TP Noté - Traduction Automatique

L3 - Intelligence Artificielle

Présentation

Dans ce TP vous allez entraîner des modèles de traduction automatique de l'anglais vers le français. Plusieurs niveaux de difficultés sont possibles, de sorte qu'il soit possible d'aller progressivement vers les niveaux les plus complexes.

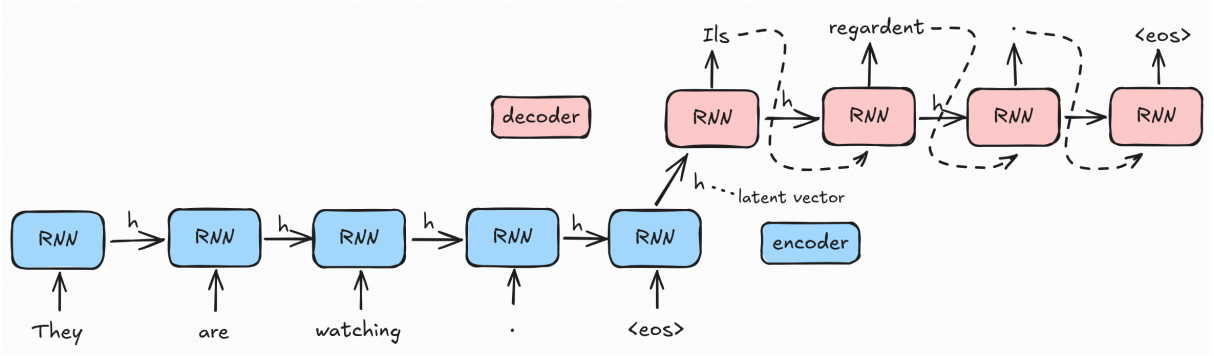
En plus du code, vous devrez aussi rendre un rapport expliquant les résultats de vos expériences pour chacun des niveaux de difficultés. La note finale va dépendre de **l'ensemble des niveaux de difficultés** que vous avez attaqués ainsi que **de la pertinence de votre compte rendu**.

Aucun code ne vous est fourni. Vous ne serez pas noté sur la qualité de votre code, mais si vous me rendez un code illisible je pourrais vous pénaliser. Le dataset kaggle est un fichier `csv` contenant des paires `(anglais, français)` permettant d'entraîner de manière supervisée vos modèles.

Ce TP est volontairement difficile. Vous devriez avoir des questions à me poser. N'hésitez pas à me demander de l'aide et des conseils, je suis là pour ça !

Entraînement d'un modèle de traduction automatique

Tout modèle de traduction automatique est composé de deux modules : un `encoder` et un `decoder`. L'encodeur va lire la phrase dans la langue source (ici l'anglais) afin de produire un ou plusieurs vecteurs latents. Le décodeur peut ensuite générer la traduction correspondante dans la langue cible (ici le français).



Les phrases sont tokenisées et chaque token est donné l'un après l'autre en input. Le décodeur génère aussi la phrase sous forme tokenisée en samplant les tokens petit à petit. Les tokens samplés sont redonnés en input afin que le modèle puisse savoir l'état actuel de la traduction.

Cependant lors de l'entraînement on ne se permet pas de sampler les tokens de traduction un par un pour calculer la loss finale. À la place on réalise du *teacher forcing*, ce qui consiste à remplacer les inputs du décodeur par les *ground truths* afin de l'aider à rester proche de la traduction attendue lors de son apprentissage. De plus, cela va grandement accélérer l'entraînement de vos Transformers.

La loss est simplement la *cross-entropy* appliquée sur les outputs du décodeur par rapport aux tokens attendus. Les tokens sont identifiés par un `id` unique et sont interprétés en input par une couche `nn.Embedding`.

Niveaux de difficultés

1. Character-level models

Ce premier niveau vous demande d'établir les grandes lignes sur lesquelles vous construirez les multiples améliorations par la suite. **Vous devez entraîner un RNN où chaque token est un simple caractère.** Vous devez utiliser directement la classe fournie par PyTorch (`nn.RNN`).

Vous devez :

- Construire un tokenizer très simple qui divise chaque phrase en leur séquence de caractère.
- Construire le `dataset` qui s'occupe de charger les paires de phrases, les tokeniser et mapper aux `ids` correspondants.
- Instancier votre RNN.
- Créer toute la boucle d'entraînement. Vous devez mesurer la `loss`, la `top-1` et `top-5` accuracy des prédictions du décodeur.
- Implémentez une méthode qui permet de traduire des phrases en samplant les tokens provenant du décodeur petit à petit.

Cette première partie est déjà assez difficile et très longue, mais il est nécessaire de bien la faire pour facilement ajouter des modifications par la suite.

N'implémentez pas toutes les métriques d'un coup. Vous pouvez vous contenter de la loss pour commencer. N'essayez pas d'entraîner le meilleur des modèles avant de passer au reste du TP, ne passez pas plus de 2h à entraîner vos modèles ici.

La tokenization étant très simple, caractère par caractère, vous aurez des inputs/outputs qui seront très longs. Traiter de telles séquences prendra trop de calcul et de mémoire, donc vous devriez ignorer les phrases trop longues du dataset ou bien les couper. Ne passez pas beaucoup de temps à implémenter un tokenizer complexe, vous allez de toute façon le remplacer par la suite.

N'oubliez pas que vous devez ajouter les tokens spéciaux `<eos>` et `<bos>` en plus des tokens de vos phrases. Aussi, pensez à garder dans votre tokenizer ou votre dataset la taille de votre vocabulaire (utile pour vos couche `nn.Embedding` et votre couche de sortie).

Avant de passer au niveau suivant, pensez à prendre des screenshots de vos expériences et commencez déjà à rédiger votre rapport.

2. Tokenizers

Les tokens utilisés dans la première partie sont très simples mais ils vous forcent à manipuler de très longues séquences. **Dans cette partie, vous allez devoir utiliser des tokenizers bien plus performants.** Utilisez `sentencepiece` pour entraîner vos propres tokenizers et remplacer votre ancien par ceux-ci.

Vous devez :

- Entraîner un tokenizer par langue.
- Remplacer votre tokenizer par vos deux nouveaux tokenizers entraînés.
- Modifier la fonction qui génère la traduction pour prendre en compte les nouveaux tokenizers.

Le dataset est trop gros pour être utilisé pour entraîner vos tokenizers. N'utilisez qu'un sous-ensemble.

3. Transformer

Vous êtes maintenant prêts à utiliser un Transformer. Utilisez les classe fournies par PyTorch pour instancier votre modèle.

Vous devez :

- Instancier votre Transformer.

- Bien définir les masques d'attention qui permettront d'entraîner le Transformer.
- Comparez votre Transformer avec votre RNN. Quel est le meilleur modèle ? Lequel est le plus rapide ? Lequel consomme le moins de mémoire ?

Faites bien attention à vos masques d'attention !

4. Sampler

Votre fonction qui génère la traduction en samplant depuis les sorties du décodeur n'est pas la meilleure. **Cette partie vous demande de remplacer le sampling classique par le top-p sampling.** Ce sampler ne considère que les tokens les plus probables avant de les sampler en renormalisant la lois de probabilité.

Vous devez :

- Remplacer votre sampler par un sampling `top-p`.

5. RNN *from-scrach*

Cette partie vous demande d'implémenter votre propre RNN, sans utiliser la classe PyTorch. Commencez par écrire votre propre classe `RNN` en utilisant `nn.RNNCell`, puis implémentez votre propre `RNNCell` aussi.

Vous devez :

- Implémenter votre propre `RNN` et `RNNCell`.

6. Transformer *from-scrach*

Vous êtes arrivés si loin ? Alors il est maintenant temps d'implémenter votre propre Transformer. Comme pour le RNN, commencez par implementer la classe `Transformer` en réutilisant les couches préconstruites `nn.TransformerEncoderLayer` et `nn.TransformerDecoderLayer`. Ensuite, implémentez vos propres `TransformerEncoderLayer` et `TransformerDecoderLayer` en utilisant `nn.MultiheadAttention`.

Vous devez :

- Implémenter votre propre `Transformer`.