
Projet NLP

GONCALVES Céline et SARMINI DET SATOUF Arthur
- ING3 IA G1 - 15 janvier 2023

Sous la direction de

Anca-Alexandra Munteanu,
Merit Kuldkepp,
Nicolas Merli



Table des matières

Remerciements	3
I. Introduction	4
II. Description du jeu de données	5
Lecture des données	5
Étude des données	5
Baseline MFS i.e. Most Frequent Sense	6
III. Analyse du réseau de neurones pour la WSD	7
Tokenization de type BERT	7
Le réseau de neurones MLP	8
Le modèle de classification WSDClassifier	8
IV. Analyse des entraînements	9
Fine-tuning sur la tâche de WSD	9
Configurations des entraînements	9
Résultats de l'entraînement	10
V. Résultats et Conclusion	11
Évaluation	11
Conclusion	11

Remerciements

Je remercie toute l'équipe du module NLP pour la mise en place de celui-ci dans de bonnes conditions. Je remercie particulièrement Madame Anca-Alexandra Munteanu, Madame Merit Kuldkepp et Monsieur Nicolas Merli pour la qualité des cours. Par ailleurs, les ressources pédagogiques étaient toujours présentes au bon moment et bien organisées.

I. Introduction

La désambiguïsation lexicale consiste à déterminer le sens d'un mot dans une phrase lorsque celui-ci peut avoir plusieurs sens possibles. Dans le domaine du traitement du langage naturel, cette tâche se nomme WSD i.e. Word Sense Disambiguation. Il s'agit d'associer à un mot son sens le plus approprié parmi un inventaire de sens prédéfini (*Exemple : le mot « accepter » peut avoir les sens suivant, « Respond_to_proposal », « FR_Taking_sides », « FR_Agree_or_refuse_to_act », « Ratification », « Other_sense » ou bien « FR_Awareness-Certainty-Opinion »*).

Dans cette étude, les données ASFALDA (i.e. analyse syntaxique en français pour l'analyse automatique des dépendances) French FrameNet seront utilisées (16 000 annotations pour environ 100 frames distincts). Le projet ASFALDA s'est terminé en juin 2016 et a été financé par l'Agence Nationale de la Recherche (ARN). Celui-ci vise à fournir à la fois un corpus français avec des annotations sémantiques et des outils automatiques d'analyse sémantique superficielle.

II. Description du jeu de données

Lecture des données

La fonction « `load_asfalda_data` » charge et traite des données à partir de deux fichiers. Un fichier contenant des données annotées, et un autre qui spécifie quelles données doivent être incluses dans les ensembles d'entraînement, de développement et de test.

La fonction a un argument optionnel appelé « `val_proportion` », qui est défini par défaut à « `None` ». Si « `val_proportion` » est défini à une valeur comprise entre 0 et 1, la fonction divisera les données d'entraînement en un ensemble d'entraînement et un ensemble de validation, avec l'ensemble de validation représentant la proportion spécifiée dans la fonction « `split_list` ».

La fonction renvoie trois dictionnaires répartis en train/dev/test/val : « `sentences` », « `tg_wrks` » et « `labels` ».

Étude des données

Par la suite, la fonction « `frequence` » retourne un premier dictionnaire dont les clés sont des lemmes et les valeurs sont des dictionnaires qui associent à chaque label sa fréquence d'apparition pour le lemme correspondant. Le second associe à chaque lemme le label qui apparaît le plus fréquemment pour ce lemme.

```
defaultdict(<function __main__.frequence.<locals>.<lambda>()>,
            {'créer': defaultdict(int,
                                   {'Other_sense': 71,
                                    'FR_Cause_to_start-Launch_process': 6})},
```

Fig.1 : exemple d'un calcul de fréquence pour le lemme « créer »

Parmi les différents labels qui peuvent être affectés à un lemme, seul, le plus fréquent est récupéré et utilisé. Ainsi, *dict_most_frequent*, retourne pour chaque lemme son label le plus fréquent. Soit, pour l'exemple précédent : « `Other_sense` ».

À partir du dictionnaire *label2i* nous pouvons obtenir l'id des labels comme par exemple celui de « Other_sense » qui est 76.

Puis, l'étude des lemmes, des sens et de l'association lemme-sens inconnus dans dev est réalisée.

Exemple d'association inconnue dans dev :

	Train	Dev
Lemme	Délibérer	Délibérer
Sens	Other_sense	FR_Chating_Discussion

Le sens « Other_sense » pour le mot délibérer est donc inconnu dans « dev ». 3,9% des lemmes existent dans le jeu de données « dev » mais n'existent pas dans les jeux de donnée « train » et « val ».

Baseline MFS i.e. Most Frequent Sense

Finalement, un modèle naïf (baseline) est implémenté et permettra d'être un point de comparaison pour d'autres modèles. Ce modèle utilise comme base, le dictionnaire précédemment créé *dict_most_frequent*. La précision obtenue sur les données « train » + « val » est de 81.396% et sur « dev » est de 83.594%.

III. Analyse du réseau de neurones pour la WSD

Tokenization de type BERT

Avant la construction de l'architecture du réseau de neurones, une étape de tokenization des données est réalisée. Ainsi, le module « Transformers » est importé. Il s'agit d'une bibliothèque de modèles pré-entraînés proposée par Hugging Face.

Dans ce projet, le modèle de langue pré-entraîné « FlauBERT » sera utilisé. Celui-ci a été développé en France par l'INRIA i.e. Institut national de recherche en sciences et technologies du numérique et le CNRS i.e. centre national de la recherche scientifique. La version "base" de « FlauBERT » se réfère à sa taille, tandis que la version "cased" indique qu'elle est sensible à la casse.

Après l'import du module, le code utilise la fonction « AutoTokenizer » pour charger le tokenizer associé au modèle de langue « FlauBERT ». Le tokenizer est utilisé pour pré traiter les données textuelles avant de les utiliser. Le tokenizer divise le texte en « tokens », leur attribuant un identifiant unique. Ensuite, il charge la configuration du modèle « FlauBERT » en utilisant la fonction « AutoConfig ». La configuration du modèle contient les hyperparamètres du modèle et les spécifications de sa structure, comme le nombre de couches et le nombre de neurones par couche.

L'encodage permettra d'obtenir en sortie une liste de phrases encodées ainsi qu'une liste contenant le rang du premier token du mot cible dans les phrases encodées.

```
[ '<s>', 'Con', 'séqu', 'emment</w>', 'leurs</w>', 'codes</w>', 'compr', 'endraient</w>',  
'des</w>', 'erreurs</w>', '.,</w>', '</s>', '<pad>', '<pad>', '<pad>', '<pad>', '<pad>',  
'<pad>', '<pad>', '<pad>']  
Len = 20 target token rank = 6 tid_seq = [0, 1198, 17358, 13299, 121, 5677, 18719, 16724, 23,  
3842, 16, 1, 2, 2, 2, 2, 2, 2, 2, 2]
```

Fig.2 : exemple d'encodage

La classe « WSDData » permettra l'encodage des données ASFALDA et leur manipulation pour la tâche de WSD. En effet, pour chaque partie du corpus, un objet

WSDData est créé en utilisant les données de phrases, les rangs des cibles, les lemmes cible, les labels et l'encodage FlauBERT. Le résultat est stocké dans un dictionnaire `wsd_data` avec pour clé la partie du corpus (train, test, dev, val).

Le réseau de neurones MLP

Un réseau de neurones multi-couches est implémenté. La classe « MLP » hérite de la classe `nn.Module` de PyTorch puis définit deux couches de neurones (`nn.Linear`) et une fonction d'activation (`nn.Tanh`). La méthode `forward` de la classe « MLP » renvoie la sortie du réseau de neurones MLP en effectuant une propagation avant à travers les couches de neurones.

Le modèle de classification WSDClassifier

La classe « WSDClassifier » représente un modèle de classification pour la tâche de WSD. Le modèle utilise un réseau de type BERT et peut utiliser, selon la configuration initiale, le réseau de neurones de la classe MLP ou une couche de neurones (`nn.Linear`). Ensuite, une fonction d'activation `LogSoftmax` sera appliquée.

En utilisant l'option « `freeze_bert` », on peut choisir de geler les paramètres du modèle de BERT (c'est-à-dire de les rendre non-entraînables). Si l'option « `add_lemmas` » est activée, on instancie un objet de la classe `nn.Embedding` qui permet de représenter les lemmes de chaque mot sous forme d'un vecteur dense.

La méthode `forward` permet de définir le calcul effectué par le modèle, en renvoyant un tenseur de scores de prédiction pour chaque classe (appliquant une fonction `log-softmax` sur les scores obtenus).

IV. Analyse des entraînements

Fine-tuning sur la tâche de WSD

Le modèle est entraîné à chaque époque sur les données d'entraînement et testé sur les données de validation. Quelques variables et hyperparamètres sont définis, tels que la taille du lot, le taux d'apprentissage, le nombre d'époques d'entraînement et les paramètres d'arrêt précoce. Le modèle de classification est créé en utilisant la classe `WSDClassifier`.

La fonction de perte est `nn.NLLLoss` et utilise selon l'entraînement des poids de classe équilibrés afin d'atténuer l'impact des classes déséquilibrée sur la perte. L'optimiseur Adam est utilisé pour mettre à jour les paramètres du modèle pendant l'entraînement. Les gradients sont calculés en utilisant la fonction `backward()`.

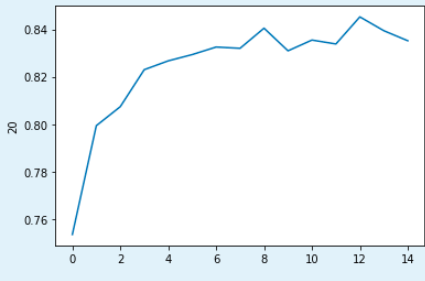
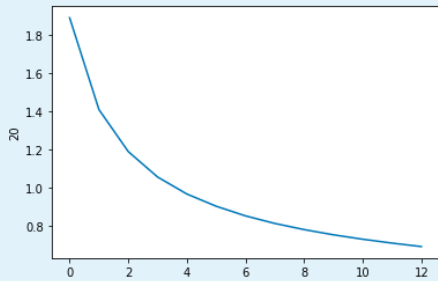
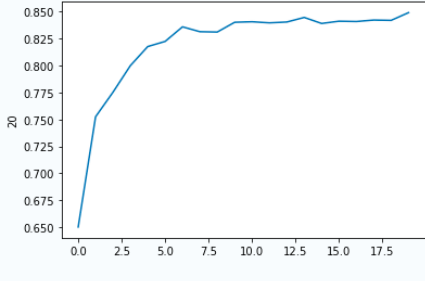
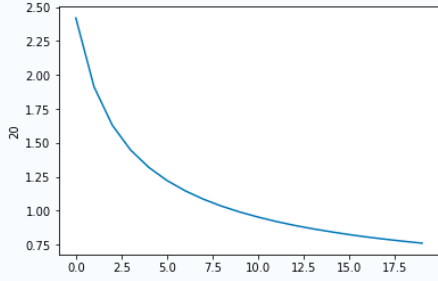
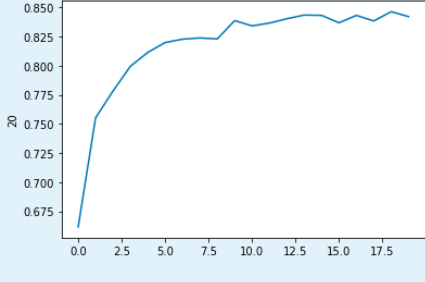
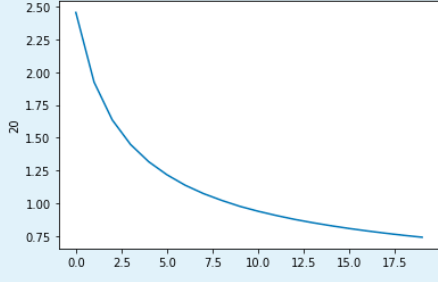
Après l'entraînement, le modèle est testé sur les données de validation et la précision est enregistrée. Si la précision de cette époque est supérieure à la précision de l'époque précédente, le compteur d'arrêt précoce est réinitialisé à zéro. Sinon, le compteur est incrémenté de un. Si le compteur atteint la valeur de `stop_early`, l'entraînement est interrompu.

Enfin, le modèle est testé sur les données de développement et les résultats sont enregistrés.

Configurations des entraînements

	use_mlp	add_lemmas	Class weight : balanced	Nb epochs
Entraînement 1	False	False	No	20
Entraînement 2	False	False	Yes	20
Entraînement 3	True	True	Yes	30
Entraînement 4	True	False	Yes	25

Résultats de l'entraînement

	Early stopping	Val accuracy	Loss
Entraînement 1	Yes	NA	NA
Entraînement 2	Yes		
Entraînement 3	No		
Entraînement 4	No		

Les résultats des différents entraînements sont proches.

V. Résultats et Conclusion

Évaluation

	Dev accuracy	Test accuracy
Entraînement 1	0.845	0.851
Entraînement 2	0.852	0.861
Entraînement 3	0.859	0.863
Entraînement 4	0.857	0.870
Baseline	0.836	0.843

Le classificateur utilisant le réseau de neurones MLP ainsi que des poids, mais également des vecteurs denses, obtient la meilleure métrique « test accuracy » (0.870). Cependant, on ne constate qu'une très petite variation (± 0.1) entre les résultats des différents classificateurs. Le rééquilibrage des poids semble néanmoins aidé légèrement à l'amélioration de la précision.

Conclusion

Nous ne pouvons pas décider quel est le meilleur classificateur. En effet, les scores obtenus sont relativement proches de ceux du modèle baseline. Ainsi, ces classificateurs ne sont pas assez performants et peuvent être améliorés.

Nous avons remarqué précédemment que pour les entraînements 3 et 4, il n'y a pas eu d'arrêt prématuré, l'augmentation du nombre d'époques pourrait optimiser leurs résultats. De plus, l'utilisation du réseau de neurones MLP a aussi augmenté les performances, envisager l'approfondissement du modèle en ajoutant des couches pourrait également être une piste d'amélioration.

Par ailleurs, le temps d'exécution des entraînements étant excessivement long (même en utilisant un GPU), il faudrait considérer la programmation parallèle afin de maximiser les différentes boucles "for".