Sorbonne Université
Sciences et ingénierie
Master Informatique

Recherche d'Information

Antoine Théologien - 21400184

Adam Keddis

Gautier Cai

Parcours :	DAC
Cours :	RITAL
Date :	May 8, 2025

1 Introduction

Ce rapport est consacré à notre projet de recherche d'information au sein de l'UE RITAL. Nous y détaillerons les différentes étapes de la ré implémentation du papier : *On the Benefit of Incorporating External Features in a Neural Architecture for Answer Sentence Selection*, ainsi que les résultats des expérimentations que nous avons pu mener.

2 Présentation du papier

L'article publié lors de la conférence SIGIR 2017, s'intéresse à la tâche de sélection de phrases-réponses dans un système de question-réponse. Les modèles récents de deep learning ont souvent tendance à ignorer les caractéristiques manuelles classiques (external handcrafted features) jugées obsolètes. Les auteurs remettent en question ceci et montrent que ces features, lorsqu'elles sont bien choisies, peuvent toujours être bénéfiques, même dans le cadre de modèles neuronaux modernes.

2.1 Contributions principales

Les auteurs proposent d'intégrer 21 caractéristiques externes dans un modèle de base utilisant un CNN bi-encodé. Ces caractéristiques sont regroupées selon trois axes :

- **Appariement lexical et sémantique**
- **Lisibilité des phrases**
- **Focus de la question**

Ils montrent qu'une telle approche permet d'atteindre des performances supérieures à plusieurs architectures complexes récentes, tout en gardant une architecture relativement simple.

2.2 Détail des Features

Appariement lexical et sémantique :

- `Length` : longueur de la réponse (en nombre de mots).
- `ExactMatch` : si la question est une sous-chaîne de la réponse.
- `Overlap`, `OverlapSyn` : taux de recouvrement lexical ou basé sur les synonymes.
- `LM`, `BM25` : scores issus de modèles de langage probabilistes.
- `ESA`, `TAGME` : mesure de similarité sémantique basée sur Wikipedia.
- `Word2Vec` : similarité cosinus entre vecteurs moyens.

Lisibilité : Mesures classiques pour évaluer la complexité textuelle :

- `CPW`, `SPW`, `WPS` : caractères, syllabes, mots par phrase.
- `CWPS`, `CWR` : nombre et proportion de mots complexes.
- `LWPS`, `LWR` : mots longs par phrase et proportion.
- `DaleChall` : indice de lisibilité de Dale-Chall.

Focus (MatchedNGram) : Cette caractéristique mesure la similarité entre les k premiers mots de la question et des n-grammes consécutifs dans la réponse. On prend le maximum de similarité cosinus :

$$\text{MatchedNGram}(Q, A) = \max_{l} \cos\left(\sum_{i=1}^{k} \vec{q}_i, \sum_{j=l}^{l+n-1} \vec{a}_j\right)$$

Les meilleures performances sont obtenues avec $k \in \{2, 3\}$ et $n \in \{2, 3\}$.

2.3 Modèle utilisé

Le modèle de base est un CNN bi-encodé suivant Severyn et Moschitti (2015), avec :

- Deux encodeurs convolutifs parallèles (un pour la question, un pour la réponse).
- Word embeddings préentraînés (Wikipedia/AQUAINT en 50d, Google News en 300d).
- Max pooling, activation `tanh`, et couches entièrement connectées.
- Les 21 features externes sont concaténées à la représentation vectorielle concaténée question-réponse.
- Optimisation avec AdaDelta ($\rho = 0.95$), early stopping après 5–10 époques.

Une variante intègre également un module d'attention inspiré d'ABCNN-1, calculant l'interaction entre les cartes de caractéristiques question/réponse.

2.4 Expérimentations et données

Deux benchmarks sont utilisés :

- **TREC QA** : 1 229 questions pour l'entraînement, 82 pour la validation, 100 pour le test.
- **WikiQA** : 873 (train), 126 (val), 243 (test). Ce corpus est plus difficile car moins basé sur la similarité lexicale.

Les résultats montrent que :

- L'ajout des 21 features externes améliore nettement les performances du modèle CNN.
- L'approche reste simple à entraîner, rapide, et peu coûteuse en ressources.

3 Ré-Implémentation

Après la lecture et la compréhension du papier, nous avons pu commencer l'implémentation et la reproduction des expérimentations de celui-ci. Pour cela, nous avons pu bénéficier du dépôt *Github* du projet, créé par l'auteur original, facilitant nos futures expérimentations.

3.1 Structure du code

Le code de ce papier se présente en deux parties distinctes : la partie ajoutée par le papier, portant sur l'implémentation du réseau de neurones convolutionnel avec les différentes features, ainsi que les différents makefiles permettant de créer les jeux de données, ceux-ci étant assez conséquents. La seconde partie est

contenue dans un dossier *deep-qa*, et provient d'un second papier cité dans l'article original, qui implémente l'apprentissage global d'un réseau de neurones pour les problèmes de type QA. Ainsi, pour effectuer les différentes implémentations, nous n'avions en théorie qu'à remplacer le réseau de neurones du second papier par celui du papier que nous voulions réimplémenter. En pratique, les choses étaient bien différentes, comme nous allons le voir dans le section suivante.

3.2 Contraintes et ajouts

Le premier problème dans dans cette implémentation était la version python dans laquelle celle-ci avait été créée. En effet le papier sur lequel se basait celui que l'on étudie ayant été publié en 2015, le code avait été écrit en Python 2, rendant incompatible l'exécution de celui-ci avec nos environnements habituels, en Python 3. Nous avons donc créé un environnement virtuel dans la bonne version de Python, afin de pouvoir effectuer nos différentes expérimentations.

Ainsi, après avoir téléchargé les nombreuses librairies nécessaires à l'exécution du programme, nous pouvions lancer l'exécution. Malheureusement, nous avions également dû faire face à un problème de taille : les embeddings utilisés dans le calcul des features n'étaient plus accessibles dans le lien fourni sur le GitHub. Nous avons donc trouvé une version très similaire, qui devrait en principe correspondre aux embeddings originaux, mais pouvant peut-être affecter les résultats.

Une fois tous les fichiers et embeddings nécessaires téléchargés, nous n'avions plus qu'à exécuter le code.

4 Résultats

System	Attn?	Drop?	TREC QA		WikiQA	
			MAP	MRR	MAP	MRR
Runs (AQUAINT/Wikipedia)						
CNN	✗	✗	75.25	80.9	26.37	26.84
Combined Model	✗	✗	77.44	82.19	23.46	23.86
Combined Model	✗	✓	77.44	82.19	23.46	23.86
CNN	✓	✗	75.37	80.8	25.09	25.49
Combined Model	✓	✗	76.59	81.45	25.23	25.66
Combined Model	✓	✓	76.59	81.45	25.23	25.66
Runs (Google News)						
CNN	✗	✗	75.18	80.57	25.87	26.24
Combined Model	✗	✗	76.41	82.90	24.82	25.22
Combined Model	✗	✓	76.41	82.90	24.82	25.22
CNN	✓	✗	74.15	80.6	25.43	25.95
Combined Model	✓	✗	75.16	79.68	25.34	25.87
Combined Model	✓	✓	75.16	79.68	25.34	25.87

Table 1: Résultats expérimentaux sur TREC QA et WikiQA (MAP, MRR). Les valeurs entre parenthèses indiquent le gain/perte relatif par rapport au modèle CNN.