1. Introduction

-Presentation of the Problem: Neural Machine Translation

Neural Machine Translation (NMT) represents a significant shift from earlier translation methods. Traditional statistical translation models segmented the translation process into several different probabilistic calculations, each handled by distinct modules, concluding translation models and language models. These systems would typically analyze large bodies of text (corpora) to derive statistical probabilities that inform the translation of new sentences.

On the other hand, NMT utilizes a unified model structure based on neural networks, specifically designed to process sequences of data. The architecture of an ANN in this context typically involves layers that process input data (source language text), transform it through multiple layers that capture various linguistic features and dependencies, and finally output the translated text in the target language.

Furthermore, NMT uses a neural network that autonomously adjusts its weights, and can be expanded by adding more layers that allows for increasing complexity and depth in learning, which can lead to improvements in translation quality.

The goal of the project is to help develop a deeper understanding of the OpenNMT engine's functionality and performance. In the paper, we will analyse and evaluate the advantages and disadvantages of this method.

We will describe the process of model training, which is designed for translation tasks from English to French, using bilingual corpora such as Europarl and Emea as training data. We will then further evaluate its efficiency by performing evaluation methods, such as BLEU scores and BERT Score.

2. Presentation of the OpenNMT Neural Translation Engine

- Paragraph describing the operation of OpenNMT (based on articles)

OpenNMT-py is a Python implementation of the OpenNMT framework, which is designed for neural machine translation and other natural language processing tasks. It is a part of the OpenNMT project, which aims to provide open-source tools for neural machine translation.

To use the toolkit, you need to fill a .yaml file to indicate the machine learning parameters.

- Approach used: based on LSTM, Transformers, etc.

LSTM (Long Short-Term Memory): LSTM networks are a type of recurrent neural network (RNN) capable of learning long-term dependencies, making them suitable for sequence prediction tasks such as language modeling and translation.

Transformer: Transformer models have revolutionized the field of neural machine translation by using a self-attention mechanism to evaluate the importance of different words in a sentence regardless of their position.

In the .yaml file, you can specify the encoder and decoder types as RNN or Transformer. By setting the rnn_size and layers parameters, we can define the hidden layer size and the number of layers for the LSTM network. For Transformer models, by setting the model_size, ffn_size, heads, and layers parameters, we can define the size of the model, the size of the feed-forward network, the number of attention heads, and the number of layers. Additionally, by setting the dropout parameter, we can prevent the model from overfitting.

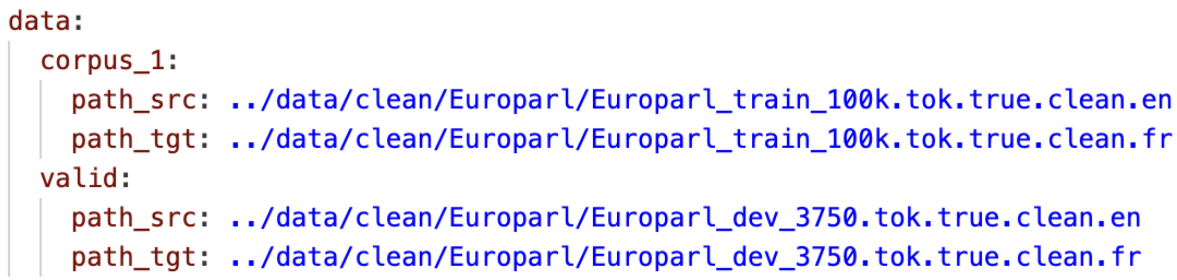
3. Evaluation of the OpenNMT Neural Translation Engine on an Inflected Forms Corpus

Describe the training and evaluation corpus (number of sentences).

This project includes two corpora, Europarl and Emea. The Europarl corpus is extracted from the transcripts of the European Parliament meetings, while the Emea corpus is drawn from public documents of the European Medicines Agency (EMA). The data was processed using Moses for tokenization and automatic case adjustment of words (truecase-model). Sentences no longer than 80 characters were retained for training. To determine which training method produces better results, we divided the data into two experimental groups, namely run1 and run2.

# run1_en_fr.yaml:

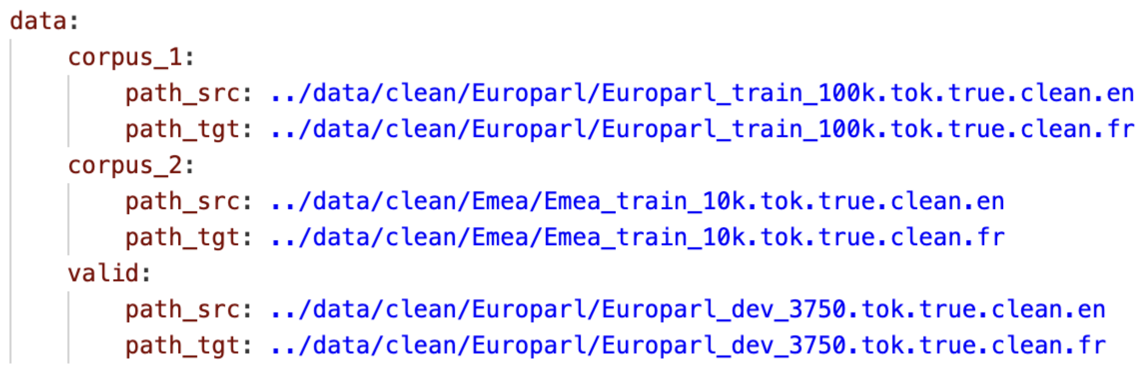
run1: Uses 100,000 sentences from Europarl as the training set, with an additional 3,750 sentences as the validation set.



```
data:
  corpus_1:
    path_src: ../data/clean/Europarl/Europarl_train_100k.tok.true.clean.en
    path_tgt: ../data/clean/Europarl/Europarl_train_100k.tok.true.clean.fr
  valid:
    path_src: ../data/clean/Europarl/Europarl_dev_3750.tok.true.clean.en
    path_tgt: ../data/clean/Europarl/Europarl_dev_3750.tok.true.clean.fr
```

# run2_en_fr.yaml:

run2: Combines 100,000 sentences from the Europarl training set with 10,000 sentences from Emea, used as the training data, and uses 3,750 sentences from Europarl as the validation set.



```yaml
data:
    corpus_1:
        path_src: ../data/clean/Europarl/Europarl_train_100k.tok.true.clean.en
        path_tgt: ../data/clean/Europarl/Europarl_train_100k.tok.true.clean.fr
    corpus_2:
        path_src: ../data/clean/Emea/Emea_train_10k.tok.true.clean.en
        path_tgt: ../data/clean/Emea/Emea_train_10k.tok.true.clean.fr
    valid:
        path_src: ../data/clean/Europarl/Europarl_dev_3750.tok.true.clean.en
        path_tgt: ../data/clean/Europarl/Europarl_dev_3750.tok.true.clean.fr
```

Describe the evaluation metrics: BLEU Score.

The BLEU score is a metric used to evaluate the quality of machine translation by comparing the translated text to one or more reference translations. The score is composed of precision measurements of different lengths of word sequences, known as n-grams.

Firstly, the score assesses how well individual words in the translation match the reference translation, providing an indication of the basic vocabulary accuracy. Next, it examines pairs of consecutive words, evaluating how well the translation captures simple word relationships. The assessment continues with three-word sequences, where the focus shifts to the translation's ability to maintain proper context and sentence flow. Finally, the evaluation includes even longer sequences, assessing the translation's effectiveness in capturing more complex syntactic and semantic structures.

To calculate the BLEU score, we first carry out translation operations using the model. In both run1 and run2, we translate 500 English sentences each from the Europarl and Emea corpora. Once the translations are completed, we use the corresponding 500 French sentences as reference texts for evaluation. The model settings include a training configuration of 10000 steps, a validation frequency of every 8000 steps, and a learning rate of 1, which are configured to optimize performance and accuracy in translation outcomes.

This is achieved by running the multi-bleu.perl script from the Moses toolkit, which compares the machine translation outputs with the reference translations. The script then calculates the BLEU score, allowing us to quantify the accuracy and fluency of the translations.

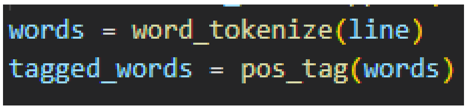
Furthermore, we have calculated BERT scores, which is another method used to evaluate the quality of text generated by machine translation systems. While BLEU primarily focuses on lexical precision by measuring the overlap of n-grams, BERT Scores are more sensitive to the overall meaning and less to exact wording due to its use of contextual embeddings. This can potentially result in higher scores compared to the BLEU score.

Results table.

Based on these two methods, the results are shown in the table in Section 4 below.

4. Evaluation of the OpenNMT neural translation engine on a lemmatised corpus

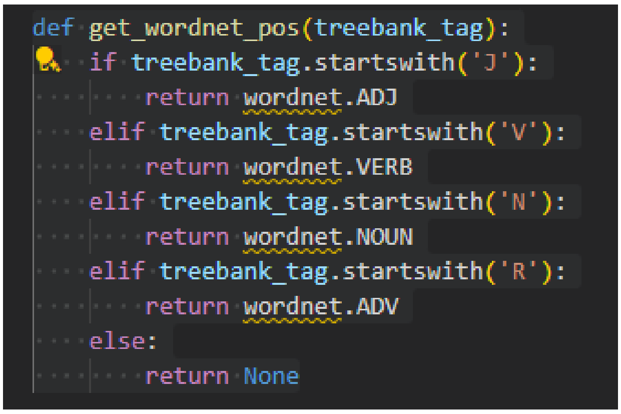
In order to use *WordNetLemmatizer* and *FrenchLefffLemmatizer*, we need to get the pos tag for each word, and this can be achieved with functions from the *nltk* library.



```
words = word_tokenize(line)
tagged_words = pos_tag(words)
```

We use *word_tokenize* to break the text into words and *tagged_words* to get the pos tag for each word.

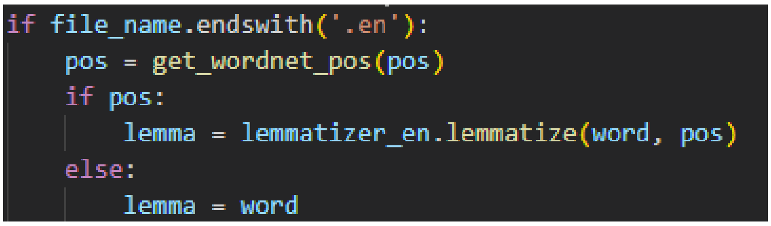
Describing the NLTK lemmatizer for English



```
def get_wordnet_pos(treebank_tag):
    if treebank_tag.startswith('J'):
        return wordnet.ADJ
    elif treebank_tag.startswith('V'):
        return wordnet.VERB
    elif treebank_tag.startswith('N'):
        return wordnet.NOUN
    elif treebank_tag.startswith('R'):
        return wordnet.ADV
    else:
        return None
```

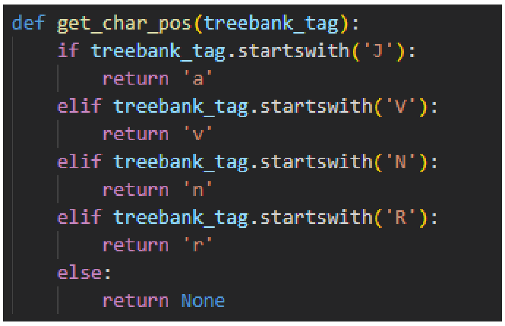
tagged_words gets very detailed attributes for a word, but we only need the first character at the beginning. This is then converted into the *wordnet* form needed by WordNetLemmatizer.

After putting the parameters into the WordNetLemmatizer, we obtain the lemma we need.



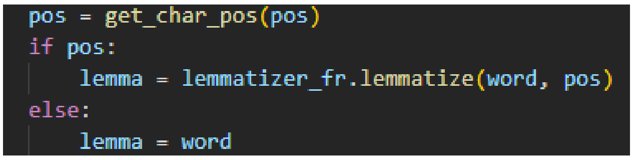
```
if file_name.endswith('.en'):
    pos = get_wordnet_pos(pos)
    if pos:
        lemma = lemmatizer_en.lemmatize(word, pos)
    else:
        lemma = word
```

Describing the NLTK lemmatizer for French



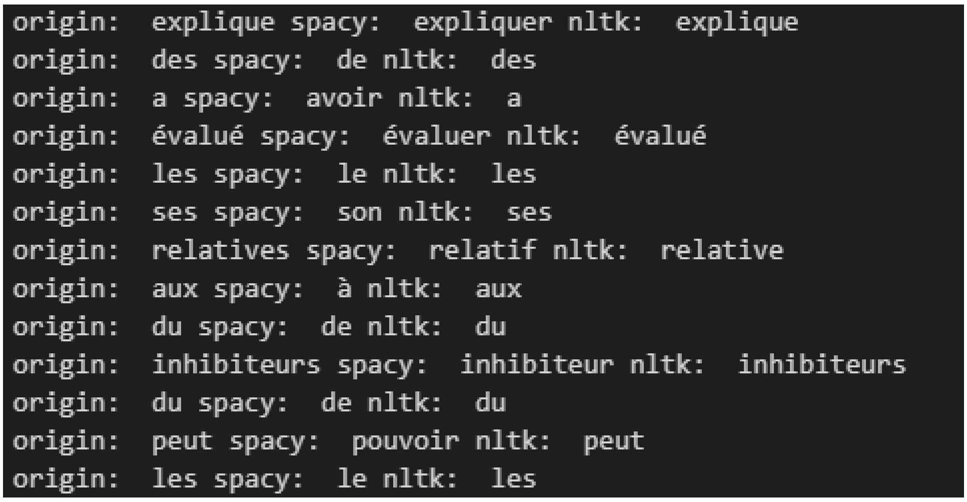
```
def get_char_pos(treebank_tag):
    if treebank_tag.startswith('J'):
        return 'a'
    elif treebank_tag.startswith('V'):
        return 'v'
    elif treebank_tag.startswith('N'):
        return 'n'
    elif treebank_tag.startswith('R'):
        return 'r'
    else:
        return None
```

The processing of French is basically the same as English, except that *FrenchLefffLemmatizer* accepts different forms of parameters than *WordNetLemmatizer*. It needs to convert the first uppercase letter of the tag obtained by tagged_words into the lowercase letter of the corresponding meaning.

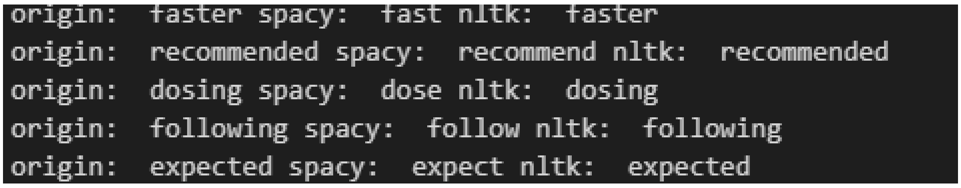


```
pos = get_char_pos(pos)
if pos:
    lemma = lemmatizer_fr.lemmatize(word, pos)
else:
    lemma = word
```

Using spaCy to improve the accuracy of lemmatization



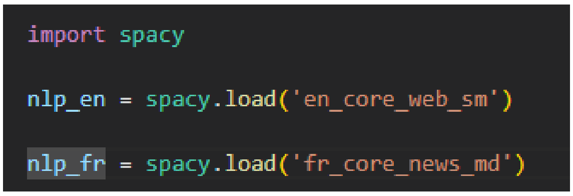
```
origin:  explique spacy:  expliquer nltk:  explique
origin:  des spacy:  de nltk:  des
origin:  a spacy:  avoir nltk:  a
origin:  évalué spacy:  évaluer nltk:  évalué
origin:  les spacy:  le nltk:  les
origin:  ses spacy:  son nltk:  ses
origin:  relatives spacy:  relatif nltk:  relative
origin:  aux spacy:  à nltk:  aux
origin:  du spacy:  de nltk:  du
origin:  inhibiteurs spacy:  inhibiteur nltk:  inhibiteurs
origin:  du spacy:  de nltk:  du
origin:  peut spacy:  pouvoir nltk:  peut
origin:  les spacy:  le nltk:  les
```



```
origin:  faster spacy:  fast nltk:  faster
origin:  recommended spacy:  recommend nltk:  recommended
origin:  dosing spacy:  dose nltk:  dosing
origin:  following spacy:  follow nltk:  following
origin:  expected spacy:  expect nltk:  expected
```

However, we found that *nltk*'s lemmatization does not seem to be very accurate.

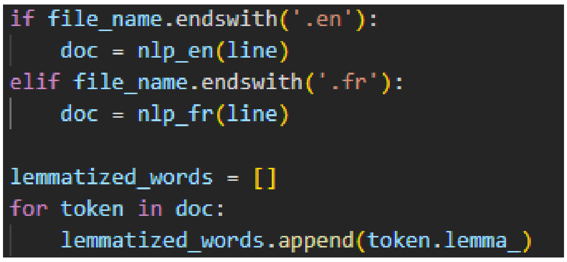
We also lemmatised the words of the text using *spaCy*, and observed that in many cases, *nltk* fails to find the lemma of the word, whereas spaCy has a better performance in lemmatization.



```
import spacy

nlp_en = spacy.load('en_core_web_sm')

nlp_fr = spacy.load('fr_core_news_md')
```



```
if file_name.endswith('.en'):
    doc = nlp_en(line)
elif file_name.endswith('.fr'):
    doc = nlp_fr(line)

lemmatized_words = []
for token in doc:
    lemmatized_words.append(token.lemma_)
```

We therefore turn to *spaCy* for lemmatization. Although it results in a longer running time of the code, it improves the accuracy of the results. Moreover, the code will be neater with better readability.

Results table

-n_sample = 100000
train_steps = 10000
valid_steps = 8000

Model	Train Acc.	Val Acc.	BLEU		F1	
			Europarl	Emea	Europarl	Emea
Run 1 inflected forms	40.59	50.81	0.47	0.57	0.8	0.67
Run 2 inflected forms	38.89	51.83	0.46	0.73	0.79	0.73
Run 1 lemme	45.73	55.81	0.47	0.56	0.77	0.67
Run 2 lemme	44.26	56.24	0.46	0.73	0.76	0.71

5. Strengths, limitations and difficulties encountered

Strengths

The OpenNMT framework is highly versatile, supporting multiple neural network architectures, including LSTM and Transformer models. This flexibility allows for extensive experimentation and optimization.

Limitations

Training neural networks requires significant computational resources, including powerful GPUs and substantial memory.

The quality of the training data has a significant impact on model performance. Inconsistent or noisy data can lead to poor translation quality.

Difficulties

The difficulties mainly come from the instability of OpenNMT and the long time required for training. Since OpenNMT is a closed command, we can't debug the content of the scripts, and can only modify the content of the yaml file. Training the desired model therefore takes a long time and requires repeated attempts. And many times it just pops up error messages that are hard to understand and fix.

6. Organisation

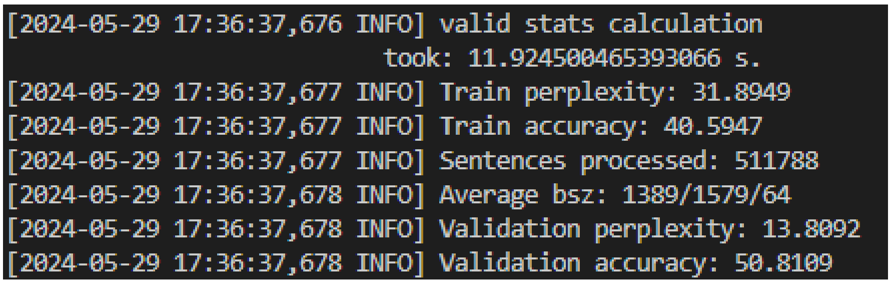
Every member of the group has actively participated in the project, and the results are obtained based on the collaborative work.

The project's github repository was created by Yuntian. For data preparation, Xinlei wrote the code to split the raw data to three parts: train, dev and test, Siman pre-processed the data using mosedecoder, and then the three of us worked together to complete the lemmatization code. Given the large amount of data, Xinlei, who has a MacbookPro M3 and Yuntian, who has a Windows system, are in charge of the training part. For the evaluation part, Xinlei and Yuntian computed the BLEU scores and Siman calculated the BERT score. Finally, Yuntian wrote a notebook to combine the training of the model, and the evaluation on the results directly together using both BLEU and BERT scores.

The report is written by three of us, and each can modify or add comments, which allows for the exchange of ideas, and facilitates our understanding of the project.

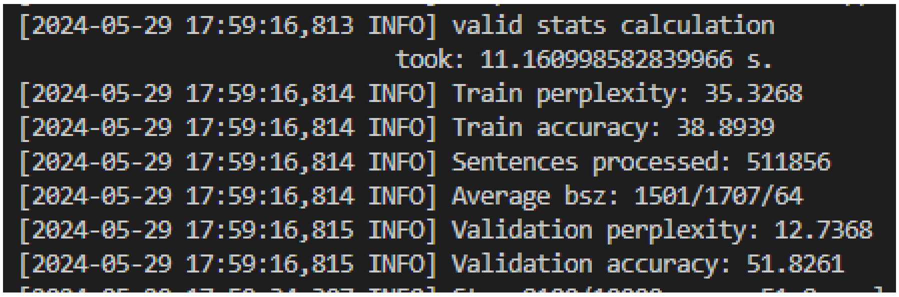
7. Annexes

Screenshots of results



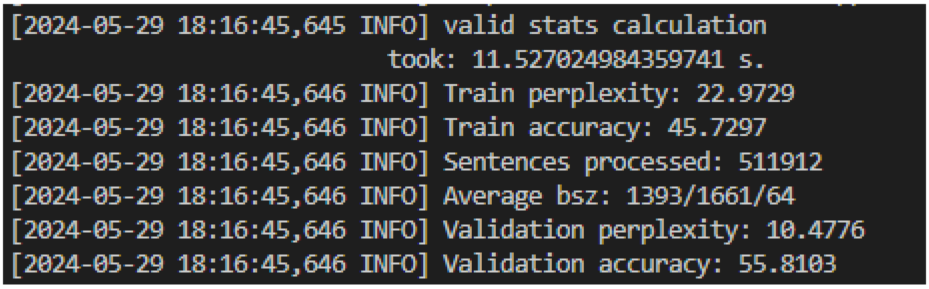
```
[2024-05-29 17:36:37,676 INFO] valid stats calculation
                           took: 11.924500465393066 s.
[2024-05-29 17:36:37,677 INFO] Train perplexity: 31.8949
[2024-05-29 17:36:37,677 INFO] Train accuracy: 40.5947
[2024-05-29 17:36:37,677 INFO] Sentences processed: 511788
[2024-05-29 17:36:37,678 INFO] Average bsz: 1389/1579/64
[2024-05-29 17:36:37,678 INFO] Validation perplexity: 13.8092
[2024-05-29 17:36:37,678 INFO] Validation accuracy: 50.8109
```

Run 1 inflected forms, train_steps = 10000



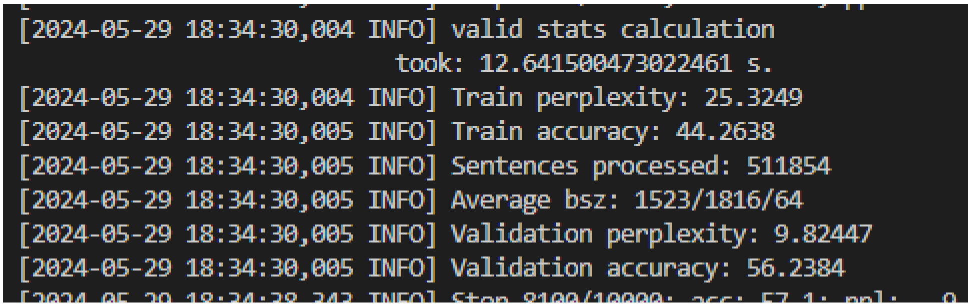
```
[2024-05-29 17:59:16,813 INFO] valid stats calculation
                           took: 11.160998582839966 s.
[2024-05-29 17:59:16,814 INFO] Train perplexity: 35.3268
[2024-05-29 17:59:16,814 INFO] Train accuracy: 38.8939
[2024-05-29 17:59:16,814 INFO] Sentences processed: 511856
[2024-05-29 17:59:16,814 INFO] Average bsz: 1501/1707/64
[2024-05-29 17:59:16,815 INFO] Validation perplexity: 12.7368
[2024-05-29 17:59:16,815 INFO] Validation accuracy: 51.8261
```

Run 2 inflected forms, train_steps = 10000



```
[2024-05-29 18:16:45,645 INFO] valid stats calculation
                           took: 11.527024984359741 s.
[2024-05-29 18:16:45,646 INFO] Train perplexity: 22.9729
[2024-05-29 18:16:45,646 INFO] Train accuracy: 45.7297
[2024-05-29 18:16:45,646 INFO] Sentences processed: 511912
[2024-05-29 18:16:45,646 INFO] Average bsz: 1393/1661/64
[2024-05-29 18:16:45,646 INFO] Validation perplexity: 10.4776
[2024-05-29 18:16:45,646 INFO] Validation accuracy: 55.8103
```

Run 1 lemme, train_steps = 10000



```
[2024-05-29 18:34:30,004 INFO] valid stats calculation
                           took: 12.641500473022461 s.
[2024-05-29 18:34:30,004 INFO] Train perplexity: 25.3249
[2024-05-29 18:34:30,005 INFO] Train accuracy: 44.2638
[2024-05-29 18:34:30,005 INFO] Sentences processed: 511854
[2024-05-29 18:34:30,005 INFO] Average bsz: 1523/1816/64
[2024-05-29 18:34:30,005 INFO] Validation perplexity: 9.82447
[2024-05-29 18:34:30,005 INFO] Validation accuracy: 56.2384
[2024-05-29 18:34:38,343 INFO] Step 8100/10000; acc: 57.1; ppl:    9
```

Run 2 lemme, train_steps = 10000