Université Paris-Sud

Master 1 Informatique

TER : Modèles neuronaux pour le traitement des langues

Raisonnement automatique question/réponse

Fait par :
Thierry Loesch
Bryce Tichit

Professeur :
Alexandre Allauzen

26 avril 2017

Table des matières

1 Introduction

L'objectif de ce TER est d'utiliser les réseaux de neurones afin de pouvoir traiter automatiquement le langage naturel dans un problème non-trivial sous forme de texte. Il s'agit d'un problème complexe à réaliser par ordinateur en utilisant des moyens classiques, car celui-ci est assimilable à un raisonnement humain. L'utilisation de réseaux de neurones dans ce contexte est une approche proposant un modèle de calcul calqué sur le fonctionnement des neurones du cerveau humain.

Ceux-ci sont performants dans le sens où ils n'ont pas besoin de règles sémantique pour traiter le langage mais sont au contraire capable de dériver leur propre règles à partir de leur architecture.

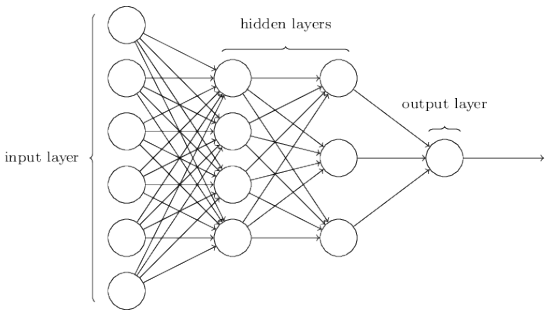


FIGURE 1 – Réseau de neurones

Pour cette étude, nous avons modélisé deux énoncés afin d'inférer un nombre (une notation) ou un mot (réponse à une question). Dans ce cadre, nous avons mis en oeuvre un projet traitant d'un problème de langage naturel consistant à raisonner automatiquement sur un texte, c'est-à-dire chaque phrase qui le compose, afin d'inférer la réponse à une question. Ce projet s'intitule **FaceBook Babi Tasks** [1], il s'agit d'un sujet de recherche porté par l'équipe de recherche de FaceBook et traité dans leur article[1].

L'approche utilisée est celle des Words Embedding, popularisée par Google, qui permet de faire correspondre des mots (et donc des phrases) à des entiers (et donc des vecteurs). Il s'agit d'un ensemble de techniques qui permettent de concentrer l'information d'une phrase dans un vecteur et ainsi le donner en entrée à des réseaux de neurones. Nous utilisons des réseaux récurrents, un modèle qui maintient une représentation d'un historique caché et se met à jour en fonction de ses observations et de cet état caché, permettant de "lire" une phrase et d'en tirer un résultat.

1. https ://research.fb.com/downloads/babi/

Nous avons par la suite adapté nos travaux sur un second sujet : l'inférence automatique de la note d'un étudiant en fonction de sa réponse écrite à une question posée, écrite également. Ce problème est comparable à celui des Babi Tasks dans le sens où il s'agit de raisonner sur des phrases, et d'en déduire des informations. Il est toutefois légèrement plus complexe car le vocabulaire n'est pas maîtrisé contrairement à ce dernier. Nous nous sommes basés sur l'article **Learning to Grade Short Answer Questions using Semantic Similarity Measures and Dependency Graph Alignments**[2].

2 Projet Babi Tasks

2.1 Description des données

Les données sont représentées dans des textes suivant une certaine structure, il y a différentes histoires dans un seul texte et chaque phrase de l'histoire est numérotée. Ainsi lorsque le compteur est à 1 nous commençons une nouvelle histoire. Pour chaque histoire il y a une ou plusieurs questions qui portent sur celle-ci, la réponse, ainsi que les phrases qui justifient cette réponse.

```
1 Mary moved to the bathroom.
2 John went to the hallway.
3 Where is Mary?        bathroom        1
4 Daniel went back to the hallway.
5 Sandra moved to the garden.
6 Where is Daniel?      hallway     4
```

FIGURE 2 – Exemplaire de données

```
ID text
ID text
ID text
ID question[tab]answer[tab]supporting fact IDS.
...
```

FIGURE 3 – Format des données

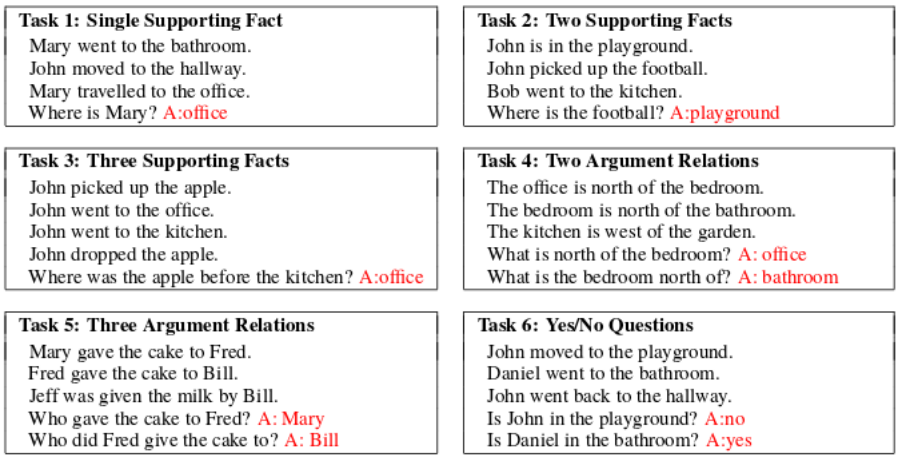
Le projet est partitionné en tâches, celles-ci étant au nombre de 20, avec pour chacune un niveau de difficulté différent. Ces tâches représentent des aspects différents du raisonnement, allant du raisonnement sur une, deux ou trois phrases

à un raisonnement plus complexe sur le lieu ou le temps par exemple. On notera que le vocabulaire utilisé est maîtrisé : les tâches sont générées par un programme limitant le vocabulaire, afin de rester sur des problèmes moins complexes et plus pertinents, en s'affranchissant d'un traitement de langage général.

Les différentes tâches sont pour les 6 premières,

- Tâche 1 : Raisonnement à partir d'une phrase justificative
- Tâche 2 : Raisonnement à partir de deux phrases justificatives
- Tâche 3 : Raisonnement à partir de trois phrases justificatives
- Tâche 4 : Relations de deux arguments
- Tâche 5 : Relation de trois arguments
- Tâche 6 : Question/Réponse binaire

FIGURE 4 – Exemple des tâches et données assimilées



2.2 Méthodologie

Dans cette section nous donnons une vue d'ensemble des différentes méthodes utilisées dans le cadre de la réalisation des objectif de notre projet.

2.2.1 Word Embedding

La méthode du word embedding est une méthode d'apprentissage automatique qui repose sur l'apprentissage d'une représentation de mot. Elle permet au système de se faire une représentation d'un mot et d'une phrase par des vecteurs afin de faciliter l'analyse syntaxique et sémantique de manière automatique, et non pas de manière classique avec des règles décrites par un système. Cette technique est issue du deep learning et a permis de révolutionner le traitement automatique de la langue.

La représentation permet de condenser les particularités d'un texte sous forme d'un vecteur beaucoup plus petit que si on devait stocker tout les mots un à un, diminuant donc le coût d'apprentissage dans le cas d'un réseau de neurones.

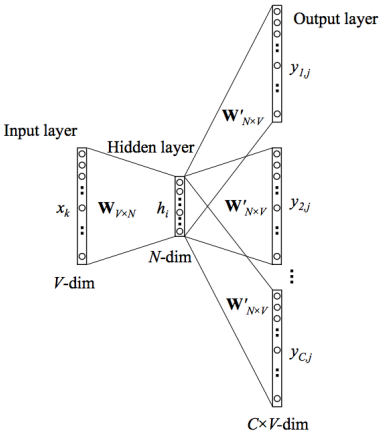


FIGURE 5 – Réseau de neurones skip-gram implémentant les word embedding

2.2.2 Réseaux récurrents

Nous utilisons pour ce projet des réseaux récurrents, il s'agit de réseaux de neurones qui interagissent de manière non-linéaire et contenant un cycle. Le principe étant de maintenir une représentation interne (cachée) de l'historique (les mots qu'on a déjà vus), mise à jour en fonction des observations et de l'état précédent. De ce fait, notre sortie prend en compte tout ce qui a été vu. Cette mémoire interne du réseau nous permet par exemple raisonner sur un tout dans le cas de notre projet.

Plus précisément, nous utilisons des réseaux LSTM (**L**ong **S**hort **T**erm **M**emory) qui sont comme leur nom l'indique des réseaux de mémoire à court terme. Il existe aussi des réseaux récurrents plus performants qui permettent de mémoriser complètement un texte et de raisonner dessus beaucoup plus efficacement : il s'agit des Memory Networks [1] (**MemNN**)

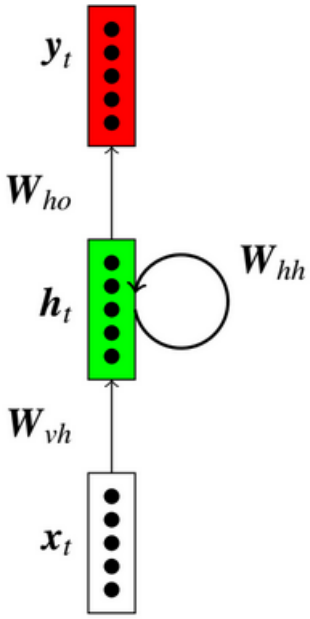


FIGURE 6 – Schéma d'un réseau récurrent

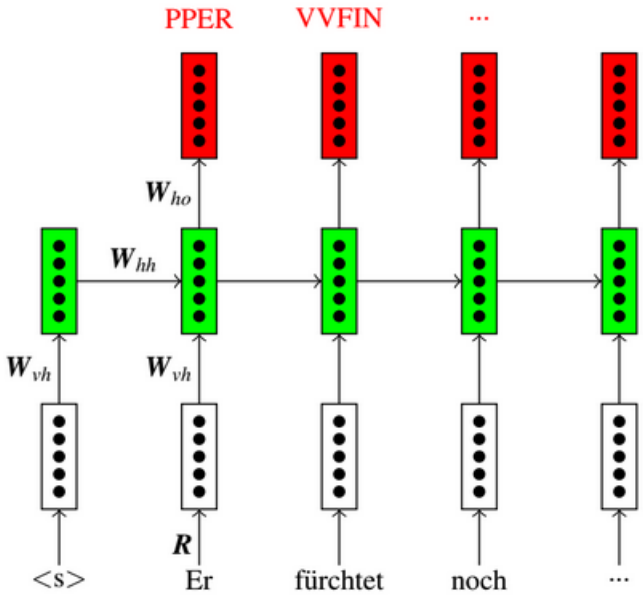


FIGURE 7 – Exemple d'un réseau récurrent déplié

2.2.3 Modèles

Nous créons par la suite notre modèle keras, nous avions choisi donc de faire un modèle pour les stories et pour les questions afin de pouvoir raisonner sur chacun d'eux différemment, puis de les combiner en un modèle grâce à la couche **Merge**, ainsi nous avons un seul modèle mais qui traite différemment les story des questions. On a un LSTM dans le modèle des questions car il faut raisonner d'abord sur les questions seules puis les histoires avec les questions (raisonner sur les histoires seules ne veut pas dire grand chose du point de vue d'un raisonnement). Il s'agit du modèle utilisé dans l'article [1].

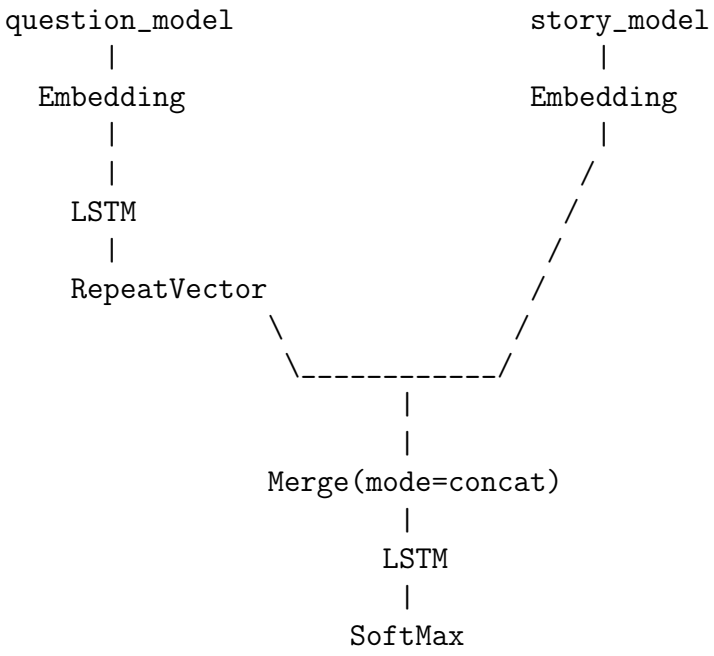


FIGURE 8 – Schéma du modèle utilisé

Nous utilisons donc le modèle décrit dans la figure 8. Une telle structure permet de donner au réseau de neurones la question ainsi que l'histoire, le mode utilisé pour le Merge n'induit que des différences minimes et nous utiliserons le mode concat pour ce projet. Ce mode consiste en la concaténation des vecteurs du premier modèle et du second mis bout à bout. Le résultat final est obtenu par un réseau LSTM final, qui conclut la capacité de raisonnement du modèle, avant de passer par une couche d'activation **SoftMax**.

Nous avions, avant de passer à ce modèle, deux autre modèles dont l'architecture était différente mais contre tout-attente ceux-ci donnaient des résultats beaucoup moins bon. Il s'agissait d'appliquer un LSTM sur les deux modèle puis de faire passer le résultat par une couche Dense afin d'obtenir un vecteur plus

concis puis de récupérer le résultat par une couche d'activation **SoftMax**, on présente ce modèle dans la figure 9. Un autre modèle était de reprendre la même architecture en remplaçant les LSTM par des bi-LSTM, ceux-ci consistent simplement en deux réseaux LSTM qui lisent chacun les phrase dans un sens différent, il s'agit du modèle présenté en figure 10.

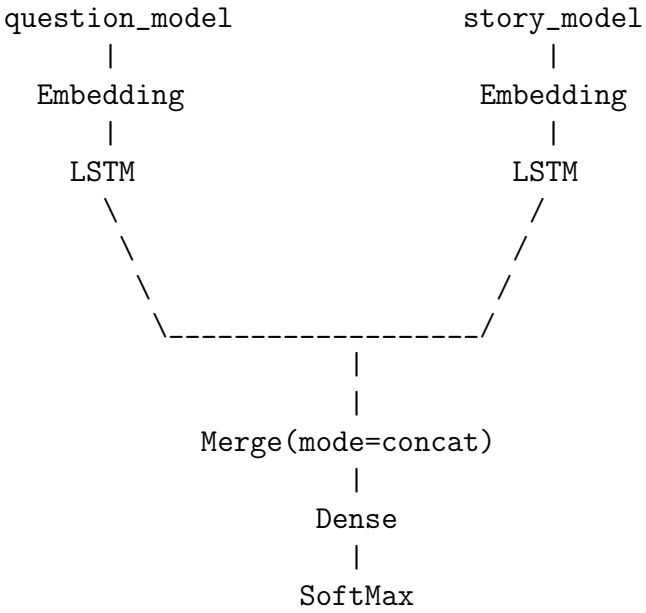


FIGURE 9 – Schéma du modèle 2

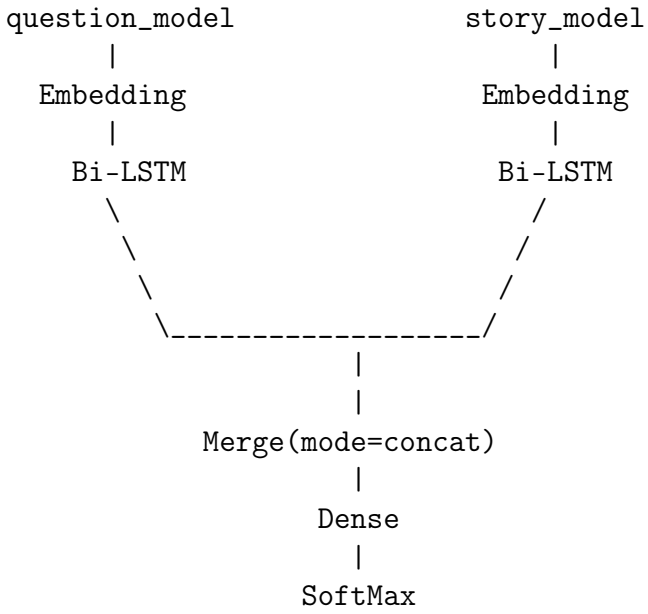


FIGURE 10 – Schéma du modèle 3

2.3 Résultats

Nous présentons dans cette section les résultats de nos différents modèles sur la première tache du projet. Nous avons choisi de prendre cette tache simple afin de départager les modèles et choisir le plus performant pour le reste du projet. Nous commençons par mesurer le taux de précision des modèles avec un ensemble d'apprentissage contenant 1000 exemples ainsi qu'un ensemble de test et un ensemble de validation d'une taille de 500 exemples chacun. Les paramètres utilisés sont, dans un premier temps, une taille de vecteur d'embedding de 50, un nombre d'epoch de 60 et un batch_size de 32. Nous entraînons les modèles avec ces paramètres et la fonction **Model.fit** de Keras et évaluons notre modèle sur l'ensemble de test avec la fonction **Model.evaluate**, les résultats sont visibles sur la table 1.

Par la suite, et pour étayer nos tests, nous réalisons la même expérience mais avec des ensemble de données plus conséquents. Nous aurons 5500 exemples pour notre ensemble d'apprentissage et 2200 exemples pour chacun de nos deux autres ensembles. Nous remarquons que les modèles 2 et 3 n'arrivent pas à progresser malgré l'augmentation de la taille des ensembles. On obtient quasiment les mêmes résultats sur ces deux modèles que ce soit avec 1000 samples ou bien avec 5500 samples. Le modèle 1 lui, reste en haut du podium avec une précision de 66% (gain de 20 points) il obtient des résultats très corrects. Il est très intéressant de voir que ce modèle permet une précision bien supérieure à ses congénères, le fait d'avoir déjà un réseau LSTM sur les questions lui permet probablement de mieux

Modèle	Précision (1000 samples)	Précision (5500 samples)
Modèle 1	48%	66%
Modèle 2	36%	37%
Modèle 3	35%	36%

TABLE 1 – Précision des modèles présentés sur la première tache

raisonner par la suite sur les histoires.

L'observation des courbes d'apprentissages de ces différents modèles confirment nos affirmations (l'évolution des métriques [précision/perte] en fonction du nombre d'epochs). En effet le modèle 2 et le modèle 3 n'arrivent pas à généraliser, et on voit que le taux de perte diminue de plus en plus sur l'ensemble d'apprentissage alors qu'il augmente sur l'ensemble de validation (à partir de l'epoch 7 environ) ce qui est caractéristique d'un sur-apprentissage et du fait que le modèle n'arrive pas à généraliser.

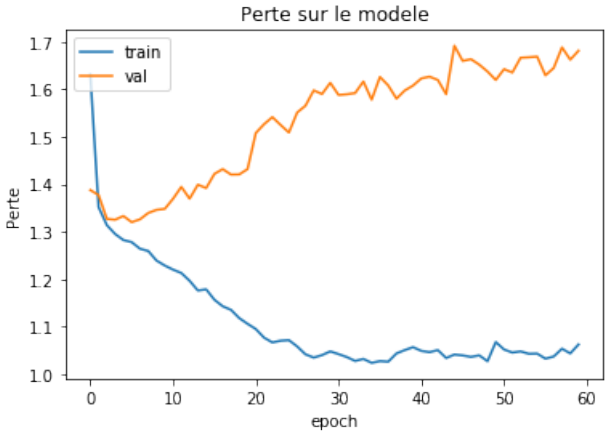


FIGURE 11 – Courbe d'apprentissage de la perte du modèle 2 (fig 9)

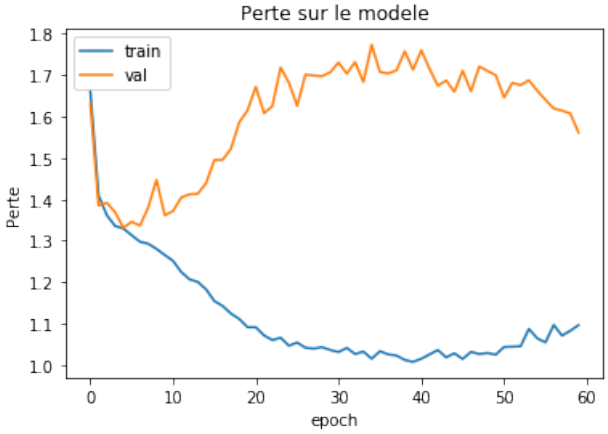


FIGURE 12 – Courbe d'apprentissage de la perte du modèle 3 (fig 10)

En revanche le modèle 1 affiche des performances bien meilleures, on voit que le taux de perte sur l'ensemble d'apprentissage suit de près le taux de perte sur l'ensemble de validation ce qui est caractéristique d'un modèle qui arrive à bien généraliser. Cette courbe nous apprends également que le taux de perte converge déjà à partir de l'epoch 15, on peut donc diminuer largement le nombre d'epochs que l'on utilise et passer de 60 epochs à 20 epochs par exemple.

Il était toutefois intéressant de remarquer que le modèle 1 réalise des performances lorsque celui-ci n'est pas soumis au bruit, en effet nous incluons dans les données l'ensemble de l'histoire y compris les faits n'aidant pas à répondre à la question mais il est possible de ne garder que les faits justificatifs grâce à la structure des données. Nous observons dans ce cas dans performances atteignant jusqu'à 100% sur la première tache. Le bruit présent dans les données est donc un réel problème dans notre cas, une solution apportée par les auteurs de l'article[1] est l'utilisation des Memory Networks qui supportent bien mieux le bruit que les LSTM.

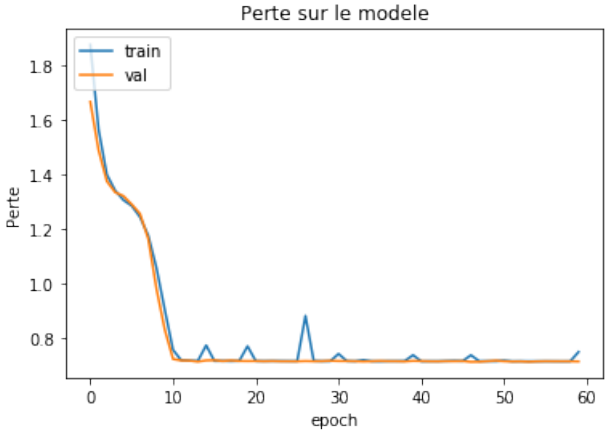


FIGURE 13 – Courbe d'apprentissage de la perte du modèle 1 (fig 8)

Tache	Perte	Précision
1 Single Supporting Fact	1.19	51%
2 Two Supporting Facts	1.781	28%
3 Three Supporting Facts	1.718	19%
4 Two Argument Relations	1.458	35%
5 Three Argument Relations	1.183	39%
6 Yes/No Questions	0.697	48%
7 Counting	0.720	68%
8 Lists/Sets	1.352	48%
9 Simple Negation	0.674	63%
10 Indefinite Knowledge	1.022	47%
11 Basic Coreference	1.082	71%
12 Conjunction	0.748	74%
13 Compound Coreference	0.423	92%
14 Time Reasoning	1.767	20%
15 Basic Deduction	1.384	27%
16 Basic Induction	1.345	41%
17 Positional Reasoning	0.707	50%
18 Size Reasoning	0.723	50%
19 Path Finding	2.499	8%
20 Agent's Motivation	0.130	92%

TABLE 2 – Résultats du modèle 1 sur l'ensemble de test

3 Notation automatique de réponses d'étudiants

3.1 Description des données

Pour ce second sujet, nous disposons à la fois de textes sous forme de phrases mais également de nombres entiers. Le modèle sera entraîné sur une série de questions portant sur les sciences de l'informatique. Chaque question est accompagnée de sa réponse correcte, de toutes les réponses écrites données par des étudiants, et enfin des 2 notes obtenues par les étudiants pour chaque question. On notera que chaque étudiant est donc corrigé deux fois, par deux correcteurs différents, pour chaque réponse donnée. Chaque question et chaque réponse ne contiennent qu'une unique phrase.

Nos données de test seront constituées de la même manière, à l'exception des notes obtenues par les élèves, que nous cherchons à déterminer.

	Sample questions, correct answers, and student answers	**Grades**
Question:	What is the role of a prototype program in problem solving?	
Correct answer:	To simulate the behavior of portions of the desired software product.	
Student answer 1:	A prototype program is used in problem solving to collect data for the problem.	1, 2
Student answer 2:	It simulates the behavior of portions of the desired software product.	5, 5
Student answer 3:	To find problem and errors in a program before it is finalized.	2, 2
Question:	What are the main advantages associated with object-oriented programming?	
Correct answer:	Abstraction and reusability.	
Student answer 1:	They make it easier to reuse and adapt previously written code and they separate complex programs into smaller, easier to understand classes.	5, 4
Student answer 2:	Object oriented programming allows programmers to use an object with classes that can be changed and manipulated while not affecting the entire object at once.	1, 1
Student answer 3:	Reusable components, Extensibility, Maintainability, it reduces large problems into smaller more manageable problems.	4, 4

FIGURE 14 – Exemples de questions posées, de réponses correctes, de réponses d'étudiants et leurs notes

3.2 Méthodologie

Compte tenu de la nature du sujet, notre première approche fut très similaire à celle des bAbI Tasks. La différence principale étant que nous cherchons à inférer une note au lieu d'un mot.

Comme relevé précédemment, notre vocabulaire n'est pas maîtrisé cette fois. Sa taille est initialement de 2200 termes. Puisque que les premières estimations de précision n'étaient pas satisfaisantes, nous avons cherché à réduire ce nombre en regroupant les mots avec leurs synonymes, et améliorer l'efficacité de l'apprentissage. Ainsi, tous les mots appartenant à un même "groupe de synonymes" seront reconnus comme un seul est même terme. Notre regroupement est conçu de sorte

à ce qu'un mot n'apparaisse que dans un seul groupe, l'objectif étant de constater une première amélioration de précision même sur un regroupement grossier.

```
entire -->
[u'total', u'entire', u'full']
```

FIGURE 15 – Exemple de regroupement

Pour réaliser cette tache, nous avons utilisé le python **Natural Language Toolkit** [2] ou **nltk** avec le corpus Wordnet. C'est une librairie fournissant des outils pour manipuler du texte. Dans notre cas, elle sera utilisée pour obtenir tous les synonymes d'un mot.

Cette démarche s'effectue en deux temps dans notre code : la première étape est de récupérer tous les synsets. Un synset est un groupe de synonymes partageant un sens commun. Puis, pour chaque synset nous récupérons tous les lemmes qu'il contient. une fois tous les lemmes de tous les synsets obtenus, nous éliminons les doublons.

3.2.1 Modèle

Le modèle utilisé dans le cadre de ce projet est quelque peu similaire au modèle 1 utilisé dans le projet Babi Tasks (fig 8), excepté que nous avons désormais 3 donnée par exemplaire au lieu de 2 : nous avons la question posée, la réponse de l'étudiant et la réponse du correcteur. Nous avons donc modifié le modèle pour prendre en compte ces 3 donnée mais en s'inspirant du modèle Babi Tasks. Dans ce modèle nous avons un simple embedding pour les question posée, un embedding et un lstm pour les réponses de correcteurs et des étudiants. Le tout récupéré avec une couche **Merge** avant de passer par une couche LSTM, une couche Dense puis par une activation par sigmoid. On entraîne le modèle avec la fonction de coût *Minimum Squared Error* (MSE).

2. http ://www.nltk.org/

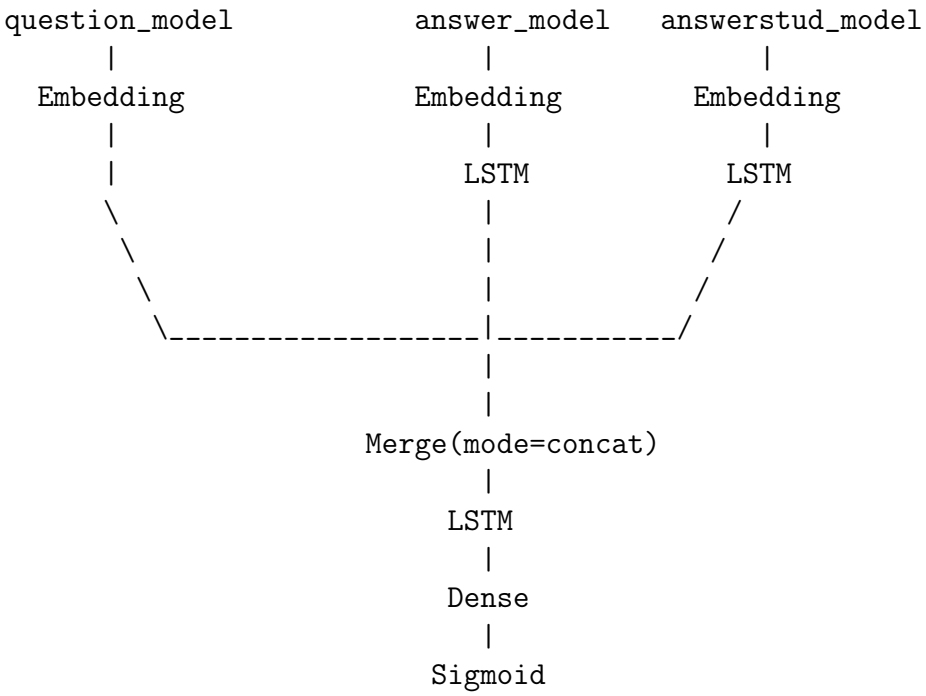


FIGURE 16 – Schéma du modèle utilsé

3.3 Résultats

Après avoir exécuté notre modèle sur les données de test, nous constatons une précision de 50% seulement.

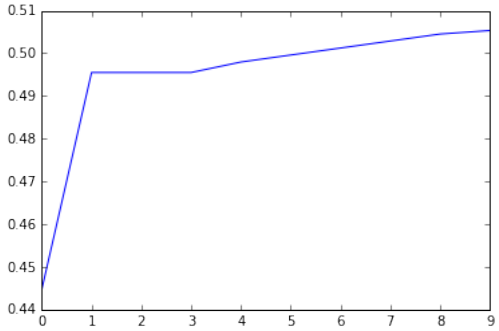


FIGURE 17 – Évolution de la précision en fonction du nombre d'epoch

Nous avons pu réduire la taille de notre vocabulaire effectif de 2200 à 1200 termes avec le regroupement par synonyme. Cependant nous ne constatons qu'une très faible amélioration de notre précision à la suite de ce changement (environ 2%).

Il était prévu à la base de coupler cette méthode avec du stemming, qui consiste à transformer un mot en son radical, pour réduire encore plus la taille du vocabulaire. Mais l'amélioration suite au regroupement étant négligeable, nous ne l'avons pas jugé nécessaire.

3.4 Difficultés

Malgré les différentes approches, nous n'avons pas réussi à obtenir un taux de précision satisfaisant. Notre première méthode pour rassembler les termes entre eux et réduire le vocabulaire fut d'utiliser Word2Vec, un outil comprenant 2 algorithmes et permettant, entre autres, de quantifier la similarité entre des mots. Il s'est rapidement avéré après quelques tentatives que cette solution n'était pas idéale pour les synonymes. Ntlk s'est montré plus approprié.

4 Conclusion

Pour conclure, notre modèle implémenté pour le sujet bAbI Tasks fonctionne de manière satisfaisante. Nos résultats sont très proches de ceux de l'article[1]. Cependant, s'il est adapté pour modéliser un énoncé et inférer un mot, la taille du vocabulaire et les types d'entrées sont des éléments déterminants pour son efficacité.

De ce fait, le modèle est limité à une précision moyenne dans le cas des Notations Automatiques malgré les modifications pour adapter notre système. Nous devons traiter 1200 mots dans le vocabulaire, contre seulement 21 pour la première tache du bAbI Tasks par exemple. Il est également probable que d'autres adaptations soient nécessaires, notamment pour gérer les nouvelles entrées.

Il serait encore possible d'améliorer nos résultats pour l'inférence d'un mot de manière significative en utilisant des Memory Network, et réduire les perturbations dues au bruit. Notre limitation dans ce cas fut matérielle.

Références

[1] Jason Weston, Antoine Bordes, Sumit Chopra, Alexander M. Rush, Bart van Merriënboer, Armand Joulin, Tomas Mikolov
Towards AI-Complete Question Answering : A Set of Prerequisite Toy Tasks

[2] M. Mohler, R. Bunescu, R. Mihalcea
Learning to Grade Short Answer Questions using Semantic Similarity Measures and Dependency Graph Alignments