Et5-Info

Module : Traitement Automatique des Langues - Traduction Automatique

Année universitaire : 2022-2023

Projet

Consignes générales

- Vous avez le choix entre deux sujets. Le sujet 1 correspond à une analyse et une évaluation à grande échelle du moteur de traduction neuronale OpenNMT. Le sujet 2 a pour objectif l'implémentation d'un désambiguiseur morpho-syntaxique en utilisant un réseau de neurones récurrent de type LSTM (Long short-term memory). Ce sujet exige des compétences théoriques et pratiques sur les réseaux de neurones récurrents et le framework PyTorch.
- Vous pouvez utiliser un dépôt ***git*** à partager entre les membres de votre groupe pour assurer la compatibilité entre les modifications que vous faites. Ce répertoire git devra être structuré comme suit :
    1. Un fichier texte nommé ***README*** contenant les indications nécessaires pour exécuter les scripts Python que vous avez développés. Vous indiquerez notamment les options pour la ligne de commande ainsi que des exemples d'utilisation.
    2. Un dossier ***src*** pour les codes source de vos scripts Python. La lisibilité du code sera l'un des critères de notation (commentaires précis, concis, clairs et bien orthographiés).
    3. Un dossier **doc** pour votre rapport écrit en LaTeX ou en Word et compilé en un PDF de 5 à 7 pages nommé ***nom1-nom2-nom3.pdf*** qui décrit votre travail. Vous y décrirez l'objectif du projet, les résultats d'évaluation du moteur de traduction OpenNMT, ses points forts et ses limitations. Si vous savez comment résoudre ces limitations, mentionnez-le et décrivez vos idées en les présentant comme des pistes pour un travail futur. Vous devez également inclure une petite section qui décrit la contribution respective de chaque membre du groupe. Le rapport peut être écrit en anglais ou en français.
    4. Un dossier **data** pour les données manipulées. Veillez à utiliser les mêmes noms de fichiers utilisés en TPs et en projet.
    5. Un dossier **tp** pour les codes source de vos scripts Python des deux TPs. Il faut ajouter à ce dossier un ***README*** contenant les indications nécessaires pour exécuter ces scripts Python.

Vous m'enverrez par mail (nasredine.semmar@cea.fr) un lien vers votre projet **au plus tard le dimanche 5 mars 2023**.

Sujet 1 : Evaluation du moteur de traduction neuronale OpenNMT sur un corpus parallèle anglais-français

Travail demandé

Vous allez analyser et évaluer système de traduction neuronale OpenNMT. Le but de ce projet est de montrer que vous pouvez installer, évaluer et rédiger un rapport décrivant les principaux résultats, points forts et limitations d'un du moteur de traduction neuronale.

Documentation sur OpenNMT: https://opennmt.net/OpenNMT/

Guide d'installation : https://pypi.org/project/OpenNMT-py/

I. Installation (30 minutes, 6 Go)

Prérequis : Python >= 3.6, PyTorch == 1.6.0

a. Installer un environnement Python pour le Machine Learning avec Anaconda :
   - MS Windows 10 : https://mrmint.fr/installer-environnement-python-machine-learning-anaconda
   - Ubuntu >= 18.04 LTS: https://phoenixnap.com/kb/how-to-install-anaconda-ubuntu-18-04-or-20-04

b. Installer le framework Deep Learning PyTorch (https://pytorch.org/get-started/locally/)
   - `conda install pytorch torchvision -c pytorch`

c. Installer Gensim (https://pypi.org/project/gensim/)
   - `pip3 install --upgrade gensim`

d. Installer Numpy
   - `pip3 install numpy`

e. Installer OpenNMT-py (https://pypi.org/project/OpenNMT-py/#setup)
   - `pip3 install OpenNMT-py`

Remarque : Vous pouvez éventuellement utiliser Google Colab :

- Initiation au Deep Learning avec Google Colab (https://moov.ai/fr/blog/deep-learning-avec-google-colab/)
- Google Colab: charger et enregistrer des données provenant de sources externe (https://colab.research.google.com/notebooks/io.ipynb)

II. Expérimentation

1. Vérifier l'installation d'OpenNMT sur le corpus bilingue anglais-allemand contenant 10k phrases parallèles tokenisées fourni sur le site d'installation (https://pypi.org/project/OpenNMT-py/#setup):

```
wget https://s3.amazonaws.com/opennmt-trainingdata/toy-ende.tar.gz

tar xf toy-ende.tar.gz

cd toy-ende
```

Remarques :

- Utiliser le fichier de configuration YAML « toy_en_de.yaml » fourni.
- La commande « gpu_ranks: [0] » est mis en commentaire puisque la machine virtuelle n'a pas de GPU. Il faudrait enlever # si votre poste possède une GPU.
- Les instructions pour la préparation de données (Step 1: Prepare the data), l'entrainement du modèle (Step 2: Train the model) et la traduction (Step 3: Translate) sont décrites sur le lien : https://pypi.org/project/OpenNMT-py/#setup
- Les scripts d'évaluation de la traduction sont décrits sur le lien : https://github.com/ymoslem/MT-Evaluation. Il faut utiliser le score BLEU.

2. Vérifier que le moteur OpenNMT fonctionne correctement en utilisant les petits corpus TRAIN, DEV et TEST que je vous ai transmis pour le TP2 :

   TRAIN : Europarl_train_10k.en et Europarl_train_10k.fr
   DEV : Europarl_dev_1k.en et Europarl_dev_1k.fr
   TEST : Europarl_test_500.en et Europarl_test_500.fr
3. Evaluer OpenNMT en utilisant le score BLEU sur le corpus de test.

III. Evaluation sur des corpus parallèles en formes fléchies à large échelle

1. Réaliser des expérimentations sur deux corpus parallèles pour le couple de langues anglais-français. La taille (en nombre de phrases) de l'ensemble des corpus bilingues utilisés pour l'apprentissage et le développement du système de traduction OpenNMT est décrite dans le tableau ci-dessous :

N° du run	Apprentissage (nombre de phrases)	Tuning (nombre de phrases)
1	100K (Europarl)	3,75K (Europarl)
2	100K+10K (Europarl+Emea)	3,75K (Europarl)

2. Evaluer OpenNMT en utilisant le score BLEU. Il faudrait effectuer 2 runs avec, pour chacun d'entre eux, un test avec des données du domaine et un autre hors-domaine.

Remarques :

1. Pour pouvoir comparer les différents résultats, vous utiliserez les mêmes corpus d'évaluation du système de traduction statistique Moses.
2. Vous pouvez récupérer la totalité des corpus parallèles Europarl pour le domaine général et Emea pour le domaine médical à partir des liens http://opus.nlpl.eu/Europarl.php et http://opus.nlpl.eu/EMEA.php. Il faut utiliser « Bottom-left triangle: download plain text files (MOSES/GIZA++) ».

Corpus d'apprentissage (TRAIN) :

- 100K (Europarl) : Europarl_train_100k.en, Europarl_train_100k.fr ➔ Il faut prendre les 100 000 premières phrases du corpus.
- 10K (Emea) : Emea_train_10k.en, Emea_train_10k.fr ➔ Il faut prendre les 10 000 premières phrases du corpus.

Corpus de développement (DEV) :

- Europarl_dev_3750.en ➔ Il faut prendre 3750 phrases à partir du corpus Europarl en commençant par la phrase au rang 100 001
- Europarl_dev_3750.fr ➔ Il faut prendre 3750 phrases à partir du corpus Europarl en commençant par la phrase au rang 100 001

Corpus de test (TEST) :

- Europarl : Europarl_test_500.en, Europarl_test_500.fr ➔ Il faut prendre 500 phrases à partir du corpus Europarl en commençant par la phrase au rang 103751
- Emea : Emea_test_500.en, Emea_test_500.fr ➔ Il faut prendre 500 phrases à partir du corpus Emea en commençant par la phrase au rang 10001

Remarques :

- L'idéal pour le corpus de test d'extraire de manière aléatoire un ensemble de 500 paires de phrases à partir du corpus Europarl comme un corpus correspondant au domaine et 500 autres paires de phrases à partir du corpus Emea comme un corpus hors-domaine.
- Vous pouvez utiliser la procédure train_test_split du module sklearn.model_selection pour créer les corpus d'apprentissage et de test via une partition au hasard.

IV. Evaluation sur des corpus parallèles en lemmes à large échelle

1. Lemmatiser à l'aide de la plateforme NLTK tous les corpus parallèles du domaine général (Europarl) et du domaine de spécialité (Emea) de l'exercice III.
2. Réaliser des expérimentations sur les deux corpus parallèles pour le couple de langues anglais-français selon le protocole expérimental de l'exercice III.
3. Evaluer OpenNMT en utilisant le score BLEU en utilisant le même protocole d'évaluation décrit dans l'exercice III.
4. Quelles conclusions peut-on avoir à partir de ces deux évaluations (celle de l'exercice III et celle de l'exercice IV) ?

Remarques :

- La lemmatisation consiste à remplacer dans les corpus parallèles du domaine général (Europarl) et du domaine de spécialité (Emea) tous les mots (en formes fléchies) par leur lemme. Les lemmes sont générés par le lemmatiseur de la plateforme NLTK. Il faut bien entendu garder la structure d'origine des corpus parallèles, c'est-à-dire une phrase par ligne.
- Pour la lemmatisation des corpus en anglais, il faut utiliser WordNetLemmatizer :

```
import nltk
from nltk.stem import WordNetLemmatizer
nltk.download('wordnet')
lemmatizer = WordNetLemmatizer()
```

Guide d'installation et d'utilisation de WordNetLemmatizer:
https://www.holisticseo.digital/python-seo/nltk/lemmatize

- Pour la lemmatisation du français, il faut utiliser FrenchLefffLemmatizer :
  `from french_lefff_lemmatizer.french_lefff_lemmatizer import FrenchLefffLemmatizer`

  Guide d'installation et d'utilisation de FrenchLefffLemmatizer:
  https://github.com/ClaudeCoulombe/FrenchLefffLemmatizer

Sujet 2 : Un désambiguïseur morpho-synatxique (Part-of-Speech tagger) basé sur des réseaux de neurones récurrents (LSTM)

Travail demandé

Vous allez utiliser une LSTM pour implémenter un POS tagger. Le fonctionnement du modèle neuronal est décrit comme suit :

Supposons que nous avons une phrase en entrée composée de M mots : w_1, …, w_M ou $w_i \in V$ (Vocabulaire). Supposons que T correspond à l'ensemble des étiquettes morpho-syntaxique et y_i est l'étiquette du mot w_i. Supposons que la prédiction de l'étiquette morpho-syntaxique du mot w_i est $\hat{y}_i$. La sortie du modèle est une séquence $\hat{y}_1$, …, $\hat{y}_M$ où $\hat{y}_i \in T$.

Pour réaliser la prédiction, il faut passer une LSTM sur la phrase en entrée. L'état caché au timestep i est représenté par h_i. Chaque étiquette morho-syntaxique lui est assignée un seul index. La prédiction pour l'étiquette de $\hat{y}_i$ est donnée par la formule suivante :

$$\hat{y}_i = \text{argmax}_j \left(\log \text{Softmax}(Ah_i + b)\right)_j$$

L'étiquette morpho-syntaxique prédite correspond à l'étiquette ayant la valeur maximale dans le vecteur.

Documentation sur les LSTMs en Pytorch :
https://pytorch.org/tutorials/beginner/nlp/sequence_models_tutorial.html

Documentation sur les word embeddings :
https://pytorch.org/tutorials/beginner/nlp/word_embeddings_tutorial.html

Installation du framework PyTorch

1. Environnement Windows 10
   a. Installer Anaconda : https://mrmint.fr/installer-environnement-python-machine-learning-anaconda
   b. Lancer Anaconda en tant qu'administrateur :
      i. Installation du Driver CUDA : conda install cudatoolkit=11.0 cudnn
      ii. Installation de PyTorch : conda install pytorch torchvision -c pytorch

2. Environnement Linux (Ubuntu 18.04 LTS)
    a. Installer Anaconda : https://www.digitalocean.com/community/tutorials/how-to-install-anaconda-on-ubuntu-18-04-quickstart-fr
    b. Créer et activer un environnement Conda:
        i. Création: conda create -n env_pytorch python=3.7
        ii. Activation : conda activate env_pytorch
    c. installer PyTorch : pip install torchvision

5. Construction d'un POS tagger basé sur une LSTM à partir d'un petit corpus d'apprentissage

1. Préparation de données
    a. Editer le programme «sample_lstm_pos_tagging_preparing_data.py» et identifier les principales étapes pour préparer les données d'apprentissage.
    b. Lancer le programme «sample_lstm_pos_tagging_preparing_data.py» dans un environnement Pytporch et analyser les sorties.
2. Création du modèle neuronal
    a. Editer le programme «sample_lstm_pos_tagging_training_model» et identifier les principales étapes pour créer le modèle neuronal.
    b. Lancer le programme «sample_lstm_pos_tagging_training_model» dans un environnement Pytporch et analyser les sorties.

6. Amélioration du POS tagger basé sur une LSTM pour prendre en compte un gros corpus d'apprentissage

3. Construire à partir du corpus annoté « pos_reference.txt.lima » les données d'apprentissage (80% du corpus annoté) et les données de test (20% du corpus annoté).
4. Modifier le programme de préparation de données «sample_lstm_pos_tagging_preparing_data.py» pour prendre les nouvelles données d'apprentissage (80% du corpus annoté).
5. Modifier le programme de création et d'entrainement du modèle neuronal «sample_lstm_pos_tagging_preparing_data.py» pour prendre en compte les nouvelles données d'apprentissage (80% du corpus annoté) et les données de test (20% du corpus annoté).

7. Amélioration du POS tagger basé sur une LSTM en ajoutant une représentation basée sur des caractères

Dans l'exemple précédent, chaque mot avait un embedding qui a servi d'entrée au modèle neuronal. L'objectif de cet exercice est d'augmenter ces word embeddings avec une représentation basée sur des caractères.

Supposons que c_w est la représentation basée sur des caractères du mot w et que x_w est le word embedding, l'entrée du modèle neuronal sera la concaténation de x_w et c_w.

Pour obtenir la représentation au niveau du caractère, il faut passer une LSTM sur les caractères d'un mot et considérer w_c comme l'état caché final de cette LSTM.