Université Paris Descartes

U.F.R DES SCIENCES MÉDICALES

Master 2 Santé Publique

Parcours Informatique Biomédicale

Soutenance UE 1

Classification par deep-learning de données structurées et textuelles d'articles issus de PubMed

Nicolas GIRAUD

Imed KERAGHEL

1 Introduction

L'objectif de ce travail était d'élaborer plusieurs modèles de deep-learning utilisant les titres et abstracts d'un dataset fourni d'articles issus de PubMed, afin de prédire par apprentissage supervisé la catégorie SIGAPS associée à l'article. Ce travail a été réalisé en différentes étapes, que nous détaillerons une à une : analyse du dataset et pré-traitement des données, construction du modèle de deep-learning puis optimisation des différents modèles créés selon leurs performances prédictives sur l'échantillon de test non utilisé pour entrainer le modèle.

2 Matériels et méthodes

L'ensemble du codage a été réalisé en langage python, sur les IDE PyCharm et Jupyter Notebook. Les bibliothèques suivantes ont été utilisées : json pour l'extraction du dataset ; tensorflow, keras et sklearn pour l'élaboration des modèles de deep learning ; matplotlib pour la création de figures ; pandas et numpy pour le travail sur les matrices et l'analyse de données ; nltk pour la structuration des données textuelles (segmentation, racinisation) ; plaidml.keras pour l'utilisation des ressources du GPU ; nlpaug pour générer des données textuelles additionnelles.

Tensorflow Projector (projector.tensorflow.org) a été utilisé pour la visualisation interactive des word embeddings.

Enfin, les vecteurs de mots pré-entrainés issus de GloVe (Global Vectors for Word Representation), développés par l'Université de Stanford, ont été utilisés.

3 Pré-traitement des données

Le dataset fourni pour ce projet contient 10000 articles issus de PubMed, contenant pour chacun différentes informations dont le titre, l'abstract, la catégorie SIGAPS, l'année de publication, les mots-clés en rapport, ... Pour ce travail, nous nous intéresserons uniquement aux titres et abstracts de celui-ci, utilisés pour tenter de prédire la catégorie SIGAPS de l'article. Arbitrairement,

nous avons d'abord divisé le dataset en 8000 articles utilisés pour entrainer l'algorithme (dont 20% comme cohorte de validation) et les 2000 restants uniquement utilisés pour le tester. La première étape consiste à modifier les données d'entrée (titres et abstracts des articles) afin qu'elles soient utilisables par le réseau de neurones :

- transformation de l'ensemble du texte en lettres minuscules
- segmentation (tokenisation) des phrases en mots élémentaires (fonctions tokenize de nltk, Tokenizer de keras et fonctions créées)
- retrait des « stopwords » (dédiés à PubMed) et des caractères spéciaux (filtre du Tokenizer de keras)
- racinisation (stemming) des mots (fonction PorterStemmer de nltk.stem.porter)
- transformation du texte en vecteurs numériques et padding (mise sous forme de vecteurs de même longueur) Ces différentes étapes sont réalisées à la fois sur la cohorte d'entrainement et celle de test.

La variable prédite (catégorie SIGAPS) nécessite également la création d'un dictionnaire reliant la catégorie à un identificateur afin de les classer, avec ajout d'une catégorie « Unknown » pour les articles de l'échantillon de test dont la catégorie n'existe pas dans l'échantillon d'entrainement. Par exemple, si 96 catégories existent dans l'échantillon d'entrainement (97 avec la catégorie « Unknown »), cela conditionne l'output du réseau (prédiction catégorielle en softmax) à 97 dimensions.

Le modèle de deep-learning peut se baser soit sur la création d'un word embedding personnalisé, soit utiliser un word embedding pré-entrainé (GloVe par exemple). Ce word embedding est une représentation vectorielle des mots : des mots proches dans l'espace ont des similarités entre eux (hypothèse distributionnelle). Ils peuvent être visualisés sur l'interface Projector de Tensorflow (exemple dans la Figure 1 d'un embedding développé par notre algorithme).

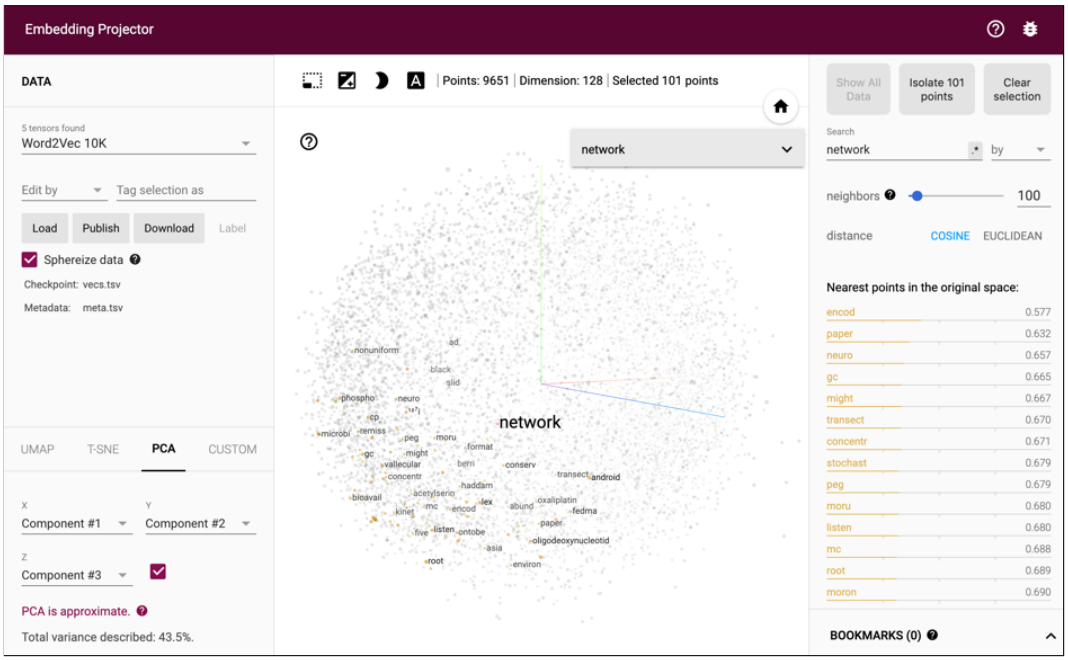

FIGURE 1 – Exemple de visualisation sur Tensorflow Projector d'un word embedding créé

4 Elaboration des modèles de deep-learning

Après le pré-traitement des données, nous avons débuté la construction de l'algorithme de deep-learning (réseau neuronal) à l'aide des bibliothèques tensorflow et keras. Plusieurs architectures sont classiquement utilisées en Natural Language Processing (NLP) : les réseaux neuronaux à convolution (CNN) et les réseaux Long Short Term Memory (LSTM). Ces deux architectures ont été réalisées, en testant dans un premier temps pour un réseau neuronal simple à convolution l'impact de la variation de l'étape de pré-traitement des données sur les résultats finaux, en cherchant à optimiser l'exactitude de prédiction du modèle sur l'échantillon de test. Ces résultats sont résumés dans la Table 1 (moyenne de la « test_accuracy » sur 5 itérations du réseau, 5 epochs pour limiter l'overfitting) :

Pré-traitement	T	T + RS	T + RS + St	T + RS + St + Gl	T + RS + Gl
Test-accuracy	32,86% [32%-33,5%]	33,69% [32,8%-34,45%]	33,81% [33%-34,55%]	31,51% [30,2%-31,75%]	34,09% [33,15%-34,6%]

(T=tokenisation ; RS=retrait des stopwords ; St=stemming ; Gl=Embedding GloVe)

Table 1. Impact du pré-traitement des articles sur les résultats de prédiction

On observe une amélioration des performances avec l'ajout (après tokenisation) du retrait des stopwords dédiés à PubMed [1] et du stemming des mots avec conservation d'un embedding personnalisé. Avec l'utilisation de l'embedding pré-entrainé du GloVe et stemming des mots, les performances semblent légèrement diminuer, notamment car après stemming seulement 53,6% des mots de notre échantillon d'entrainement étaient couverts par le GloVe (contre 83,5% sans stemming). Les meilleures performances étaient retrouvées avec la tokenisation, retrait des stopwords et utilisation de l'embedding du GloVe, sans stemming.

Ensuite, après construction des réseaux, nous avons cherché à les optimiser en faisant varier la complexité des modèles (nombre de couches de neurones, nombre de neurones par couche), ainsi que leurs hyperparamètres (taille des batchs, nombre d'epochs), nombre de mots du texte utilisés en entrée, utilisation et importance d'un oubli itératif partiel des poids des neurones en cours d'entrainement (dropout), variation de la dimension de l'embedding des mots, LSTM uni ou bi-directionnel... Afin d'optimiser le réseau en fonction de ces différents paramètres, nous avons développé un code itératif traçant en fonction du paramètre choisi la courbe d'évolution de la véracité de prédiction du modèle sur l'échantillon test (exemple dans la Figure 2).

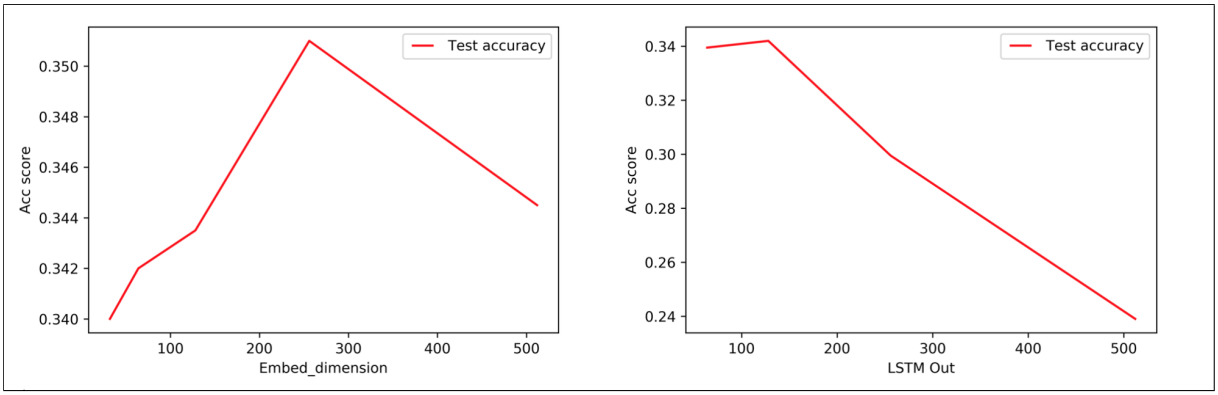

FIGURE 2 – Optimisation de la test_accuracy, ici en fonction de la dimension d'embedding et de la taille de batch en LSTM

Après optimisation des réseaux, l'utilisation conjointe des titres et des abstracts comme données d'entrée du réseau obtient de meilleurs résultats que ceux-ci pris séparément pour un réseau à convolution : exactitude de prédiction sur échantillon de test de 37,0% avec les titres seuls, 39,4% avec les abstracts seuls et 40,0% avec les 2 ; alors qu'en LSTM l'utilisation des seuls abstracts fait

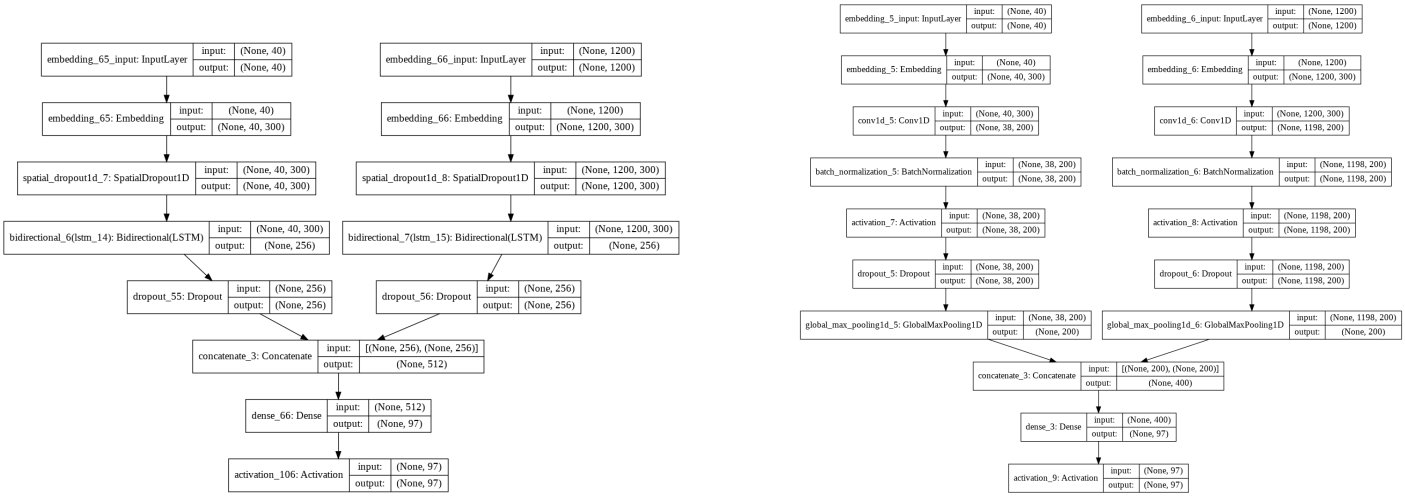

FIGURE 3 – Architecture des réseaux optimisés (LSTM à gauche, CNN à droite)

très légèrement mieux (43,2%) que les titres seuls (40,2%) et les deux associés (42,8%).

Par ailleurs, après ces différentes étapes d'optimisations, l'architecture de réseau à convolution semble obtenir de moins bons résultats qu'en LSTM, ceux-ci restant néanmoins globalement décevants, inférieurs à 45% de prédictions exactes sur l'échantillon de test (Figure 3). A noter que nous avons tenté de développer un modèle mixte associant une branche en CNN et une branche en LSTM, avec des résultats similaires (aux alentours de 40,4%).

Avant augmentation			**Avant augmentation**		
CNN TITRE	**val_acc**	**test_acc**	**LSTM TITRE**	**val_acc**	**test_acc**
	36,69	37		40,19	40.6
	37,69	36,7		42,75	39,79
	38,44	38		42,6	40,7
	37,56	37,1		41,06	40,15
	37,56	36,3		41,16	40,05
Moyenne	37,588	37,02	**Moyenne**	41,552	40,1725
CNN ABSTRACT	**val_acc**	**test_acc**	**LSTM ABSTRACT**	**val_acc**	**test_acc**
	39,37	39,3		43,95	43,4
	38,37	39,35		43,88	43,1
	38,44	40,7		44,81	43,4
	39,99	38,9		45,88	43,2
	38,8	38,5		45,12	42,8
Moyenne	38,994	39,35	**Moyenne**	44,728	43,18
CNN 2 INPUTS	**val_acc**	**test_acc**	**LSTM 2 inputs**	**val_acc**	**test_acc**
	39,87	40,45		45	42,5
	42	41,8		44,42	42,8
	41,38	39,65		45,37	42,95
	40,75	38		42,69	42,8
				42,81	42,9
Moyenne	41	39,975	**Moyenne**	44,058	42,79

Table 2. Résultats de prédiction après optimisation des réseaux neuronaux

Une des explications pouvant expliquer ces faibles performances est la mauvaise répartition des catégories à prédire avec un nombre conséquent de catégories très peu représentées (6 catégories comportent un seul article et 41 moins de 20 articles). Une solution à ce problème est d'augmenter

artificiellement les données : on parle de « data augmentation ». Plusieurs méthodes existent : remplacement de certains mots par des synonymes, suppression de mots au hasard, changement de position des mots, ...

Grâce à cette technique, nous avons généré artificiellement des exemples d'articles (uniquement pour les catégories comportant moins de 200 articles), appris au modèle. Au total, le nombre d'articles final était de 23648.

Après optimisation, cela nous a permis d'obtenir des performances considérablement meilleures, le modèle final atteignant une exactitude de prédiction sur l'échantillon de test avoisinant les 80,02% en utilisant titres et abstracts des articles sans embedding préentrainé (Table 3), avec les mêmes architectures que précédemment.

Après augmentation		
CNN TITRE SANS EMBEDDING	val_acc	test_acc
	72,94	72,23
	73,2	72,09
	72,73	71,99
	73,02	71,86
	72,28	72,02
Moyenne	72,834	72,038
CNN TITRE AVEC EMBEDDING	val_acc	test_acc
	72,99	72,59
	72,78	71,65
	75,73	73,1
	72,08	72
	72,33	71,09
Moyenne	73,182	72,086
CNN ABSTRACT SANS EMBEDDING	val_acc	test_acc
	80,5	79,73
	79,57	78,77
	80,1	79,89
	79,99	79,77
	80,58	79,66
Moyenne	80,148	79,564
CNN ABSTRACT AVEC EMBEDDING	val_acc	test_acc
	78,73	76,8
	80,36	78,52
	79,92	78,87
	79,97	78,9
	79,99	78,2
Moyenne	79,794	78,258
CNN 2 INPUTS SANS EMBEDDING	val_acc	test_acc
	80,52	79,64
	80,68	80,9
	80,73	80,24
	80,36	79,74
	80,87	79,56
Moyenne	80,632	80,016
CNN 2 INPUTS AVEC EMBEDDING	val_acc	test_acc
	78,54	80,59
	78,78	79,94
	79,81	79,957
	79,18	77,97
	78,86	76,93
Moyenne	79,034	79,0774

Après augmentation		
LSTM TITRE SANS EMBEDDING	val_acc	test_acc
	69,24	68,37
	68,74	68,16
	68,95	68,65
	69,77	69,22
	69,13	68,84
Moyenne	69,166	68,648
LSTM TITRE AVEC EMBEDDING	val_acc	test_acc
	73,86	74,33
	74,13	74,77
	73,18	74,5
	73,31	75,32
	72,41	74,09
Moyenne	73,378	74,602
LSTM ABSTRACT SANS EMBEDDING	val_acc	test_acc
	70.61	69.74
	71.70	70.07
	71,88	72.38
	72,35	71,24
	70,01	68,03
Moyenne	71,4133333	69,635
LSTM ABTRACT AVEC EMBEDDING	val_acc	test_acc
	78,25	79,89
	77,8	80,01
	78,33	78,67
	79,6	77,92
Moyenne	78,495	79,1225
LSTM 2 INPUTS SANS EMBEDDING	val_acc	test_acc
	73,34	71,88
	72,04	70,71
	72,12	70,96
	71,88	70,36
Moyenne	72,345	70,9775
LSTM 2 INPUTS AVEC EMBEDDING	val_acc	test_acc
	79,76	79,91
	78,99	77,48
	78,39	77,86
	78,86	78,93
Moyenne	79	78,545

Table 3. Résultats de prédiction après optimisation des réseaux neuronaux et « data augmentation »

5 Conclusion

Le traitement automatique du langage vise à construire des algorithmes afin d'analyser automatiquement et de représenter le langage humain. De tels systèmes sont utilisés avec des applications diverses (moteurs de recherche, assistants vocaux, ...).

Durant ce travail, nous avons appris à créer et optimiser des réseaux neuronaux ayant pour dessein une prédiction catégorielle simple à partir de données textuelles structurées. Cela nous a notamment permis de mettre en exergue l'importance prépondérante de l'exploration préalable et du pré-traitement des données, influant beaucoup sur les résultats finaux de prédiction de l'algorithme. Cela nous a également permis d'explorer les possibilités d'amélioration des performances par « data augmentation », permettant de créer artificiellement de nouvelles sources d'apprentissage au réseau de neurones, améliorant ainsi drastiquement ses capacités prédictives.

D'autres méthodes de machine-learning existent (forêts aléatoires, machines à vecteurs de support...), visant à faire mieux qu'une régression logistique simple. A titre d'exemple, cette dernière, grâce aux abstracts fournis, obtient environ 45,1% d'exactitude de prédiction.

Références :

[1]www.ncbi.nlm.nih.gov/CBBresearch/Wilbur/IRET/DATASET/ ?fbclid=AR0pKTYlFKL65EMn8KX6o8uk7vMOY1kX6UuBfeGve3o9dyHeFU53zmEg