Entrainement de modèles de classification textuelle

---

monôme :
Mohamed Amine KASMI

Encadrants :
Philippe MULLER
Chloé BRAUD

Contents

1 Introduction

Dans ce TP, nous allons apprendre à entraîner des modèles de classification textuelle en explorant trois approches basées sur les embeddings de mots. À travers la tâche de classification des sentiments sur le jeu de données IMDB, nous allons expérimenter différents réglages pour analyser leurs performances.

2 Entraînement de modèles de classification textuelle à partir d'embeddings statiques précalculés

2.1 Chargement des embeddings statiques

- Nous avons utilisé **GloVe** (Global Vectors for Word Representation), un modèle d'embeddings statiques disponible via `gensim`.
- Le modèle `glove-wiki-gigaword-100`, qui génère des vecteurs de 100 dimensions pour chaque mot, a été chargé.
- Le vocabulaire a été extrait pour identifier les mots présents dans le modèle.

2.2 Encodage des phrases

- Les phrases ont été encodées en calculant la moyenne des vecteurs des mots qui les composent.
- Les mots sont prétraités en minuscules et débarrassés de leur ponctuation avant d'être convertis en vecteurs.
- La fonction d'encodage a été testée sur une phrase courte pour évaluer sa performance et optimiser son implémentation.

2.3 Jeu de données utilisé

- Le jeu de données IMDB, contenant des critiques de films annotées avec des sentiments positifs ou négatifs, a été utilisé.
- Une petite fraction du jeu de données a été sélectionnée pour réduire le temps d'entraînement :
  - 4% pour l'entraînement.
  - 2% pour le test.

2.4 Encodage des données

- Une fonction a été développée pour convertir les critiques en vecteurs à l'aide des embeddings GloVe et associer les vecteurs aux labels correspondants.
- Cette étape a permis de préparer les données pour l'entraînement des modèles de classification.

2.5 Entraînement des modèles

- Deux modèles simples ont été entraînés :
  - **Régression logistique** avec une précision de 76,6% sur les données d'entraînement.
  - **Perceptron multicouche (MLP)** avec une précision de 81,2% sur les données d'entraînement.

2.6 Évaluation des modèles

- Les modèles ont été testés sur un ensemble de validation :
  - Le modèle MLP a obtenu une précision de 73,8% sur l'ensemble de test.

2.7 Analyse des résultats

- Les performances obtenues sont raisonnables pour un modèle utilisant des embeddings statiques.
- Le score de 73,8% sur les données de test reflète la capacité limitée des embeddings statiques à capturer les subtilités contextuelles, un problème que les modèles d'embeddings dynamiques (comme BERT) pourraient mieux résoudre.
- Toutefois, ces résultats sont satisfaisants pour une approche simple et rapide basée sur GloVe et des classifieurs standards.

3 Un modèle de classification neuronal à partir de zéro

Nous allons développer des modèles en `PyTorch` en commençant à zéro. Le premier modèle sera un réseau "feed forward" entièrement connecté. Voici les étapes principales :

3.1 Préparation des données

- Transformation des données en tenseurs torch pour les transférer sur le GPU/TPU si disponible.
- Tokenisation des données d'entrée en utilisant un tokenizer pré-entraîné (`BertTokenizerFast` dans cet exemple).
- Application d'un *padding* pour aligner les séquences de texte à une longueur uniforme.

3.2 Définition du modèle

Un modèle simple avec une seule couche d'embedding, une couche intermédiaire et une couche de sortie. L'architecture comprend :

- Une couche d'embedding gérant les séquences de longueur variable via `nn.Embedding`.
- Une couche linéaire transformant les embeddings vers une dimension cachée.
- Une activation ReLU pour la non-linéarité.
- Une deuxième couche linéaire pour produire la sortie.

3.3 Entraînement

- Création des *DataLoaders* pour gérer les *batches*.
- Utilisation de `torch.optim.Adam` comme optimiseur et de la fonction `nn.CrossEntropyLoss` pour calculer l'erreur.
- Mise à jour des paramètres après chaque lot en suivant les étapes classiques : propagation avant, calcul de la perte, rétropropagation et mise à jour des poids.

3.4 Évaluation

- Évaluation des performances via des métriques comme l'accuracy, la précision, le rappel et le score F1.
- Calcul de la matrice de confusion pour une analyse détaillée des prédictions.

3.5 Visualisation des résultats

Les courbes de perte et d'accuracy sont tracées pour chaque époque, permettant de vérifier la convergence du modèle.

- La courbe de perte montre une diminution régulière indiquant que l'entraînement progresse bien.

- L'accuracy augmente progressivement, témoignant d'une amélioration des performances.

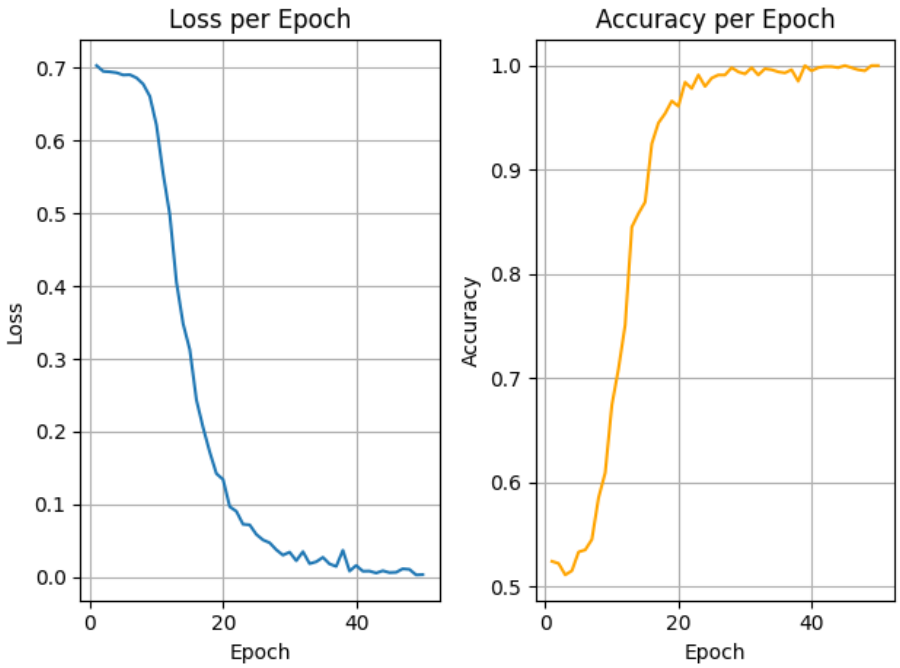

Figure 1: Visualisation

4 Fine-tuning d'un modèle contextuel préentrainé

Nous avons configuré un **entraîneur (Trainer)** pour entraîner un modèle en ajustant soigneusement les paramètres d'entraînement, tels que :

- **Taille de batch** : 64 pour l'entraînement et l'évaluation, **nombre d'époques** : 10, et **learning rate** : 3×10^{-5}.
- **Stratégies d'évaluation et de sauvegarde** activées à chaque époque pour monitorer les performances.
- Utilisation de la **précision mixte (fp16)** pour accélérer l'entraînement et économiser de la mémoire sur GPU.
- Ajustement des hyperparamètres : décroissance de poids, steps de préchauffage et accumulation des gradients.

Nous avons ensuite effectué les étapes suivantes :

1. Chargé la métrique d'évaluation `accuracy` pour mesurer les performances du modèle.
2. Implémenté une fonction personnalisée `compute_metrics` pour calculer les scores de précision sur le jeu de test.
3. Entraîné le modèle avec `Trainer`, et récupéré le meilleur modèle basé sur l'accuracy.
4. Testé le modèle sur un exemple individuel, en générant des **probabilités normalisées** avec la fonction Softmax et en affichant la classe prédite.
5. Généré les prédictions pour le jeu de test et tracé un **histogramme des probabilités prédites**.

4.1 Observation

La distribution des probabilités montre une forte confiance du modèle dans ses prédictions. Bien que cela puisse refléter un bon apprentissage, une validation rigoureuse est indispensable pour exclure tout surapprentissage ou biais des données.

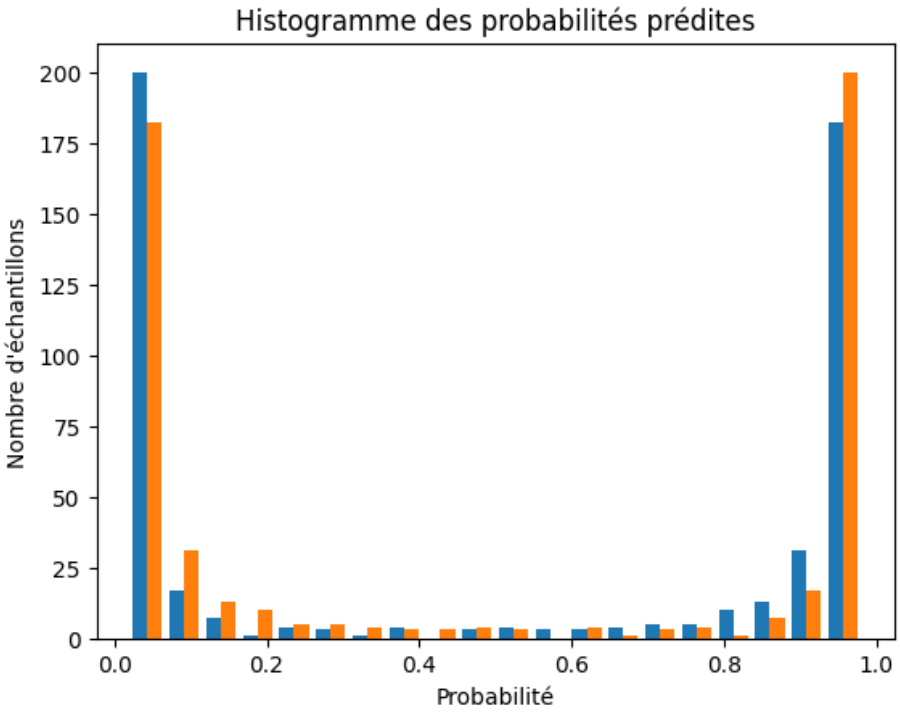

Figure 2: Visualisation

5 conclusion

Ce TP nous a permis d'explorer différentes approches pour la classification textuelle, allant des embeddings statiques aux modèles contextuels pré-entraînés. Nous avons analysé l'impact des réglages sur les performances, renforçant ainsi notre compréhension des modèles textuels modernes. Ces compétences sont essentielles pour résoudre divers problèmes en NLP.