TP deep learning for NLP

21 novembre 2024

Sam Vallet

Résumé

Ce rapport a pour objectif de présenter l'implémentation de deux modèles, un modèle Bag-of-Words et un modèle CNN, pour la classification de critiques de films. L'objectif est de déterminer si une critique est positive ou négative. Le but de ce TP n'est pas d'obtenir les meilleurs performance mais plutot d'implementer ces deux modèles pour bien comprendre leur fonctionement.

1 Introduction

La classification de critiques de films en catégories positives ou négatives est une tâche essentielle du traitement du langage naturel (NLP). Cette analyse permet d'extraire des informations précieuses sur les préférences des utilisateurs ou sur la perception globale d'un film.

Nous avons pour ce problème exploré deux modèles de traitement de language (BOW et CNN nous rappelons rapidement leur principe théorique).

Bag-of-Words (BoW) Une méthode classique qui repose sur une représentation vectorielle simple des textes. Chaque document est transformé en un vecteur représentant les fréquences des mots. Les mots sont traités de manière indépendante, sans tenir compte de l'ordre ou du contexte. Cette tâche est réalisée par une couche d'embedding qui permet de convertir les fréquences des mots en vecteurs. Ensuite, des couches linéaires dans le réseau de neurones sont utilisées pour effectuer la classification.

Convolutional Neural Networks (CNN) Les CNN utilisent des filtres convolutifs appliqués aux représentations vectorielles des textes, ce qui permet de capturer les relations locales et contextuelles dans les données. Après l'embedding des données textuelles, des couches de convolution sont utilisées pour extraire ces informations, suivies de couches linéaires pour effectuer la classification.

2 Préparation des Données

Avant de pouvoir passer les données à nos modèles, plusieurs transformations sont nécessaires. Ces étapes de prétraitement sont détaillées ci-dessous.

Nettoyage des textes Tout d'abord, nous convertissons tous les mots du texte en minuscules. Ensuite, nous supprimons certains caractères et les remplaçons par des espaces. Nous ajoutons également des espaces autour de la ponctuation afin de traiter les mots séparément des symboles. Ce nettoyage est effectué par la fonction `clean_str`. Lors de la lecture du texte, chaque ligne est traitée individuellement à l'aide de cette fonction. Les mots sont ensuite séparés, et la liste ainsi obtenue est ajoutée au dataset. Cette tâche est réalisée par la fonction `loadTexts`.

Division du Dataset Dans notre étude, nous disposons de deux datasets : un contenant des critiques positives et un autre des critiques négatives. Nous allons diviser ces données en trois ensembles distincts : un pour l'entraînement, un pour la validation et un pour les tests. Pour ce faire, nous utilisons 70% des données pour l'entraînement, 15% pour la validation et les 15% restants pour l'ensemble de test. Nous faisons attention d'avoir 50% de texts negatifs et positifs pour chaque partie, nous profitions de cette étape pour crées les labels (1 pour positif et 0 pour negatif) que nous ajoutons a nos datasets.

Création du dictionnaire La première étape consiste à créer un dictionnaire qui associe chaque mot unique à un identifiant numérique. Ce dictionnaire est construit à partir du jeu de données d'entraînement, en parcourant chaque phrase et chaque mot. Nous ajoutons deux tokens à notre dictionnaire : le token `'<unk>'` pour les mots inconnus (ceux qui ne sont pas présents dans le jeu de données d'entraînement) et le token `'<pad>'` pour effectuer le padding et ainsi garantir que toutes les séquences aient la même longueur.

Dernières préparations Nous passons ensuite nos données sous forme de tenseurs. Les mots sont vectorisés à l'aide du dictionnaire : chaque mot de la phrase est remplacé par son identifiant dans le dictionnaire. Si un mot ne figure pas dans le dictionnaire, il est remplacé par l'identifiant du token `<unk>`. Cette opération est réalisée par la fonction `sentence_to_tensor`. Nous effectuons le padding à l'aide de la fonction `collate_fn`, ce qui permet d'ajuster les séquences à une longueur uniforme. Les données sont ensuite organisées en lots pour faciliter l'entraînement. Nous avons paramétré

`shuffle = True` afin de mélanger les critiques positives et négatives. La préparation des lots est effectuée grâce à la fonction `DataLoader` de PyTorch, et nous avons choisi une taille de lot de 32 (batch size = 32).

3 Modèles de Classification

Nous présenterons ici l'architectures des modèles choisit, leurs resultats et le choix de leurs hyper parametres.

Bag-of-Words Notre modèle Bag-of-Words (BOW) possède l'architecture suivante :

- **Entrée** : Séquence de mots, où chaque mot est représenté par son indice dans le vocabulaire.
- **Embedding** : Transformation des mots en vecteurs denses (embeddings) qui capturent des informations sémantiques.
- **Moyenne des embeddings** : Agrégation des informations des mots en une seule représentation dense de la phrase, calculée par la moyenne des embeddings.
- **Couche linéaire** : Calcul de la prédiction à partir de la représentation moyenne de la phrase à travers une couche linéaire.
- **Sigmoïde** : Fonction d'activation pour obtenir une probabilité binaire, avec des résultats compris entre [0,1], ce qui permet de faire de la classification binaire.

Nous utilisons la fonction d'activation `sigmoid` pour obtenir un résultat compris entre [0, 1], ce qui est adapté à la classification binaire. Concernant les couches linéaires, nous commençons avec une seule couche, mais nous prévoyons d'ajouter des couches cachées pour évaluer si cela améliore les résultats. Pour l'opération d'agregations des embeddings nous avons prit la moyenne mais la somme est également envisagable.

Modèle CNN Notre modèle CNN possède l'architecture suivante :

- **Entrée** : Séquence de mots, chaque mot étant représenté par son indice dans le vocabulaire, qui est transformé en vecteur dense à l'aide d'une couche d'Embedding.
- **Convolution** : Application d'une couche convolutive 1D sur les embeddings pour capturer les caractéristiques locales des séquences de mots. Cette opération est effectuée avec un noyau de taille `kernel_size` et un nombre de filtres `num_filters`.
- **Max pooling** : Après la convolution, un *max pooling* est utilisé pour obtenir la meme dimension.
- **Couche linéaire** : Une couche linéaire est utilisée pour transformer la sortie du pooling en une prédiction binaire.
- **Sigmoïde** : Une fonction d'activation `sigmoid` est utilisée pour obtenir une probabilité binaire, avec des résultats compris entre [0,1], adaptée à la classification binaire.

Nous commencons avec une couche de convolutions et une lineaire que nous pourrons modifier.

Fonction de Perte Pour la fonction de perte, nous utilisons `BCELoss` (Binary Cross-Entropy Loss), qui est particulièrement adaptée pour la classification binaire, où le modèle prédit une probabilité pour chaque exemple.

Hyperparamètres Dans cette section, nous discuterons des différents hyperparamètres utilisés dans nos modèles, ainsi que des valeurs que nous avons testées pour optimiser les performances.

- **Optimiseur :** Nous avons testé deux optimisateurs : Adam et SGD.
- **Learning Rate :** Nous avons expérimenté différents taux d'apprentissage pour chaque optimiseur afin de trouver la valeur qui permet une convergence rapide tout en évitant le sur-apprentissage.
- **Embedding Dimension (embedding_dim) :** La dimension des embeddings a été testée pour explorer la capacité du modèle à capturer des représentations significatives des mots. Des valeurs comme 100 et 300 ont été utilisées.
- **Kernel Size (kernel_size) :** Pour le modèle CNN, nous avons exploré différentes tailles de noyau de convolution (3, 5) afin de capturer des informations à différentes échelles.

— **Nombre d'époques (num_epochs) :** Nous commençons avec 10 époques et nous observons le score de validation à chaque époque pour déterminer quand arrêter l'entraînement.
— **Taille du batch (batch size) :** Nous avons testé différentes tailles de batch (32, 64).

4 Résultats

Nous commençons par explorer les différents hyperparamètres pour le modèle BOW. Au départ, nous utilisons un embedding de taille 100 et une taille de batch de 32. Après avoir testé les deux optimizers, nous choisissons Adam avec un taux d'apprentissage de 0.001. La valeur que nous regardons est l'accuracy.

Paramètres du modèle	**Entraînement**	**Validation**	**Test**
Adam (lr = 0.001)	88.96%	79.00%	75.53%
Adam (lr = 0.01)	96.57%	75.67%	73.53%
Adam (lr = 0.1)	95.17%	75.87%	74.27%
SGD (lr = 0.1)	62.93%	64.73%	63.80%
Taille du batch (32)	88.96%	79.00%	75.53%
Taille du batch (64)	84.87%	77.47%	74.60%
Embedding (100)	88.96%	79.00%	75.53%
Embedding (300) Taille du batch (64)	88.60%	77.27%	76.47%

On conserve donc l'optimiseur adam avec un learning rate de 0.001, un embedding de 300 et un batch size de 64. Nous faisons des testes également pour le modele CNN : Les parametres de base utilisé sont : kernel size = 3, embedding = 300 et batch size 64.

Modèle Paramètre	**Train**	**Validation**	**Test**
Adam (lr = 0.001)	51.00%	50.13%	51.40%
Adam (lr = 0.01)	48.33%	47.00%	48.27%
Kernel Size (5)	51.00%	50.60%	50.73%
Embedding (100)	51.70%	51.47%	51.07%

TABLE 1 – Résultats obtenus pour différents hyperparamètres du modèle.

Nous gardons donc le modèle avec l'optimizer Adam, un kernel size de 3 et un embedding de 300.

Nous testons ensuite le modèle BOW avec une couche linéaire caché en plus. Nous comparons ces performances sur le set de validation au modèle BOW précedents. Ce graphique nous permet également de regarder les performances du modèle en fonction de l'époque.

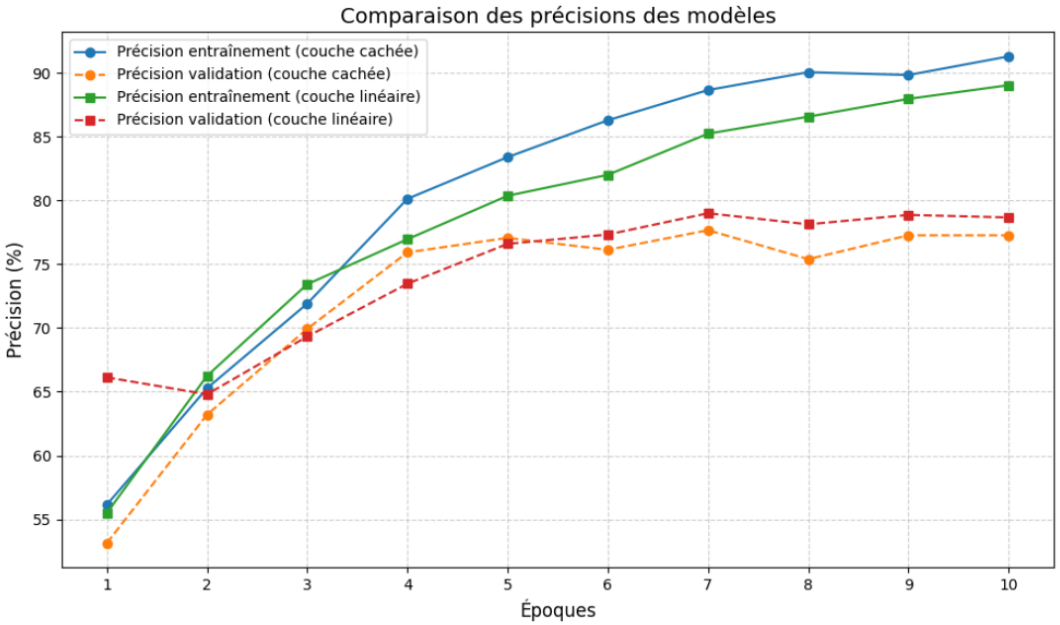

FIGURE 1 – BOW avec et sans une couche linéaire cacher

Nous remarquons que les deux modèles ont des performances a peu près similaire, le score de validation stagne après l'époque 7, continuer l'entrainement après n'est donc pas nécessaire.

5 Conclusion

Dans ce rapport, nous avons testé deux implémentations différentes pour un problème de classification de phrases et réalisé plusieurs tests afin de déterminer les meilleurs hyperparamètres. Pour une étude plus approfondie, il aurait été possible d'explorer l'ajout de couches linéaires cachées supplémentaires pour le modèle BOW, ainsi que d'enrichir le modèle CNN en ajoutant d'autres couches de convolution.