Bio-Inspired Intelligence

Classifieur de texte à partir d'un RNN

Marion Vertessen

Introduction

Ce rapport se rapporte à un travail pratique réalisé pour le cours de Bio-Inspired Intelligence à l'université Lyon 1. Il a pour but d'implémenter un classifieur sur un dataset fourni.

I. Préparation des données

Le dataset est divisé en trois jeux de données :

- Le dataset train pour l'apprentissage
- Le dataset validation pour mesurer la capacité de généralisation du réseau
- Le dataset test pour vérifier que la généralisation des performances mesurées

Dans un premier temps, on sépare les données fournies selon deux catégories : les phrases à analyser et les labels.

On coupe les phrases faisant plus de 10 mots afin qu'elles ne fassent pas trop de caractères et on ajoute des caractères vides pour aux phrases trop courtes pour qu'elles fassent toutes la même taille pour passer dans le réseau de neurones. Il est important de noter que pour les phrases ou il n'y a pas assez de caractère, on aligne la phrase à droite.

Après avoir réalisé une telle opération, on crée une correspondance entre les mots et un identifiant. Pour cela, on utilise la bibliothèque torchtext.vocab.

De plus, on utilise un encodage onehot qui va permettre de transformer l'identifiant du mot en tensor faisant la taille du nombre de mots présent dans le vocabulaire et qui possède un 1 pour l'identifiant du mot et des 0 dans le reste des indexes.

II. Apprentissage du RNN

On utilise l'architecture présente sur la figure 1.

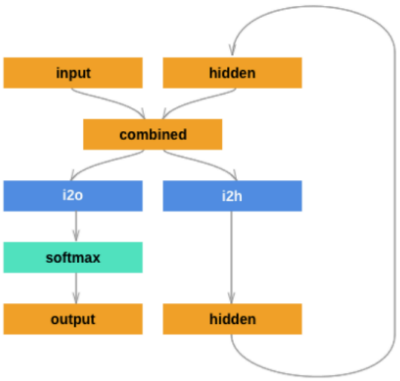

Figure 1 : Architecture du RNN utilisé

Cependant, il s'avère que cette architecture n'est pas assez puissante pour un encodage one-hot mot par mot. On ajoute donc une couche d'embedding qui va permettre d'alimenter le combined.

On utilise un DataLoader afin d'alimenter notre réseau de neurones mot par mot.

III. Résultats

On réalise une phase d'apprentissage que l'on valide à chaque epochs pour vérifier que l'on n'a pas de overfitting. Sur les figures ci-dessous, on retrouve en bleu la courbe d'apprentissage et en jaune la courbe de validation.

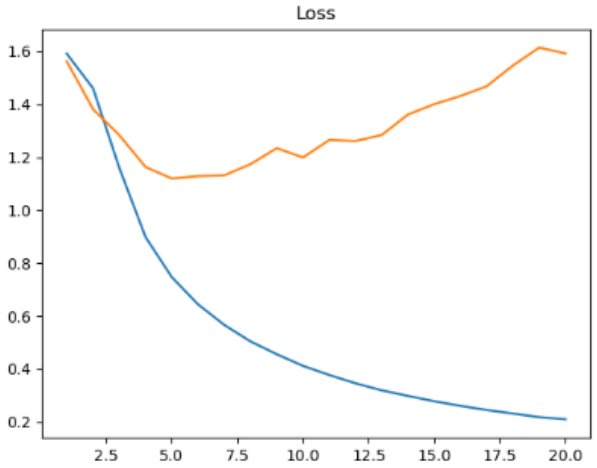

Figure 2 : loss avec nb_mot_par_phrase = 10, LR = 0.001

On remarque sur la figure 2 que l'on a de l'over fitting cela est surement dû au fait que le jeu de données est relativement petit.

Afin de le réduire nous avons augmenté le nombre de mots par phrase à 30 et diminuer le LR à 0.0009. On obtient alors la courbe en figure 3.

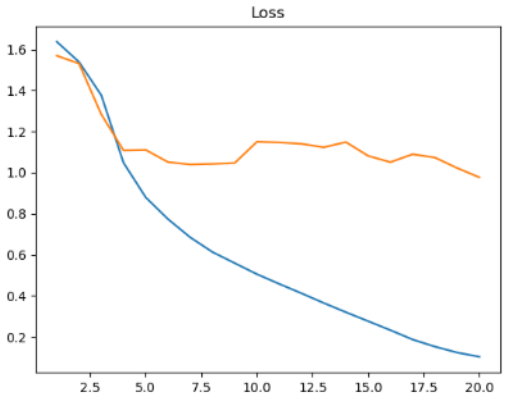

Figure 3 : loss avec nb_mot_par_phrase = 30, LR = 0.0009

On remarque sur cette courbe qu'il y a moins d'overfitting. Cependant, les meilleurs résultats sont obtenus pour les phrases contenant 40 mots comme on peut le voir sur la figure 4.

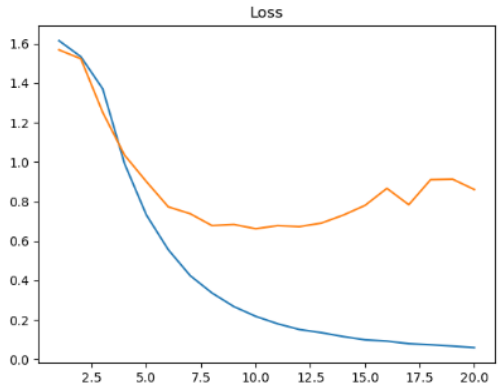

Figure 3 : loss avec nb_mot_par_phrase = 40, LR = 0.0009

On mesure maintenant la valeur de l'accuracy sur la phase d'apprentissage et sur la phase de validation, on obtient le résultat sur la figure 4.

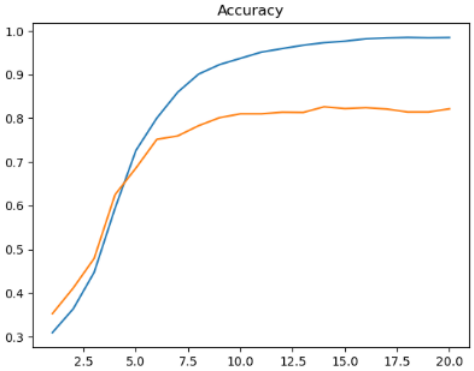

Figure 3 : accuracy avec nb_mot_par_phrase = 40, LR = 0.0009

On réalise ensuite une phase de test et on obtient la matrice de confusion présente en figure 4.

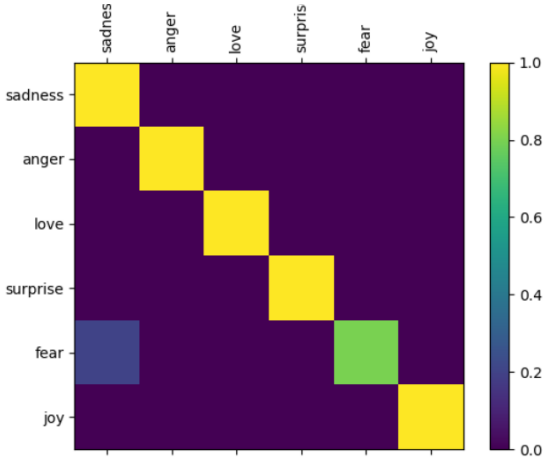

Figure 4 : Matrice de confusion avec nb_mot_par_phrase = 40, LR = 0.0009

Conclusion

Pour conclure, malgré les problèmes d'overfitting que nous avons pu observer, on obtient des résultats corrects sur les trois phases : apprentissage, validation et test.