Université Claude Bernard Lyon 1 M2 IA
Département Informatique

Intelligence Bio-Inspirée

Classifieur de textes à partir d'un RNN (Pytorch)

---

1 Travail à réaliser

Développer et tester votre classifieur (texte -> sentiment) sur le dataset fourni, en suivant les étapes précisées ci-dessous.

Rendu par binômes obligatoirement :

- (1) Déposer le code commenté en format python (.py) dans la colonne dédiée de Tomuss par binôme (si vous utilisez un notebook, merci d'extraire en fichiers python)
- (2) Déposer votre compte rendu dans la colonne Tomuss dédiée :
  - lister les fonctionnalités développées
  - pour votre meilleur hyperparamétrage : courbes d'apprentissage (train + validation), matrice de confusion (test)
  - analyser vos résultats

2 Préparation des données

2.1 Chargement du dataset

Le dataset (prétraité) disponible sur Moodle est divisé en trois jeux :

- train : utilisé pour l'apprentissage
- validation : utilisé pour mesurer la capacité de généralisation de votre réseau, et choisir l'itération propice à l'arrêt de l'apprentissage
- test : utilisé pour vérifier que la généralisation des performances mesurées. C'est la performance du jeu de test qui doit être au final reportée lors d'une évaluation impartiale.

Formatage des jeux d'apprentissages ligne par ligne : <texte de longueur variable> <séparateur> <sentiment>

Développer une fonction load_file(file) en utilisant la fonction native open(...), qui retourne la liste des textes et la liste associée des émotions (en format "texte").

2.2 Préparation des données par encodage one-hot

Un texte doit être codé mot par mot par codage one-hot, cet encodage est efficace en NLP.

Deux étapes :

- Créer une correspondance : mot -> id_mot
  utiliser pour cela la fonction build_vocab_from_iterator(..) de torchtext.vocab
- Créer un encodage one-hot (en tensor pytorch) de vos phrases
  utiliser pour cela la fonction one_hot(..) de torch.nn.functional

3 Apprentissage du RNN

3.1 Architecture du réseau récurrent : *class RNN(nn.Module)*

Vous pouvez utiliser et adapter le code associé à figure 1 :

- Cette architecture n'est pas assez puissante pour un codage one-hot par mot.
  Ajouter une couche d'embedding de taille emb_size : self.i2e = nn.Linear(input_size, emb_size), c'est donc la couche d'embedding qui alimente désormais la couche "combined"
- le code proposé prévoit un batch (première dimension du tensor en entrée) de taille 1 dans la fonction init_hidden(..).
  adapter pour prévoir un batch de taille variable
- *adapter éventuellement la dernière couche suivant la fonction loss que vous choisissez*

3.2 Préparation des batchs

- Votre réseau est prêt, il s'alimente avec des tensors (batch_size, vocabulary_size)
  tester votre réseau, en l'appelant avec un mot unique (batch_size = 1, sans récurrence)
- il est plus efficace de traiter par batch
  préparer vos données globales (tensor(sentence_length[1], batch_size, vocabulary_size)), pour alimenter votre réseau mot par mot (tensor(batch_size, vocabulary_size)), utilisez si vous le souhaitez DataLoader de torch.utils.data

1. toutes vos phrases ont la même taille (si besoin les rogner ou compléter par un mot spécial de rembourrage)

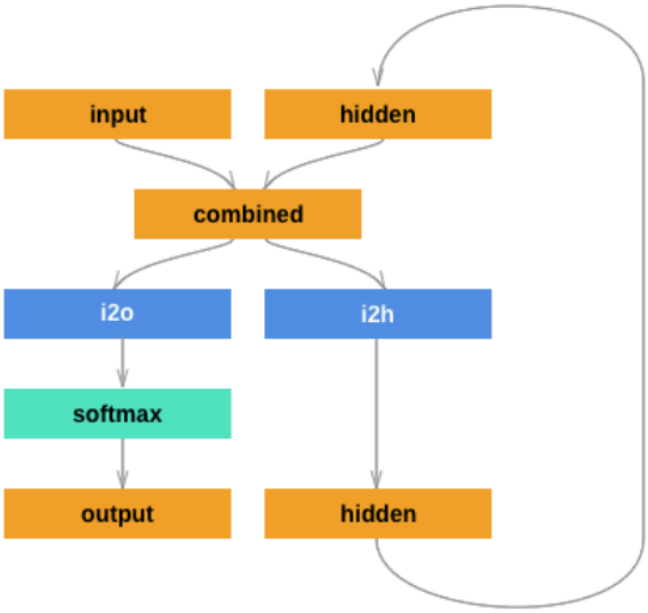

FIGURE 1 – Architecture pour RNN présentée ici : `https://pytorch.org/tutorials/intermediate/char_rnn_classification_tutorial`

3.3 Apprentissage du réseau

— *développer votre boucle d'entraînement par epoch/batch*
— analyser les performances, paramétrer votre réseau

Les performances devraient dépasser les 50% d'accuracy (et même plus ...).

3.4 Ajout d'une tâche d'apprentissage (suivant avancement)

Certaines phases contiennent des négations et peuvent être étiquetées grâce à des mots clefs que vous identifiez ('not', 'n t', etc.). L'objectif de cette partie est d'étudier l'influence de cette tâche (classifieur binaire : [contient négation | ne contient pas de négation]) sur la tâche initiale déjà développée : ceci améliore ou dégrade l'apprentissage ?

Il s'agit d'un apprentissage multi-tâche, le classifieur associé à cette tâche se branche sur le précédent réseau en sortie de la couche *combined*, et partage donc certains poids appris.

— étiqueter les phrases, réaliser l'apprentissage de la nouvelle tâche et étudier les performances
— analyser l'influence de l'introduction de cette nouvelle tâche sur la tâche initiale

3.5 Approfondissements (facultatif)

Vous pouvez alors si vous souhaitez approfondir :

— *limiter les effets du déséquilibre des classes (loss adaptée)*
— *retirer les stops words (inutiles en classification)*
— *étudier la distribution des mots et filtrer également les mots rares (TF-IDF)*

Une solution pragmatique consiste généralement à utiliser un LLM prédéveloppé et entraîné sur un très grand corpus afin de brancher votre classifieur et finaliser votre apprentissage sur votre dataset, ce qui permet de très bonnes performances (l'objectif de ce projet est cependant de créer et manipuler votre propre modèle RNN en utilisant uniquement le jeu de données fourni).