Appendix A

Analyse en composante

Fig. A.1. Analyse en composante de "In meteorology, precipitation is any product of the condensation of atmospheric water vapor that falls under gravity." produite par l'analyseur de NLTK