M2-IA

Rapport Projet

Bio-Inspired Machine Learning

Classifieur de textes à partir d'un RNN

Groupe:

NESR Mohamad

P1811466

Enseignant:

ARMETTA Frederic

Année universitaire **2022/2023**

I) Introduction:

Ce TP est réalisé dans le cadre de l'UE d'intelligence Bio-Inspirée et a pour objectif de nous familiariser avec la classification de textes à partir d'un RNN. Le travail est réalisé à l'aide de PyTorch en python avec des datasets sous format txt.

II) Preparation des donnees:

Nos données sont divisés en 3 jeux de données:

- Données d'entraînement *"train.txt"*
- Données de validation *"val.txt"*
- Données de test *"test.txt"*

Chaque ligne est composée d'une phrase suivie d'un ";" et ensuite une émotion décrivant le sentiment du texte. On commence alors en chargeant nos données (**load_file**(filename)) et en les séparant chacune en deux listes différentes, une contenant les phrases, et une autre contenant les émotions.
Ensuite, on met en place un **padding** (15 mots par phrase: **pad_text**(encoded, max_length)) pour les phrases afin qu'ils aient tous la même longueur afin de les passer a notre RNN plus tard.

Après avoir fait cela, on crée un vocabulaire (**build_vocab**(iterator)) pour nos données. Cela consiste à faire correspondre chaque mot présent dans notre texte a un indice .
Afin de réduire la taille de notre vocabulaire, on a appliqué certains filtres sur nos textes. Tout d'abord, on a enlevé tous les **stopwords** de nos textes. Ensuite nous avons enlevé tous les mots répétés moins de 5 fois ou plus de 1000 fois. Cela va nous permettre d'accélérer l'apprentissage et d'améliorer nos résultats.

On applique également un **codage one hot** (classe **do_one_hot**()) à nos données. Cela crée pour chaque mot, un tensor de la taille NxN où N est le nombre de mots différents dans notre vocabulaire. Afin que nos dimensions soient cohérentes, nous faisons tout d'abord un codage one hot de nos labels avant de créer notre dataloader et on crée celui des textes durant l'apprentissage.

III) Apprentissage du RNN:

On a créé un RNN très similaire à celui de l'exemple du sujet sauf qu'on a rajouté une couche embedding pour alimenter la couche combined (classe **RNN**(nn.Module)).

Ce RNN a trois couches linéaires (entièrement connectées): une couche d'entrée-vers-incorporation (**i2e**), une couche d'entrée-vers-caché (**i2h**) et une couche d'entrée-vers-sortie (**i2o**). La couche d'entrée prend en entrée un vecteur d'entrée de taille fixe, qui est ensuite mappé sur un espace d'incorporation à dimension inférieure par la couche i2e. La couche cachée prend en entrée le vecteur concaténé de l'incorporation d'entrée et de l'état caché précédent, et produit un nouvel état caché. La couche de sortie prend en entrée le vecteur concaténé de l'incorporation d'entrée et de l'état caché actuel, et produit la prédiction finale. Le réseau utilise une fonction d'activation **log-softmax** pour produire des probabilités de sortie.

On a également créé un **getdataset** customisé afin de charger nos données et appliquer différents réglages tel que le **padding** et la **tokenisation** etc… Ensuite, on passe le dataset customise à

TensorDataset qui est ensuite passé au **DataLoader**. Une fois cela fini, on peut démarrer l'apprentissage, la validation et les tests.

Les hyperparamètres choisis:

ETA	0.0007
Epochs	15
Batch	128
Taille de phrase	15
Hidden	128
Embedding	128
Loss	NLLLoss()
Optimizer	Adam

Figure 1: les hyper paramètres optimaux de l'apprentissage

IV) Resultats:

Les résultats obtenus sont satisfaisants. On atteint une précision de **0.92** en apprentissage et de **0.79** en validation. De même, les tests ont une précision assez élevée de **0.76**. La perte à son tour est toujours en régression et se stagne après le 10eme épisode (ne diminue que de peu).

Perte:

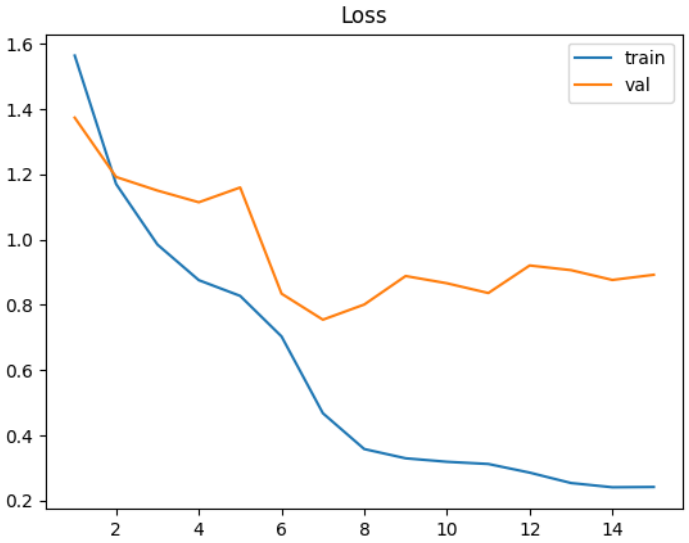

Figure 2: courbe de la perte de l'entraînement (bleu) et de la validation (orange) en fonction des épisodes

Precision:

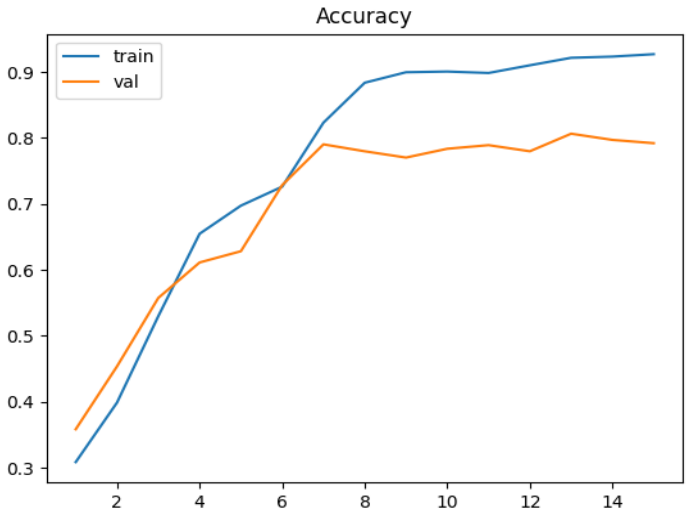

Figure 3: courbe de la précision de l'entraînement (bleu) et de la validation (orange) en fonction des épisodes

Matrice de confusion:

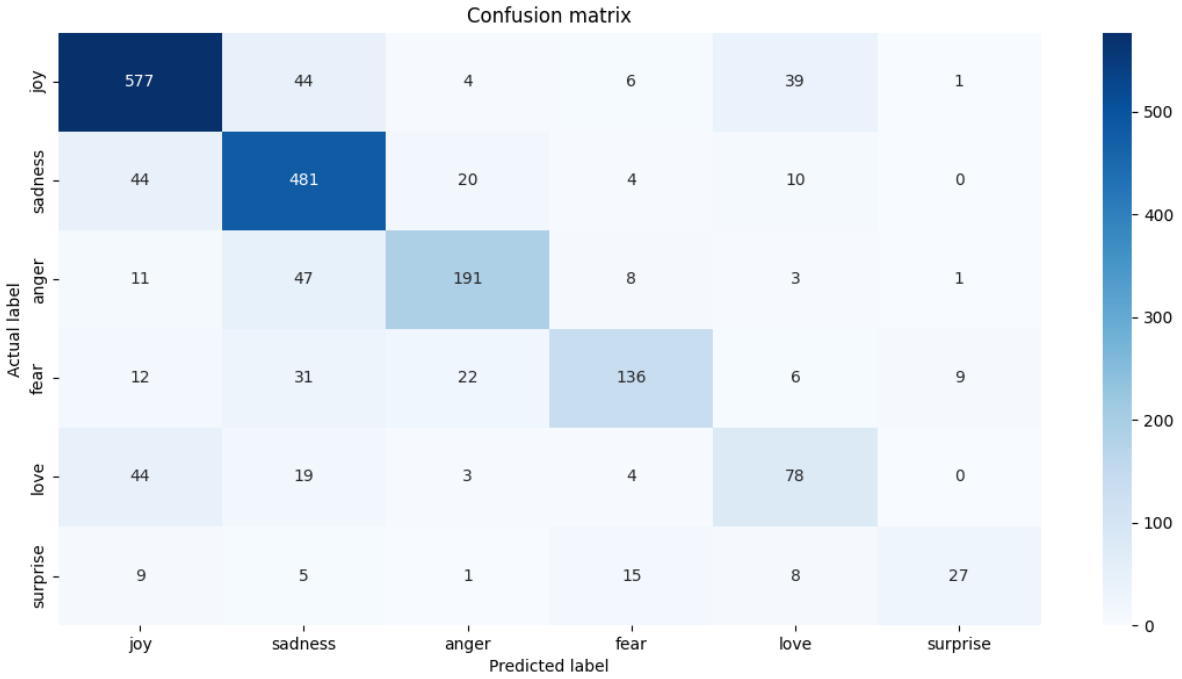

Figure 4: Matrice de confusion des resultats de test

Comme on peut voir dans la matrice de confusion, on a des meilleures prédictions sur joy et sadness. Cela est dû au fait qu'elles sont beaucoup plus présentes dans notre dataset de train que les autres: *{'joy': 5362, 'sadness': 4666, 'anger': 2159, 'fear': 1937, 'love': 1304, 'surprise': 572}*.
On peut donc constater que moins une émotion est présente, plus on aura du mal à la détecter.

V) Pour aller plus loin:

Afin d'améliorer les résultats on a:

- Retirer les stops words
    - **remove_stop_words**(text, stpwd)
- Filtrer les mots rares (moins de 5 répétitions) les mots très communs (plus de 1000 répétitions)
    - **tf_idf**(text)

VI) Conclusion:

Ce TP a été une très bonne introduction au NLP. Ceci est une première expérience avec les RNN et le traitement de texte. Une des difficultés principales de ce TP était l'encodage one hot. La fonction de l'encodage est très simple. Cependant, savoir à quel moment l'appliquer a nécessité pas mal de temps et beaucoup de tests.

Pour finir, les résultats sont plutôt satisfaisants et l'objectif du TP est atteint. Certaines améliorations restent possibles comme l'utilisation d'un LSTM par exemple qui est peut être meilleur qu'un RNN (il se peut qu'on retombe sur les mêmes résultats) mais vu la contraintes du temps et les fonctionnalités demandées pour le TP, cela n'a pas été implémenté. Une autre amélioration qui aurait pu améliorer beaucoup nos résultats est le fait de limiter les effets de déséquilibre des classes (on aurait obtenue une meilleure matrice de confusion).

Remarque:

Si vous voulez voir le code exécuté. Une version google colab déjà compilée et exécutée a été créée:
https://colab.research.google.com/drive/1Nwf-hZ0qF20dqsxOct7Wr0njDArSL3-E?usp=sharing