TP NMT

Ce TP va vous initier aux outils de traduction automatique neuronaux et vous propose d'entraîner un modèle de traduction très simple sur un jeu de données réduit et de l'évaluer. Pour ce faire nous vous fournissons un fichier de configuration minimal et relativement peu performant (*config-base.yaml*). La partie la plus chronophage de ce TP sera de faire évoluer le fichier de configuration pour apprendre des modèles qui soient plus performants. Les temps d'entraînement d'un modèle de traduction peuvent être relativement long si les calculs sont effectués sur le CPU uniquement. Ainsi, vous pouvez utilizer une GPU sur Google Colab ou, si vous disposez d'une carte GPU NVIDIA sur votre machine, vous pouvez installer CUDA, ce qui accélérera sensiblement le temps d'entraînement. La première (Colab) est vivement conseillée parce que ca vous évite d'installer du software sur votre machine locale.

Pour utiliser une GPU sur Google Colab

- Inscrivez-vous à Google Colab (si vous n'avez pas déjà de compte Google) et demandez l'accès.
- Vous pouvez ouvrir un nouveau fichier Colab depuis votre compte Google Drive en cliquant sur `+ New->More->Google Colaboratory`. Le fichier colab est un Jupyter Notebook qui utilise des recources computationnelles en cloud.
- Pour activer l'utilisation de GPU pour votre notebook: `Runtime->Change runtime type->Hardware Accelerator->GPU` .

> AIDE: Avant de commencer le TP, vous pouvez consulter les pages d'introductions à Colab et aux Notebooks

Préambule

Installez OpenNMT-py à l'aide de pip : `pip install OpenNMT-py`

Note 1 : OpenNMT3 requiert des modules lourds (notamment PyTorch ~750Mo), assurez-vous donc d'avoir suffisamment de place sur votre disque. **Note 2** : Les modèles neuronaux peuvent être relativement gros (~75Mo) et peuvent rapidement occuper beaucoup de mémoire ! **Note 3** : Si vous travaillez avec un Notebook Jupyter (sur Colab ou en locale), il est utile de connaître certaines commandes magiques. Par example:

- Le symbol `!` permet d'exécuter des commandes comme dans un terminal. Par example:
    - `!pip install OpenNMT-py`
    - `!cd path/to/dir`.
- Les commandes shell dans le Notebook sont exécutées dans une subshell temporaire. Si vous souhaitez modifier le répertoire de travail de manière plus durable, vous pouvez utiliser la commande magique `%cd path/to/dir`.
- On peut aussi écrire et executer plusieurs commandes shell dans la même cellule avec la commande magique `%%shell`. Par example, on peut executer ces commandes pour créer un nouveau répertoire qui contient un file textuel:

```
%%shell
mydir="nouveaurepertoire"
mkdir $mydir
echo "blabla" > $mydir/myfile.txt
```

Resources

Pour répondre aux questions théoriques, vous pouvez vous appuyer sur les slides du cours, mais aussi plein d'information accessibles gratuitement sur internet. Entre autre:

- Livre de référence sur le deep learning: https://www.deeplearningbook.org/
- Site web avec beaucoup de notions de ML expliquées simplement et avec exemples: https://machinelearningmastery.com/blog/
- Introduction à la NMT : https://towardsdatascience.com/neural-machine-translation-15ecf6b0b
- Tutoriel plus détaillé sur la NMT : https://arxiv.org/abs/1703.01619
- wikipedia, stack overflow, etc.

Préparation des données

- **Q1** : Que contiennent les fichiers *IWSLT10_BTEC.train.en.txt* et *IWSLT10_BTEC.train.fr.txt* ?
- **Q2** : Pour chacun des fichiers dans les répertoires train, dev et test, exécutez la commande suivante : `awk -F '\' '{print $NF}' file_int.txt > file_out.clean.txt`. Pour chaque fichier clean créé, exécutez la commande suivante (en changeant LANG par en ou fr selon la langue dans laquelle est le fichier) : `perl tokenizer.perl -l LANG -lc < ./file_in.LANG.clean.txt > file_out.LANG.tok.txt`. Quelles différences voyez-vous entre les fichiers *.clean.txt* et *.tok.txt* ? Quel est l'intérêt de l'opération effectuée ?
- **Q3** : Pourquoi avoir un corpus séparé en 3 parties (train, dev et test) ? A quoi vont servir chacun de ces fichiers ? (Note: bien expliquer la différence d'utilisation entre dev et test et la raison de cela)

Création d'un modèle de TA

Avant de créer un système de TA français/anglais, nous allons créer un vocabulaire basé sur les données d'apprentissage:

```
onmt_build_vocab -config config-base.yaml -n_sample -1
```

> Note: pour la compréhension de cette commande, ainsi que de suivantes, référez vous à la documentation d'OpenNMT-py à l'adresse suivante : http://opennmt.net/OpenNMT-py/index.html

- **Q4** : pourquoi un système de TA a-t-il besoin d'un vocabulaire ? pourquoi est-ce que l'on utilise un vocabulaire basé sur les données d'apprentissage et non-pas un vocabulaire quelconque, plus universel ?

- **Q5** : Quels informations nous donnent les lignes

```
[...] Counters src:9978
[...] Counters tgt:8194
```

Verifiez que les chemins dans le fichier config-base.yml sont bons et ensuite lancez l'entraînement d'un modèle avec la commande suivante :

```
onmt_train -config config-base.yaml
```

- **Q6** : A quoi correspond la valeur 'acc' affichée par le script ? Vaut-il mieux que cette valeur soit faible ou élevée à la fin de l'apprentissage ? Pourquoi ? Même question pour 'ppl'. Le code qui permet de calculer ces variables se trouve dans le fichier statistics.py de OpenNMT-py.
- **Q7** : Ouvrez le fichier de configuration *config-base.yaml* et observez son contenu. Décrivez en détail à quoi correspondent les paramètres suivants (n'hésitez pas à demander a Google!):
    - train_step
    - valid_step
    - enc_layers
    - dec_layers
    - enc_rnn_size
    - dec_rnn_size
    - batch_size

Evaluation d'un modèle de TA

Lors de l'entraînement, le script a sauvegardé plusieurs modèles dans le répertoire spécifié dans le fichier de configuration. Faites traduire le fichier de test *IWSLT09_BTEC.testset.fr.tok.txt* par chacun des modèles sauvegardés et conservez chaque fichier de traduction généré avec un nom unique.

```
onmt_translate -model VOTRE_MODELE.pt -src IWSLT09_BTEC.testset.fr.tok.txt -output
pred.txt
```

- **Q8** : Choisissez un exemple pour montrer la traduction de différents modèles. Montrez la phrase source suivie des phrases traduites.

En pratique, il n'est pas possible d'évaluer les traductions manuellement, ainsi des métriques automatiques ont été développées pour évaluer les traductions de manière automatique. L'une de ces métrique s'appelle BLEU.

- **Q9** : Entre quelle valeur et quelle valeur est compris un score BLEU ? Vaut-il mieux que la valeur obtenue soit élevée ou faible ? Pourquoi ?
- **Q10** : Pour chacun des modèles intermédiaires, calculer un score BLEU à l'aide du script `multi-bleu.perl`. Quel modèle a le meilleur score BLEU ?

> Usage script: `perl multi-bleu.perl /path/to/reference/translations < /path/to/model/predictions`. Par exemple: `perl multi-bleu.perl BTEC-en-fr/test/IWSLT09_BTEC.testset.en.tok.txt < models/base/pred.txt`.

Optimisation des paramètres

Pour l'instant le modèle utilisant la configuration minimale obtient de très mauvais scores. Charge à vous de faire évoluer la configuration afin d'obtenir un modèle plus performant. Ainsi, vous pouvez faire évoluer les

paramètres décris ci-après et évaluer chaque modèle sur l'ensemble de développement (dev). Enfin, vous pourrez évaluer votre meilleur modèle sur l'ensemble de test.

Afin de bien comprendre l'importance de chaque paramètres dans la performance globale du système, **vous veillerez à changer les paramètres un à un** (et non tous en même temps). De la sorte, vous pourrez quantifier précisément le rôle de chaque paramètre.

Aussi, veillez à sauvegarder les modèles appris avec des configurations différentes dans des répertoires différents avec leur fichier de configuration.

Plusieurs paramètres peuvent être changés afin d'obtenir de meilleurs traduction, par example:

- **Q11 - nombre de training steps** : Augmenter le nombre de steps permet-il d'obtenir un meilleur score BLEU ? Pourquoi? Jusqu'à combien de steps ? comment utiliser l'option `early7.2.3_stopping` (voir ici) pour trouver le nombre optimal de steps ?

- **Q12 - nombre de couches** : Augmentez le nombre de couches de l'encodeur et du décodeur (n'allez pas au delà de 3 couches). Cela permet-il toujours d'obtenir des meilleurs résultats ? Pourquoi ?

- **Q13 - nombre d'unités** : Faites varier le nombre d'unités pour l'encoder et le décodeur (n'allez pas au delà de 512. Obtenez-vous de meilleurs résultats ? Pourquoi ?

- **Q14 - attention [OPTIONNEL]** : Les architectures avec mécanismes d'attention sont devenus la norme dans de nombreuses applications comme la traduction, la reconnaissance de la parole, la description d'image, etc. (cet article de blog constitue une excellente introduction en la matière). Ajouter un mécanisme d'attention en changeant le paramètre de *global_attention* dans le fichier de configuration en le passant de *none* à *mlp*. Observez-vous un gain de performance ? Pourquoi ? Comment-est ce que c'est mécanisme fonctionne ?

La qualité des traductions générées par un système de traduction automatique dépend aussi de certains paramètres au décodage (`onmt_translate`). L'un d'eux est le **beam search**.

- **Q15** : Expliquez ce qu'est le beam search. Quelle est la taille par défaut du beam ?
- **Q16** : Changez la taille du beam en la faisant varier de 1 à 10. Comment évolue le score BLEU ?

Maintenant essayez de saisir la combination de paramètres la plus efficace possible pour atteindre le meilleur score BLEU sur le dev. Vous pouvez changer les paramètres que vous venez de tester ainsi que les autres paramètres disponibles pour l'architecture du modèle, son entraînement et le décodage.

- **Q17** Quelle est le meilleur score BLEU que vous pouvez obtenir sur dev ? Maintenant, vous pouvez tester votre meilleur modèle en calculant son score BLEU sur l'ensemble de test. Quel résultat vous obtenez?

- **Q18** Observez les prédictions de votre meilleur modèle. Avez-vous l'impression que la qualité des traductions générées par ce modèle sont meilleures de celles générées par le premier modèle que vous aviez entraîné à la question Q6 ? Donnez quelque exemple.