به نام خدا

دانشگاه تهران

دانشکدگان فنی

دانشکده مهندسی برق و کامپیوتر

درس پردازش زبان طبیعی

پاسخ تمرین شماره پنج

نام و نام خانوادگی: امیرحسین شاه قلی

شماره دانشجویی: ۸۱۰۱۹۹۴۴۱

خرداد ماه ۱۴۰۳

Table of Contents

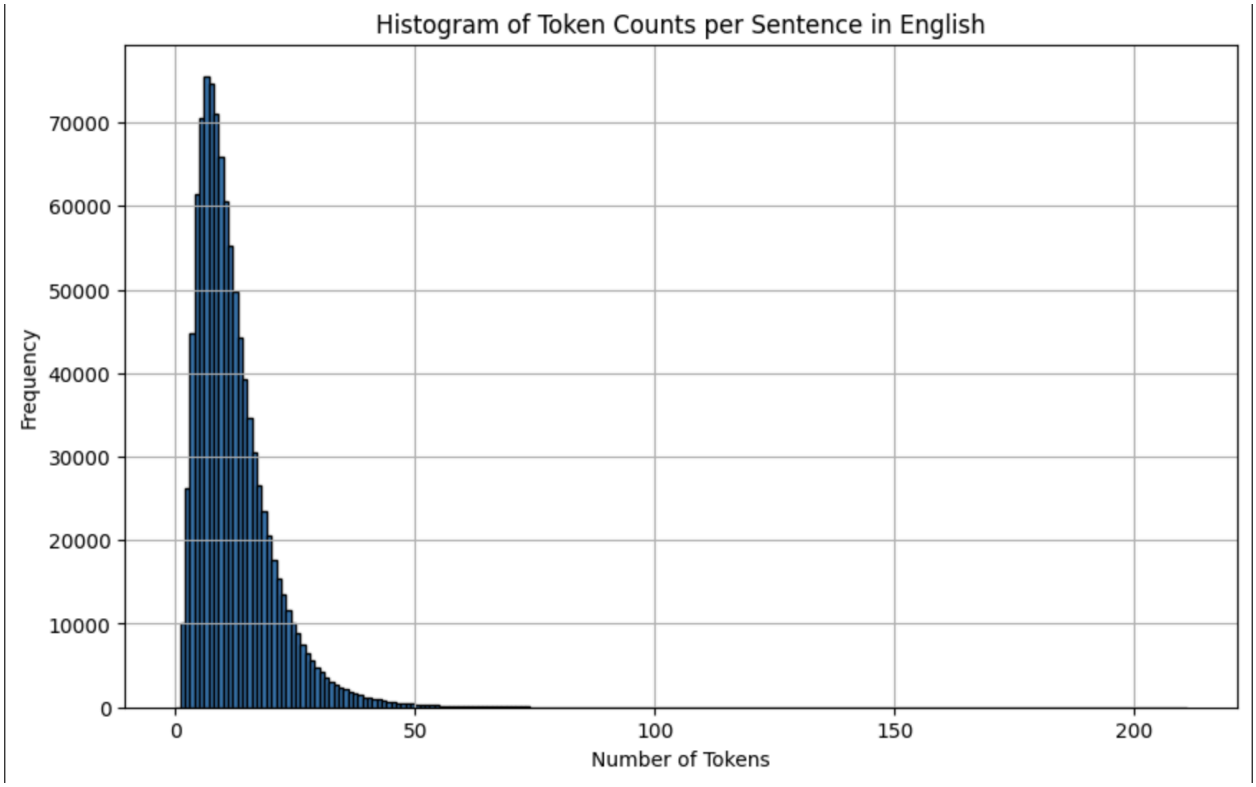
Answer of the question one

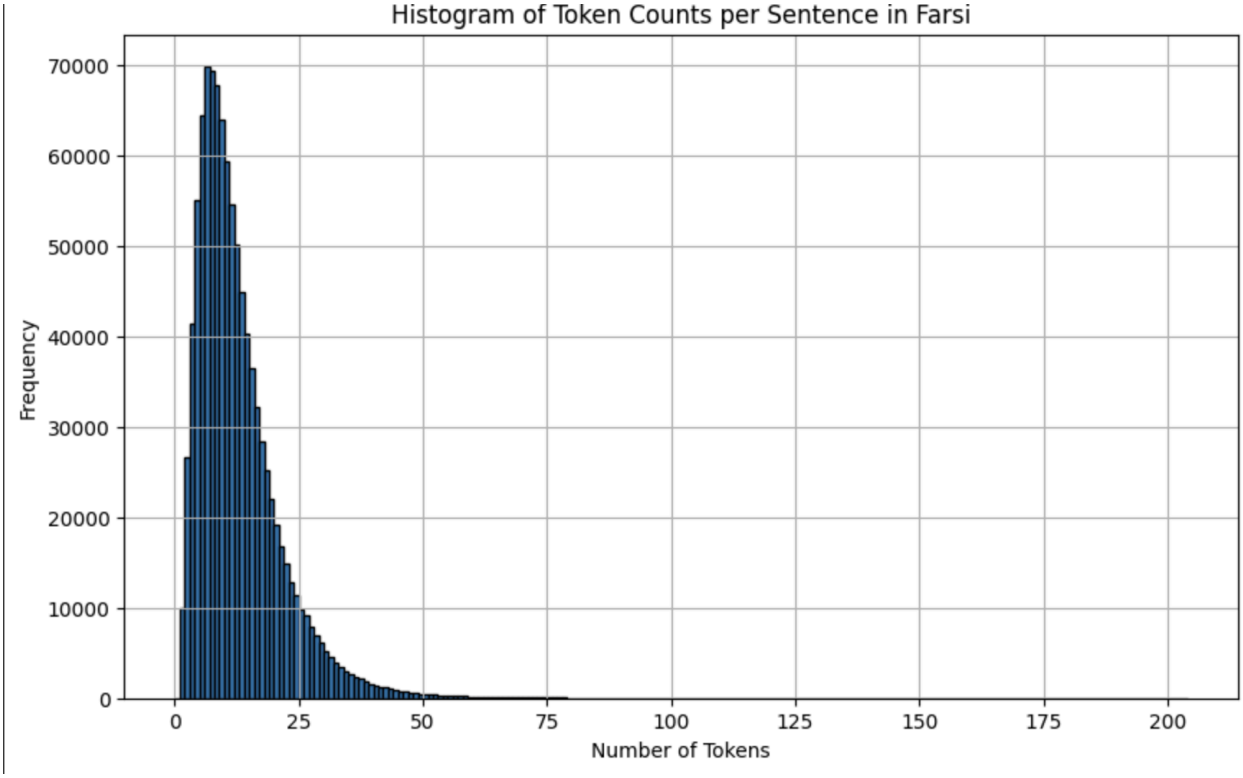
DATASET:

1. Number of lines and first three lines of each files:

```
[6] !wc -l ./raw_data/MIZAN.en-fa.en
    !wc -l ./raw_data/MIZAN.en-fa.fa

1021597 ./raw_data/MIZAN.en-fa.en
1021597 ./raw_data/MIZAN.en-fa.fa

!head ./raw_data/MIZAN.en-fa.en -n 3
!head ./raw_data/MIZAN.en-fa.fa -n 3

The story which follows was first written out in Paris during the Peace Conference
from notes jotted daily on the march, strengthened by some reports sent to my chiefs in Cairo.
Afterwards, in the autumn of 1919, this first draft and some of the notes were lost.
داستانی که از نظر شما می‌گذرد، ابتدا ضمن کنفرانس صلح پاریس از روی یادداشت‌هائی که به طور روزانه در حال خدمت در صف برداشته شده بودند
و از روی گزارشاتی که برای رؤسای من در قاهره ارسال گردیده بودند نوشته شد.
بعدا در پائیز سال 1919، این نوشته اولیه و بعضی از یادداشت‌ها، مفقود شدند.
```

2. Histograms:





3. Remove sentences with less than 10 tokens or more than 50 tokens:

Number of new file lines: 548183

4. Shuffle data and split to tran, test and validation:

```
def shuffle_data(fa_lines, en_lines, seed=42):
   combined = list(zip(fa_lines, en_lines))
   random.seed(seed)
   random.shuffle(combined)
   shuffled_fa_lines, shuffled_en_lines = zip(*combined)
   return shuffled_fa_lines, shuffled_en_lines

shuffled_fa_lines, shuffled_en_lines = shuffle_data(filtered_fa_lines,
filtered_en_lines)


train_size = 500000
```

```
valid_size = 5000
test_size = 1000

train_fa, temp_fa, train_en, temp_en = train_test_split(shuffled_fa_lines,
shuffled_en_lines, train_size=train_size, random_state=42)
valid_fa, test_fa, valid_en, test_en = train_test_split(temp_fa, temp_en,
test_size=test_size, train_size=valid_size, random_state=42)
```

PART ONE: BPE TOKENIZER TRAINING AND DATA PREPROCESSING

fairseq-preprocess is a command-line utility in the Fairseq toolkit that is used for preprocessing raw text data into a binary format that is more efficient for training neural machine translation models. Preprocessing includes tokenization, vocabulary creation, and binarization of data, among other steps. Here's a general explanation of what happens during this process and what files are generated:

What Happens During fairseq-preprocess:

- **Tokenization:** The raw text data is tokenized into subwords, sentences, or whatever unit is defined for the preprocessing. Often, tools like SentencePiece or Moses are used for tokenization before running fairseq-preprocess.
- **Vocabulary Creation:** The tokenized text is used to create a vocabulary file that lists the tokens and maps them to numeric indices. Often, a maximum vocabulary size is defined to limit the number of tokens.
- **Binarization:** The tokenized text and the vocabulary are converted into binary files that make data handling more efficient during training. Each token in the text is replaced by its corresponding index from the vocabulary file.
- **Dataset Splitting:** Data is typically split into different subsets for training, validation, and testing. Each subset (e.g., train, valid, test) undergoes the same preprocessing steps.

Here is an example command for fairseq-preprocess:

```
fairseq-preprocess --source-lang src --target-lang tgt \ --trainpref data/train
--validpref data/valid --testpref data/test \ --destdir data-bin --workers 20
```

--source-lang src and --target-lang tgt: Specify the source and target language codes.

--trainpref data/train: Path prefix to train data files. Fairseq will look for data/train.src and data/train.tgt.

--validpref data/valid: Path prefix to validation data files.

--testpref data/test: Path prefix to test data files.

--destdir data-bin: The directory where the processed binary files will be stored.

--workers 20: Number of worker threads to use for preprocessing.

After running fairseq-preprocess, the following types of files are typically generated:

- **Vocabulary Files:**
    - dict.src.txt: Vocabulary file for the source language.
    - dict.tgt.txt: Vocabulary file for the target language.
- **Training Data:**
    - train.src-bin.idx and train.src-bin.bin: Binary and index files for the source language training data.
    - train.tgt-bin.idx and train.tgt-bin.bin: Binary and index files for the target language training data.
- **Validation Data:**
    - valid.src-bin.idx and valid.src-bin.bin: Binary and index files for the source language validation data.
    - valid.tgt-bin.idx and valid.tgt-bin.bin: Binary and index files for the target language validation data.
- **Test Data:**
    - test.src-bin.idx and test.src-bin.bin: Binary and index files for the source language test data.
    - test.tgt-bin.idx and test.tgt-bin.bin: Binary and index files for the target language test data.

These binary files contain the preprocessed data in a format that Fairseq can efficiently load during training, validation, and testing.

PART TWO: LSTM ENCODER-DECODER MODEL TRAINING

Train Command:

```
!fairseq-train \
   "./data_bin/" \
   --arch lstm --encoder-layers 6 --decoder-layers 6 \
   --share-decoder-input-output-embed \
   --optimizer adam --adam-betas '(0.9,0.98)' --clip-norm 0.0 \
   --lr 2.5e-3 --lr-scheduler inverse_sqrt --warmup-updates 4000 \
   --dropout 0.25 --weight-decay 0.0001 \
   --criterion label_smoothed_cross_entropy --label-smoothing 0.2 \
   --max-tokens 4096 \
   --eval-bleu \
   --eval-bleu-args '{"beam": 5, "max_len_a": 1.2, "max_len_b": 10}' \
   --eval-bleu-detok moses \
   --eval-bleu-print-samples \
   --best-checkpoint-metric bleu --maximize-best-checkpoint-metric \
   --max-source-positions 1024 --max-target-positions 1024 \
   --fp16 --memory-efficient-fp16 \
   --max-epoch 5 \
   --save-dir ./data_bin/checkpoints/ \
   --tensorboard-logdir ./data_bin/log/
```

Explain the use of the two parameters --batch-size and --max-tokens. Report the used values and explain how these two parameters better control the size of each batch:

- **--batch-size:** This parameter sets the number of sequences to be included in each batch. When using --batch-size, the batch is created by selecting a fixed number of sequences, regardless of

their length. Not specified in the command, meaning it defaults to Fairseq's internal settings or is managed dynamically.

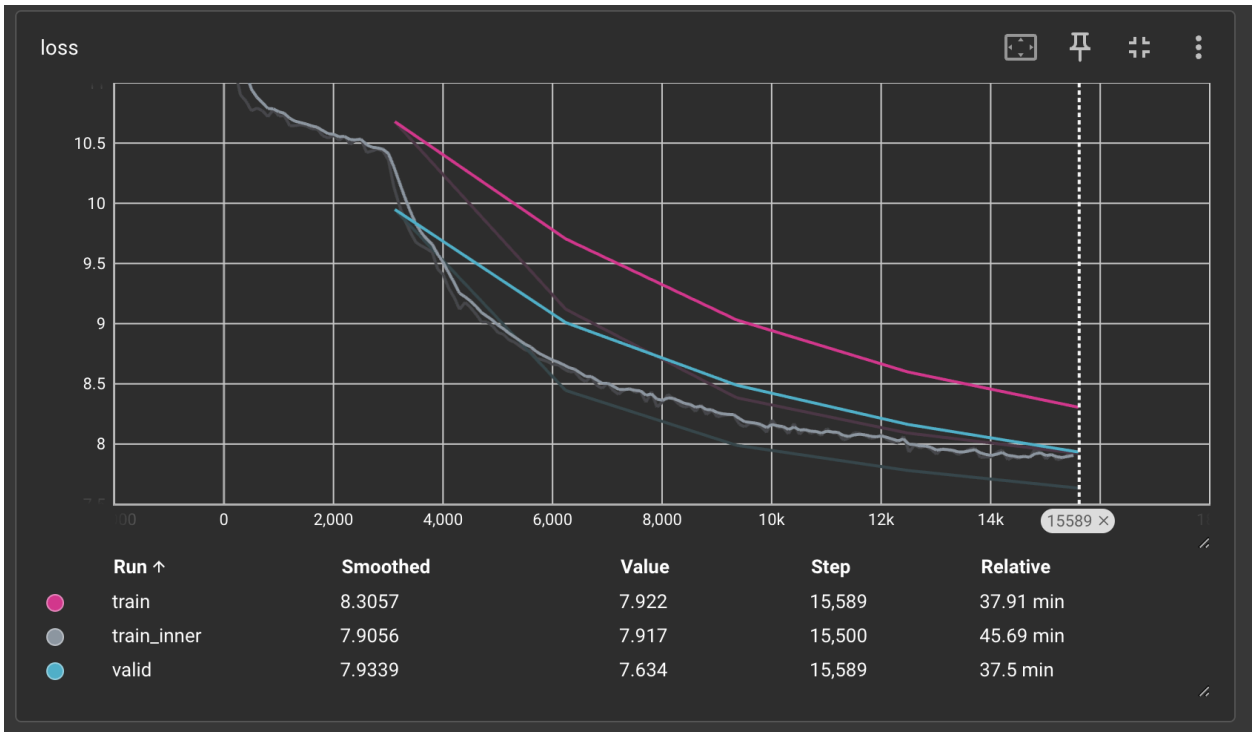
- **--max-tokens:** This parameter sets the maximum number of tokens in a batch. The total number of tokens in a batch will not exceed this value, making the size of each batch more adaptive to different sequence lengths. In the command set as 4096.

How These Parameters Control Batch Size:

Using **--max-tokens** instead of **--batch-size** provides better control over memory usage and computational efficiency, especially when dealing with sequences of varying lengths. With **--max-tokens** 4096, the total number of tokens in each batch remains relatively constant, leading to more predictable GPU memory usage and potentially shorter training times.

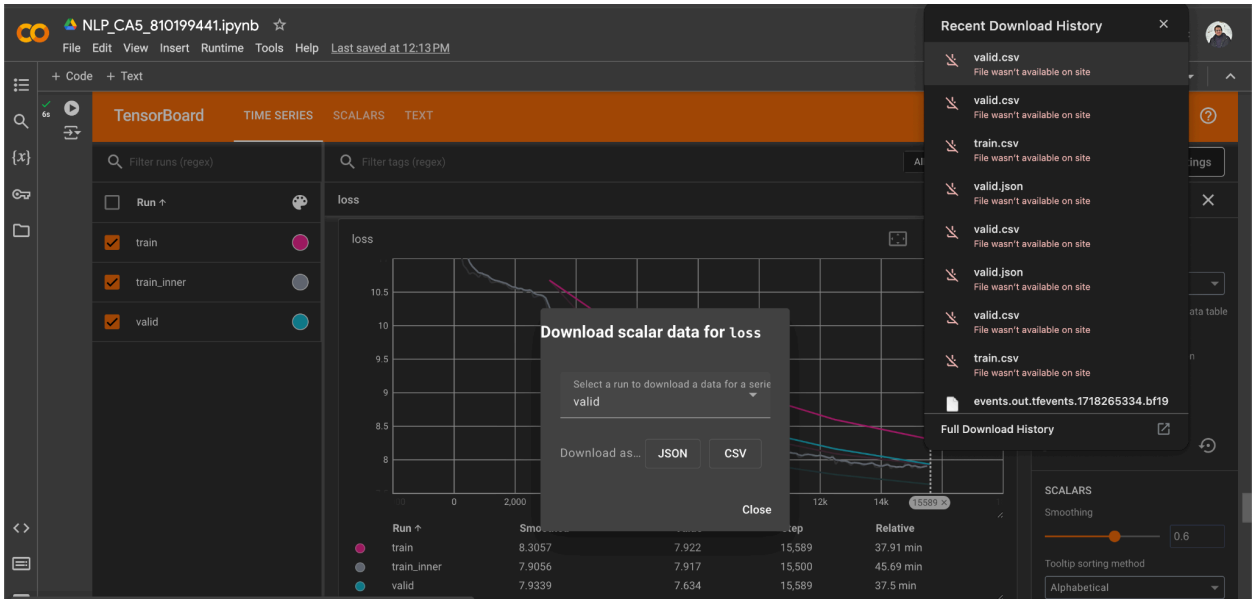
By setting **--max-tokens**, we allow the model to dynamically adjust the number of sentences in a batch based on their length, ensuring that the total token count in a batch does not exceed the specified limit. This is particularly useful for neural machine translation tasks where sentence lengths can vary significantly.

Loss Chart:



Run ↑	Smoothed	Value	Step	Relative
train	8.3057	7.922	15,589	37.91 min
train_inner	7.9056	7.917	15,500	45.69 min
valid	7.9339	7.634	15,589	37.5 min

NOTICE: The csv file can not be downloaded

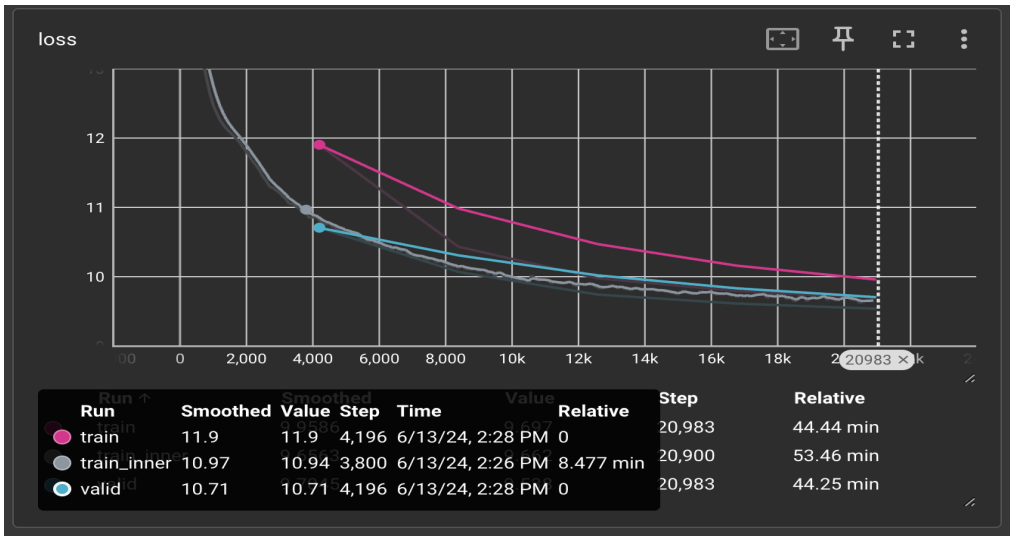


PART THREE: TRANSFORMERS ENCODER–DECODER MODEL TRAINING

Train Command:

```
!fairseq-train \
   "./data_bin/" \
   --arch transformer --encoder-layers 6 --decoder-layers 6 \
   --share-decoder-input-output-embed \
   --optimizer adam --adam-betas '(0.9, 0.98)' --clip-norm 1.0 \
   --lr 2e-5 --lr-scheduler inverse_sqrt --warmup-updates 8000 \
   --dropout 0.3 --weight-decay 0.00001 \
   --criterion label_smoothed_cross_entropy --label-smoothing 0.2 \
   --max-tokens 4096 \
   --eval-bleu \
   --eval-bleu-args '{"beam": 5, "max_len_a": 1.2, "max_len_b": 10}' \
   --eval-bleu-detok moses \
   --eval-bleu-print-samples \
   --best-checkpoint-metric bleu --maximize-best-checkpoint-metric \
   --max-source-positions 1024 --max-target-positions 1024 \
   --fp16 --memory-efficient-fp16 \
   --min-loss-scale 1.0 \
   --max-epoch 5 \
   --batch-size 128 \
   --save-dir ./data_bin/transformer_checkpoints/ \
   --tensorboard-logdir ./data_bin/transformer_log/
```

Loss Chart:



PART FOUR: EVALUATION AND REVIEW OF TEST DATA

LSTM results: `Generate test with beam=5: BLEU4 = 5.04, 33.1/8.7/3.1/1.2 (BP=0.885, ratio=0.891, syslen=19480, reflen=21865)`

COMET score for LSTM: 0.7147174855768681

Transformer results: `Generate test with beam=5: BLEU4 = 0.88, 19.0/2.2/0.3/0.1 (BP=1.000, ratio=1.105, syslen=24171, reflen=21865)`

Average COMET score for Transformer: `0.6103741136789322`

COMET, which stands for Crosslingual Optimized Metric for Evaluation of Translation, is a metric used for evaluating machine translation (MT) models. Developed by Unbabel, COMET is based on advanced machine learning techniques and designed to provide a more accurate and nuanced assessment of translation quality compared to previous metrics.

Key Features of COMET:

1. Model Type: COMET belongs to a family of metric systems known as learned metrics, which differ significantly from traditional automatic evaluation metrics like BLEU (Bilingual Evaluation Understudy), METEOR, or TER (Translation Edit Rate). Learned metrics utilize machine learning models trained on human judgments to evaluate translations.

2. Training: COMET is trained on large datasets that consist of source texts, reference translations, and human evaluations. The human evaluations help the model learn the nuances of quality translation, including fluity, adequacy, and other linguistic factors that simpler metrics might overlook.

3. Embeddings and Contextual Understanding: COMET leverages transformer-based models (such as XLM-R) to generate embeddings for translated text and reference text. These embeddings capture deep semantic and contextual information, allowing the model to assess translations on nuanced levels beyond mere lexical matching.

4. Crosslingual Performance: As suggested by its name, COMET performs well across various languages. Its training on diverse language pairs using transformer-based architectures helps it maintain robustness and adaptability across language barriers.

5. Alignment with Human Judgment: One of the significant advantages of COMET over traditional metrics is its superior correlation with human judgments of translation quality. Because it is trained

directly on human evaluations, it tends to align more closely with how human translators would rate translations.

Compare Results:

BLEU Score Analysis: The LSTM model achieved a BLEU score of 5.04 with individual n-gram scores of 33.1/8.7/3.1/1.2, while the Transformer model only reached a BLEU score of 0.88 with n-gram scores of 19.0/2.2/0.3/0.1.

1. **N-gram Precision:**
    - The higher scores in the 1-gram and 2-gram precisions for the LSTM indicate better handling of more common, perhaps simpler phrasal structures compared to the Transformer. This could imply better general translation quality on frequently seen words and phrases.
    - The LSTM also performs better in the longer 3-gram and 4-gram contexts, suggesting it can maintain more coherent and contextually appropriate translations over longer text spans than the Transformer.
2. **Brevity Penalty (BP):**
    - The LSTM has a BP of 0.885, showing that its translations are slightly shorter than the reference length but still near optimal length, while the Transformer's BP of 1.000 indicates that its translations are longer than the references. The longer length in Transformer's output doesn't contribute positively to translation quality, given the low BLEU score.
3. **Length Ratio:**
    - The LSTM outputs maintain a closer proportional length (ratio = 0.891) to the reference texts compared to the Transformer's outputs which exceed the reference length (ratio = 1.105). This suggests that while the Transformer generates longer text, it's likely adding noise or irrelevant information.

COMET Score Analysis:

COMET, a more recent machine translation evaluation metric that considers factors beyond n-gram matching, such as fluency and adequacy more holistically, also suggests a higher score for the LSTM model (0.7147) compared to the Transformer (0.6104). This supports the conclusion that the LSTM model produces translations that are generally more adequate, fluent, and closer to human translations than those generated by the Transformer.

Conclusion:

From the evaluation results, it's clear that in this setup, the LSTM-based model surpasses the Transformer-based model in terms of both BLEU and COMET scores. Despite Transformers generally being renowned for their effectiveness in many NLP tasks including translation, in this instance, the LSTM model has provided more coherent, contextually appropriate, and higher-quality translations. This could potentially be due to various factors including differences in training data, parameter tuning, model architecture suitability to the specific task, or training duration and conditions. The results underscore the importance of considering various model options and extensively tuning them according to specific task requirements and data characteristics.