Sujet 1 - Entrainer un modèle de traduction neuronale OpenNMT

1. Introduction

La traduction automatique neuronale (ou Neural Machine Translation, NMT) est une méthode de traduction automatique qui utilise des réseaux de neurones pour traduire du texte d'une langue à une autre. Parmi les méthodes récentes, la NMT se distingue par sa capacité à produire des traductions plus fluides et naturelles. Cette approche apprend à traduire des phrases en prenant en compte le contexte global. Contrairement aux anciennes méthodes comme la traduction statistique ou basée sur des règles, qui se basent principalement sur des correspondances de mots ou des règles grammaticales prédéfinies, la NMT permet une meilleure prise en compte du sens global des phrases. Cependant, son principal défaut réside dans sa dépendance à de grandes quantités de données pour obtenir des résultats satisfaisants.

2. Présentation du moteur de traduction neuronale OpenNMT

OpenNMT fonctionne selon une architecture encodeur-décodeur utilisée pour la traduction automatique et d'autres tâches de génération de texte. Le processus commence par l'encodeur, qui reçoit une séquence de mots dans la langue source et la transforme en un vecteur de contexte. Ce vecteur contient les informations essentielles de la phrase d'entrée. Ensuite, le décodeur utilise ce vecteur pour générer la traduction mot par mot dans la langue cible. Pour améliorer la qualité de la traduction, OpenNMT utilise un mécanisme d'attention qui permet au modèle de se concentrer sur certaines parties de la phrase source à chaque étape de la génération, au lieu de traiter toute la phrase en bloc. Le modèle est entraîné sur des paires de phrases (source-cible) et ajuste ses paramètres pour minimiser l'écart entre la traduction prédite et la traduction correcte.

(article utilisé : http://arxiv.org/pdf/1805.11462)

L'approche utilisée, ici pour l'entraînement, repose sur un modèle encodeur-décodeur basé sur un LSTM à 2 couches avec 500 unités cachées pour l'encodeur et le décodeur. L'entraînement s'effectue sur un seul GPU.

3. Evaluation du moteur de traduction neuronale OpenNMT sur un corpus en formes fléchies

Description du corpus d'apprentissage et d'évaluation

Le corpus utilisé pour l'apprentissage se compose, dans la run_1, de 100 000 phrases issues du corpus Europarl. Dans la run_2, il se compose des mêmes 100 000 phrases complétées par 10 000 phrases issues du corpus EMEA. Ces deux corpus d'origine couvrent des thématiques distinctes : Europarl contient des textes relatifs aux débats parlementaires européens, tandis que EMEA est centré sur des documents du domaine médical. Cela permettra de comparer la performance de différentes approches impliquant

Le corpus d'évaluation est constitué des 3 750 phrases suivantes du corpus Europarl. Afin de garantir une certaine longueur et complexité des phrases, toutes celles de moins de 80 caractères ont été exclues de l'ensemble des corpus. Ce corpus permet d'éviter le sur-apprentissage en évaluant la performance à chaque grand cycle.

Le corpus de tests est composé de 500 paires de phrases sélectionnées aléatoirement dans le corpus anglais et français, issues du corpus Europarl dans un premier cas, et du corpus EMEA dans un second cas. Bien entendu, ces paires de phrases n'apparaissent ni dans le corpus d'entraînement ni dans le corpus d'évaluation.

Description des métriques d'évaluation : Score BLEU

Le Score BLEU est une métrique fréquemment utilisée pour évaluer la qualité des systèmes de traduction automatique. Il repose sur la comparaison entre la traduction générée automatiquement et une ou plusieurs traductions de référence. Le score est calculé en mesurant la précision des mots tout en tenant compte d'un facteur de pénalisation pour les

phrases trop courtes. Un score élevé indique une plus grande similarité entre la traduction générée et les traductions de référence.

Résultat formes fléchies

Pour évaluer et comparer la performance des différentes méthodes d'entraînement des modèles de traduction neuronaux, ces derniers sont utilisés pour traduire le corpus de test de la langue source vers la langue cible. La traduction obtenue est ensuite comparée au corpus de test dans la langue cible à l'aide du programme multi-bleu.pl, qui fournit un score BLEU.

L'évaluation des différents modèles se fait en fonction des corpus d'entraînement (run_1 et run_2), des corpus de tests (in pour les tests du domaine général, out pour les tests du domaine médical), ainsi que du nombre d'étapes fait dans l'entraînement du réseau neuronaux.

Ces "étapes" (ou "steps" en anglais) correspondent au nombre d'itérations effectuées pendant l'entraînement du modèle. Chaque étape représente un cycle où le réseau de neurones traite un lot de données (batch), met à jour ses paramètres internes via la rétropropagation du gradient, et améliore progressivement sa capacité à générer des traductions précises. Ce nombre est défini par le paramètre train_steps dans le fichier de configuration YAML d'OpenNMT, qui contrôle la durée totale de l'apprentissage. Un nombre élevé d'étapes permet théoriquement au modèle de mieux converger vers une solution optimale, mais nécessite un temps de calcul accru et peut conduire au sur-apprentissage (overfitting) si les données sont insuffisantes ou redondantes. Dans nos résultats, on observe clairement que les scores BLEU s'améliorent significativement entre 1 000 et 10 000 étapes, montrant l'importance de ce paramètre dans l'entraînement des modèles de traduction neuronale.

Le tableau 1 ci-dessous donne le résultat des modèles de chaque cas sous forme de score BLEU.

	in	out	in	out	in	out	in	out
steps	run_1_en_to_fr	run_1_en_to_fr	run_1_fr_to_en	run_1_fr_to_en	run_2_en_to_fr	run_2_en_to_fr	run_2_fr_to_en	run_2_fr_to_en
1000	0	0	1,78	0	2,02	0	0,19	0
2500	non renseigné	non renseigné	non renseigné	non renseigné	1,66	0	non renseigné	non renseigné
10000	22,13	5,58	23,47	5,83	18,77	5,56	23,23	10

Tableau 1 : Score BLEU pour le moteur de traduction neuronale OpenNMT sur un corpus en formes fléchies

On remarque que l'évaluation des modèles sur un corpus de test du domaine général est systématiquement meilleure que sur un corpus de test du domaine médical. Cela s'explique par le fait que l'entraînement des modèles s'est effectué entièrement ou majoritairement sur des textes du domaine général.

On observe également que le modèle entraîné uniquement sur des textes du domaine général (run_1) obtient un score BLEU légèrement supérieur à celui du modèle ayant utilisé en partie des données du domaine médical (run_2). Cette différence s'explique par une meilleure optimisation du modèle dans le premier cas.

Enfin, la traduction du français vers l'anglais présente de meilleurs résultats que celle de l'anglais vers le français dans les deux cas. Une hypothèse pour expliquer ce phénomène serait que les phrases en français se prêtent davantage à la paraphrase que celles en anglais. En effet, les phrases du corpus de test en français correspondent souvent à des formulations alternatives que ne produirait pas nécessairement un traducteur humain.

4. Evaluation du moteur de traduction neuronale OpenNMT sur un corpus en lemmes

NLTK Lemmatizer for English

Le lemmatiseur anglais de NLTK, basé sur WordNet, permet de réduire les mots à leur forme canonique (lemme) en tenant compte de leur catégorie grammaticale (POS tag).

Contrairement à un stemmer qui tronque les suffixes de manière mécanique, le WordNetLemmatizer s'appuie sur une base lexicale structurée pour produire des lemmes valides. Par exemple, il transforme le "running" en "run" (si le POS tag est verbe) ou "better" en "good" (pour l'adjectif). Pour une précision optimale, il nécessite de spécifier le POS tag (nom, verbe, etc.) via des fonctions comme get_wordnet_pos(), qui mappe les tags Penn Treebank vers les catégories WordNet. Sans cette information, il suppose par défaut un nom (noun), ce qui peut générer des erreurs. Intégré nativement à NLTK, il est simple à utiliser mais requiert des ressources téléchargeables (wordnet, averaged_perceptron_tagger). Ces ressources sont téléchargées une seule fois dans le code, et uniquement lorsqu'elles sont utilisées afin d'optimiser le programme.

NLTK Lemmatizer for French

Pour le français, NLTK ne propose pas de lemmatiseur natif, mais des bibliothèques externes comme FrenchLefffLemmatizer (basé sur le lexique LEFFF) comblent cette lacune. Ce lemmatiseur gère les spécificités linguistiques françaises, comme les contractions ("l'" → "le"/"la") et les accords genre/nombre, en s'appuyant sur des règles morphologiques et un lexique enrichi. Contrairement à la version anglaise, il intègre automatiquement le POS tag dans certains cas (ex. "mangées" → "manger" avec le tag verbe), mais une indication explicite du POS améliore la précision. Il nécessite l'installation du package french_lefff_lemmatizer et dépend de ressources comme les modèles de POS tagging de NLTK. Bien que moins intégré que son homologue anglais, il reste un outil puissant pour le TALN francophone.

Le Tableau 2 présente les performances du modèle entraîné sur le corpus run_1 (domaine général) appliqué à des tests in-domain (Europarl) et out-of-domain (EMEA). Contraintes temporelles obligent, seul un modèle unique a pu être entraîné dans cette configuration. Les scores BLEU obtenus s'avèrent légèrement inférieurs à ceux des modèles non lemmatisés dans les mêmes conditions (cf. Tableau 1).

	in	out
steps	run_1_en_to_fr	run_1_en_to_fr
10000	20,7	5,1

Tableau 2 : Score BLEU pour le moteur de traduction neuronale OpenNMT sur un corpus en lemmes

Début d'explication :

Cette différence pourrait s'expliquer par la perte d'informations morphosyntaxiques induite par la lemmatisation. En réduisant les mots à leur forme canonique (exemple : « mangent » → « manger »), le modèle perd accès aux marqueurs de temps, de nombre ou de genre, pourtant cruciaux en traduction. Cette simplification lexicale, pertinente pour des langues flexionnelles comme le français, pourrait nuire à la capacité du réseau à restituer des structures grammaticales précises, notamment face à des phrases complexes ou hors domaine.

5. Points forts, limitations et difficultés rencontrées

De nombreux problèmes ont été rencontrés lors de l'utilisation du GPU pour l'entraînement des modèles. Parfois, une erreur survenait au milieu de l'entraînement indiquant qu'aucun GPU n'était détecté. Lorsqu'un simple redémarrage ne suffisait pas, il était nécessaire de comprendre comment redémarrer manuellement le GPU ou de mettre à jour ou réinstaller des composants graphiques. Ces nombreux redémarrages, installations et recherches de solutions ont impacté le temps disponible pour entraîner les modèles.

La lemmatisation en français avec *french_lefff_lemmatizer* s'est révélée particulièrement chronophage. Le problème a finalement été résolu en instanciant *FrenchLefffLemmatizer()* une seule fois lors de l'initialisation.

Sachant qu'un entraînement avec 2 500 étapes représentait environ une heure, le choix a été fait d'entraîner les modèles par paliers successifs : 1 000, puis 2 500, puis 5 000 étapes… Ce n'est que bien plus tard qu'il a été découvert qu'un entraînement de 10 000 étapes ne prenait en réalité que 30 minutes.

6. Organisation

L'organisation du projet s'est déroulée de manière collaborative, en tenant compte des contraintes de chacun. Antoine, ayant déjà une installation presque finalisée, a pris en charge l'intégralité du développement du code ainsi que la création du dépôt GitHub. Aurélien a apporté son aide dans la rédaction des commentaires du code, permettant ainsi d'assurer une meilleure compréhension du travail réalisé.

La rédaction du rapport a été principalement réalisée par Aurélien, avec le soutien d'Antoine pour la partie des résultats ainsi que pour apporter son point de vue sur certaines descriptions techniques. Aurélien s'est également chargé de la gestion du temps, veillant au bon avancement du projet, et restait disponible pour apporter une aide supplémentaire si nécessaire, bien qu'Antoine n'en ait pas eu besoin.

Nous nous retrouvions chaque après-midi pour avancer ensemble sur une partie du projet, ce qui permettait de garder une dynamique régulière. Après analyse du sujet, nous avons constaté qu'il n'était pas possible de paralléliser efficacement les tâches de développement, ce qui nous a conduits à opter pour cette répartition avec une personne en charge du développement et l'autre en support.

Cette organisation a permis de mener à bien le projet tout en prenant en compte les impératifs personnels d'Aurélien, qui devait rechercher un stage, et ceux d'Antoine, qui allait prochainement commencer le sien.