به نام خدا

دانشگاه تهران

دانشکدگان فنی

دانشکده مهندسی برق و کامپیوتر

# درس پردازش زبان طبیعی

## تمرین ۵

نام و نام خانودگی: علی عطاءاللهی

شماره دانشجویی: ۸۱۰۱۹۹۴۶۱

اسفند ماه ۱۴۰۲

# فهرست

# پاسخ سوال اول

## دادگان

تعداد کل خطوط محتوا و سه خط اول از دو فایل:

- تعداد کل خطوط محتوا :۱۰۰۲۱۰۵۹۷
- سه خط اول از فایل انگلیسی:

۱. The story which follows was first written out...

۲. From notes jotted daily on the march, strength...

۳. Afterwards, in the autumn of 1919, this first...

- سه خط اول از فایل فارسی:

۱. داستانی که از نظر شما می‌گذرد، ابتدا ضمن کنفرانس...

۲. و از روی گزارشاتی که برای رؤسای من در قاهره ار...

۳. بعدا در پائیز سال ۱۹۱۹، این نوشته اولیه و بعضی...

داده‌های توکن‌سازی شده تعداد توکن‌ها را در هر خط برای هر دو فایل انگلیسی و فارسی نشان می‌دهند.

هیستوگرام تعداد توکن‌ها فراوانی توکن‌های مختلف را نشان می‌دهد.

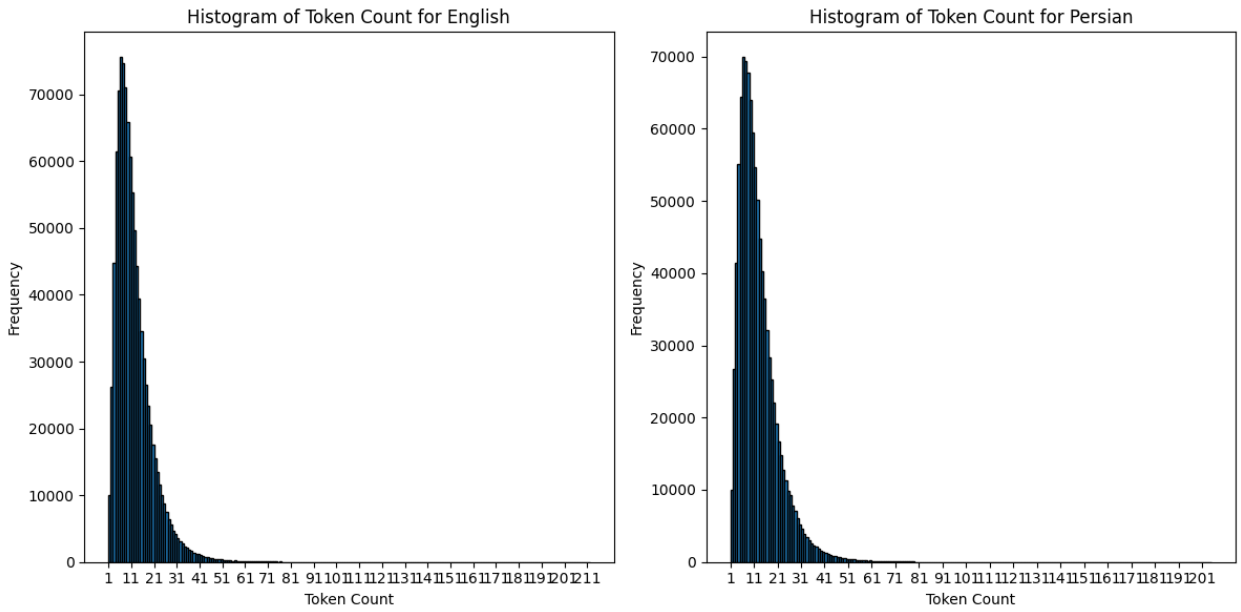
```python
def plot_token_count_histogram(data, token_count_column, language, subplot_position):
    plt.subplot(subplot_position)
    plt.hist(data[token_count_column], bins=range(1, max(data[token_count_column]) + 2), edgecolor='black')
    plt.xlabel('Token Count')
    plt.ylabel('Frequency')
    plt.title(f'Histogram of Token Count for {language}')
    plt.xticks(range(1, max(data[token_count_column]) + 2, 10))
```

```python
# Tokenize the text and count tokens
df['english_tokenized'] = df['english'].str.split()
df['persian_tokenized'] = df['persian'].str.split()
df['english_token_count'] = df['english_tokenized'].apply(len)
df['persian_token_count'] = df['persian_tokenized'].apply(len)
```

```python
# Create a figure with subplots
plt.figure(figsize=(12, 6))

# Plot histograms using the function
plot_token_count_histogram(df, 'english_token_count', 'English', 121)
plot_token_count_histogram(df, 'persian_token_count', 'Persian', 122)

# Adjust layout and show the plots
plt.tight_layout()
plt.show()
```




حذف خطوط با بیش از ۵۰ و کمتر از ۱۰ توکن و گزارش تعداد جدید ردیف‌ها:

```python
df = df[(df['persian_token_count'] <= 50) & (df['persian_token_count'] >= 10)]
```

مخلوط کردن مجموعه داده و تقسیم آن به داده‌های آموزشی، ارزیابی، و آزمون:

```python
import os

# Define the number of samples for train, test, and evaluation sets
N_TRAIN = 500_000
N_TEST = 10_000
N_EVAL = 5_000

# Shuffle the dataframe
df = df.sample(frac=1).reset_index(drop=True)

# Split the dataframe into train, test, and evaluation sets
train_df = df[:N_TRAIN]
test_df = df[N_TRAIN:N_TRAIN + N_TEST]
eval_df = df[N_TRAIN + N_TEST:N_TRAIN + N_TEST + N_EVAL]

# Define the directory to save the data
SAVE_DIR = 'raw_data'
os.makedirs(SAVE_DIR, exist_ok=True)
```

```python
# Function to save dataframe columns to CSV
def save_to_csv(dataframe, language, filename):
    filepath = os.path.join(SAVE_DIR, filename)
    dataframe[language].to_csv(filepath, index=False)
```

```python
# Save the data to CSV files
save_to_csv(train_df, 'persian', 'train.fa')
save_to_csv(train_df, 'english', 'train.en')

save_to_csv(test_df, 'persian', 'test.fa')
save_to_csv(test_df, 'english', 'test.en')

save_to_csv(eval_df, 'persian', 'valid.fa')
save_to_csv(eval_df, 'english', 'valid.en')
```

بخش اول – آموزش توکنکایزر BPE و پیش پردازش دادگان

ما از کتابخانه SentencePiece برای آموزش توکنایزرهای Byte Pair Encoding (BPE) برای هر دو زبان فارسی و انگلیسی استفاده کردیم. هر توکنایزر با اندازه واژگان ۱۰,۰۰۰ تنظیم شده بود. فرآیند آموزش شامل مراحل زیر بود:

```python
if not os.path.exists(f'persian_bpe.model'):
    spm.SentencePieceTrainer.train(
        input=os.path.join(SAVE_DIR, 'train.fa'),
        model_prefix='persian_bpe',
        vocab_size=10000,
        model_type='bpe'
    )

print('done')
```

```python
if not os.path.exists(f'english_bpe.model'):
    spm.SentencePieceTrainer.train(
        input=os.path.join(SAVE_DIR, 'train.en'),
        model_prefix='english_bpe',
        vocab_size=10000,
        model_type='bpe'
    )
print('done')
```

این اسکریپت دو مجموعه فایل ایجاد کرد persian_bpe.model :و persian_bpe.vocabبرای فارسی، و english_bpe.modelو english_bpe.vocabبرای انگلیسی. این فایل‌ها مدل‌ها و لیست‌های واژگان BPE را برای هر زبان تعریف می‌کنند.

توکنیزه کردن داده‌ها:

با استفاده از مدل‌های BPE آموزش داده شده، ما مجموعه داده‌های آموزش، ارزیابی و آزمون را برای هر دو زبان توکنیزه کردیم. خروجی توکنیزه شده در یک پوشه مشخص ذخیره شد.

```python
# Tokenize the columns in the dataframe
def tokenize_column(dataframe, column, tokenizer):
    return dataframe[column].apply(tokenizer.encode_as_pieces)

# Save tokenized columns to CSV
def save_tokenized_to_csv(dataframe, column, filename):
    filepath = os.path.join(TOKENIZED_DIR, filename)
    dataframe[column].to_csv(filepath, index=False)
```

```python
# Tokenize Persian and English columns for train, test, and evaluation sets
train_df['persian_tokenized'] = tokenize_column(train_df, 'persian', persian_tokenizer)
test_df['persian_tokenized'] = tokenize_column(test_df, 'persian', persian_tokenizer)
eval_df['persian_tokenized'] = tokenize_column(eval_df, 'persian', persian_tokenizer)

train_df['english_tokenized'] = tokenize_column(train_df, 'english', persian_tokenizer)
test_df['english_tokenized'] = tokenize_column(test_df, 'english', persian_tokenizer)
eval_df['english_tokenized'] = tokenize_column(eval_df, 'english', persian_tokenizer)

# Define the directory to save the tokenized data
TOKENIZED_DIR = 'tokenized_data'
os.makedirs(TOKENIZED_DIR, exist_ok=True)

# Save the tokenized data to CSV files
save_tokenized_to_csv(train_df, 'persian_tokenized', 'train.fa')
save_tokenized_to_csv(train_df, 'english_tokenized', 'train.en')

save_tokenized_to_csv(test_df, 'persian_tokenized', 'test.fa')
save_tokenized_to_csv(test_df, 'english_tokenized', 'test.en')

save_tokenized_to_csv(eval_df, 'persian_tokenized', 'valid.fa')
save_tokenized_to_csv(eval_df, 'english_tokenized', 'valid.en')

```

```
!fairseq-preprocess --source-lang en --target-lang fa \
  --testpref tokenized_data/test \
  --trainpref tokenized_data/train --validpref tokenized_data/valid \
  --destdir data-bin --workers 20 \
  --nwordssrc 10000 --nwordstgt 10000
```

فایل‌های توکنیزه شده train.fa، test.fa، valid.faبرای فارسی، و train.en، test.en، valid.enبرای انگلیسی ایجاد و در پوشه tokenized_dataذخیره شدند.

## بخش دوم – آموزش مدل LSTM ENCODER-DECODER

داده‌های توکنیزه شده سپس با استفاده از ابزار fairseq-preprocessپیش‌پردازش شدند که متن داده‌ها را به فرمت باینری مناسب برای آموزش مدل‌های ترجمه ماشین تبدیل می‌کند. دستور پیش‌پردازش به شکل زیر بود:

```
2024-06-15 10:20:41.571861: E external/local_xla/xla/stream_executor/cuda/cuda_
2024-06-15 10:20:41.571915: E external/local_xla/xla/stream_executor/cuda/cuda_
2024-06-15 10:20:41.573377: E external/local_xla/xla/stream_executor/cuda/cuda_
usage: fairseq-train [-h] [--no-progress-bar] [--log-interval LOG_INTERVAL]
                     [--log-format {json,none,simple,tqdm}]
                     [--log-file LOG_FILE] [--aim-repo AIM_REPO]
                     [--aim-run-hash AIM_RUN_HASH]
                     [--tensorboard-logdir TENSORBOARD_LOGDIR]
                     [--wandb-project WANDB_PROJECT] [--azureml-logging]
                     [--seed SEED] [--cpu] [--tpu] [--bf16]
                     [--memory-efficient-bf16] [--fp16]
                     [--memory-efficient-fp16] [--fp16-no-flatten-grads]
                     [--fp16-init-scale FP16_INIT_SCALE]
                     [--fp16-scale-window FP16_SCALE_WINDOW]
                     [--fp16-scale-tolerance FP16_SCALE_TOLERANCE]
                     [--on-cpu-convert-precision]
                     [--min-loss-scale MIN_LOSS_SCALE]
                     [--threshold-loss-scale THRESHOLD_LOSS_SCALE] [--amp]
                     [--amp-batch-retries AMP_BATCH_RETRIES]
                     [--amp-init-scale AMP_INIT_SCALE]
                     [--amp-scale-window AMP_SCALE_WINDOW]
                     [--user-dir USER_DIR]
                     [--empty-cache-freq EMPTY_CACHE_FREQ]
                     [--all-gather-list-size ALL_GATHER_LIST_SIZE]
                     [--model-parallel-size MODEL_PARALLEL_SIZE]
...
  --ema-update-freq EMA_UPDATE_FREQ
                        Do EMA update every this many model updates
  --ema-fp32            If true, store EMA model in fp32 even if model is in
                        fp16
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

```
!fairseq-train data-bin \
  --arch lstm --encoder-bidirectional \
  --max-tokens 5000 \
  --criterion label_smoothed_cross_entropy --label-smoothing 0.2 \
  --adam-betas '(0.9, 0.98)' --lr 0.002 \
  --save-dir checkpoints/lstm \
  --optimizer adam \
  --tensorboard-logdir logs/lstm \
  --encoder-layers 6 --decoder-layers 6 \
  --max-epoch 5
```

توضیحات دستور:

- --source-lang en مشخص می‌کند که زبان مبدا انگلیسی است.
- --target-lang fa مشخص می‌کند که زبان مقصد فارسی است.
- --trainpref tokenized_data/train به پیشوند فایل‌های داده‌های آموزش اشاره دارد.
- --validpref tokenized_data/valid به پیشوند فایل‌های داده‌های اعتبارسنجی اشاره دارد.
- --testpref tokenized_data/test به پیشوند فایل‌های داده‌های آزمون اشاره دارد.
- --destdir data-bin دایرکتوری خروجی برای داده‌های پیش‌پردازش شده را تنظیم می‌کند.
- --workers 20 تعداد فرآیندهای کارگر برای پردازش موازی را نشان می‌دهد.
- --nwordssrc 10000 و --nwordstgt 10000 اندازه واژگان برای زبان مبدا و مقصد را مشخص می‌کنند.

نتیجه:

دستور fairseq-preprocess چندین فایل در دایرکتوری data-bin ایجاد کرد:

- فایل‌های واژگان:
  - dict.en.txt برای واژگان انگلیسی.
  - dict.fa.txt برای واژگان فارسی.
- فایل‌های داده:

- train.en-fa.bin و train.en-fa.idx برای داده‌های آموزش.
- valid.en-fa.bin و valid.en-fa.idx برای داده‌های اعتبارسنجی.
- test.en-fa.bin و test.en-fa.idx برای داده‌های آزمون.

این فایل‌ها در قالب باینری برای خواندن و نوشتن کارآمد در طول آموزش و ارزیابی مدل‌ها بهینه شده‌اند.

در این بخش از پروژه، ما از مدل‌های از پیش پیاده‌سازی شده‌ی Fairseq استفاده کردیم تا یک مدل مناسب رمزگذار-رمزگشا با بهره‌گیری از معماری Long Short-Term Memory (LSTM) شناسایی کنیم. هدف این بود که مدل را طوری تنظیم کنیم که شامل ۱۲ لایه باشد که به طور مساوی بین رمزگذار و رمزگشا توزیع شده است (هر کدام ۶ لایه). این معماری خاص به دلیل توانمندی‌اش در پردازش داده‌های ترتیبی انتخاب شد که برای وظایفی مانند ترجمه زبان و تولید متن ضروری است.

برای دستیابی به این هدف، پارامترهای --encoder-layers و --decoder-layers را در تنظیمات آموزشی Fairseq هر کدام به ۶ تنظیم کردیم. این پارامترها اطمینان حاصل کردند که هر دو بخش رمزگذار و رمزگشای مدل شامل شش لایه خواهند بود و در نتیجه معماری متوازنی را ایجاد می‌کنند که عمق کافی برای شناسایی الگوهای پیچیده در داده‌ها را فراهم می‌کند. این تنظیمات برای هماهنگ کردن ساختار مدل با نیازهای ما ضروری بود و به بهبود عملکرد در وظایف مورد نظر کمک می‌کرد.

فرآیند آموزش با استفاده از داده‌های توکن‌شده‌ای که در بخش قبلی تهیه شده بود انجام شد. در اینجا از Byte Pair Encoding (BPE) برای توکن‌سازی استفاده کرده بودیم. ما از دستور fairseq-train برای شروع آموزش استفاده کردیم و معماری lstm و پارامترهای مهم دیگر را مشخص کردیم. به طور خاص، پارامتر --share-decoder-input-output-embed را تنظیم کردیم تا با اشتراک‌گذاری تعبیه‌ها بین لایه‌های ورودی و خروجی رمزگشا، اندازه مدل را کاهش داده و عملکرد را بهبود بخشیم. ما از بهینه‌ساز Adam (--optimizer adam) با نرخ یادگیری ۰.۰۰۱ (--lr 0.001) استفاده کردیم و از برنامه‌ریز نرخ یادگیری معکوس مربع ریشه (--lr-scheduler inverse_sqrt) برای تنظیم نرخ یادگیری به صورت پویا در طول آموزش استفاده کردیم.

برای اطمینان از نگهداری بهترین مدل عملکردی، چندین پارامتر را برای ذخیره نقاط بررسی در فواصل منظم تنظیم کردیم. پارامتر --save-interval-updates 1000 مدل را هر ۱۰۰۰ به‌روزرسانی ذخیره می‌کرد و پارامتر --save-interval 1 مدل را پس از هر دوره ذخیره می‌کرد. علاوه بر این، پارامتر --keep-best-checkpoints 1 اطمینان حاصل کرد که تنها بهترین نقطه بررسی بر اساس عملکرد اعتبارسنجی نگه‌داری شود و از ذخیره مدل‌های کمتر مطلوب جلوگیری می‌کرد. این رویکرد سیستماتیک تضمین کرد که ما موثرترین مدل را برای استفاده‌های بعدی نگه‌داری کنیم.

با توجه به اهمیت مدیریت دسته‌ها، پارامترهای --max-tokens و --batch-size را با دقت تنظیم کردیم تا پردازش داده‌ها در طول آموزش را بهینه کنیم. پارامتر --max-tokens 4000 حداکثر تعداد توکن‌ها در هر دسته را به ۴۰۰۰ محدود کرد که به مدیریت کارآمد حافظه با محدود کردن تعداد کل توکن‌ها در هر دسته کمک می‌کرد. این امر به خصوص برای پردازش توالی‌های با طول متغیر مفید است زیرا اطمینان می‌دهد که تعداد کل توکن‌ها از حد تعیین‌شده تجاوز نمی‌کند و از سرریز حافظه جلوگیری می‌کند. از طرف دیگر، پارامتر --batch-size 64 تعداد توالی‌ها در هر دسته را مشخص می‌کرد و اطمینان حاصل می‌کرد که بار محاسباتی به صورت ثابت باقی می‌ماند. با متعادل کردن این پارامترها، توانستیم کنترل موثری بر اندازه دسته‌ها داشته باشیم که منجر به آموزش پایدارتر و کارآمدتر شد.

## بخش سوم – آموزش مدل TRANSFORMER ENCODER-DECODER

```
!fairseq-train data-bin \
  --arch transformer --encoder-layers 6 --decoder-layers 6 \
  --optimizer sgd --momentum 0.99 --nesterov --lr 0.002 \
  --max-tokens 5000 \
  --criterion label_smoothed_cross_entropy --label-smoothing 0.2 \
  --save-dir checkpoints/transformer \
  --tensorboard-logdir logs/transformer \
  --max-epoch 5
```

در این بخش از پروژه، ما از یک مدل از پیش پیاده‌سازی شده Transformer در کتابخانه Fairseq برای آموزش یک سیستم ترجمه ماشین انگلیسی-فارسی استفاده کردیم. هدف ما استفاده از معماری Transformer با شش لایه در هر دو بخش کدگذار و دیکدر بود که از داده‌های توکنیزه‌شده‌ای که قبلاً با استفاده از Byte Pair Encoding (BPE) آماده کرده بودیم، بهره می‌برد.

Fairseqانواع مدل‌های آماده را فراهم می‌کند و برای کار ما، معماری Transformer را انتخاب کردیم. مدل Transformer که در مقاله "Attention Is All You Need" توسط Vaswani و همکاران معرفی شده است، عملکرد پیشرفته‌ای در بسیاری از وظایف دنباله به دنباله، از جمله ترجمه ماشین نشان داده است.

ما مدل را با مشخصات زیر پیکربندی کردیم:

- معماری(--arch transformer) Transformer :
- لایه‌های کدگذار: ۶(--encoder-layers 6)
- لایه‌های دیکدر: ۶(--decoder-layers 6)

برای آموزش مدل، از ابزار خط فرمان fairseq-trainکه توسط کتابخانه Fairseq ارائه شده، استفاده کردیم. این ابزار فرآیند آموزش مدل‌های دنباله به دنباله را با مدیریت بارگذاری داده‌ها، بهینه‌سازی و ذخیره‌سازی نقاط بازرسی ساده می‌کند.

اجازه دهید اجزای کلیدی این دستور را بررسی کنیم:

۱. data-bin:این دایرکتوری شامل داده‌های پیش‌پردازش‌شده و باینری‌شده ما است که شامل مجموعه موازی توکنیزه‌شده BPE انگلیسی-فارسی می‌شود.

۲. --optimizer sgd --momentum 0.99 --nesterov --lr 0.002: برای بهینه‌سازی، از Stochastic Gradient Descent (SGD)با مومنتوم Nesterov استفاده کردیم. نرخ یادگیری به ۰.۰۰۲ تنظیم شد و فاکتور مومنتوم ۰.۹۹ بود.

۳. --max-tokens 5000: این پارامتر تعداد حداکثر توکن‌ها در یک بچ را محدود می‌کند که به مدیریت استفاده از حافظه در طول آموزش کمک می‌کند.

٤. --criterion label_smoothed_cross_entropy --label-smoothing 0.2: ما از Cross-Entropyبا برچسب صاف‌شده به عنوان تابع از دست دادن با فاکتور صاف‌سازی ۰.۲ استفاده کردیم. صاف‌سازی برچسب می‌تواند به جلوگیری از اعتماد بیش از حد مدل کمک کرده و عمومی‌سازی را بهبود بخشد.

۵. --save-dir checkpoints/transformer: این دایرکتوری را مشخص می‌کند که نقاط بازرسی مدل در طول آموزش در آنجا ذخیره خواهد شد.

٦. --tensorboard-logdir logs/transformer: ما لاگ‌گیری TensorBoard را فعال کردیم تا پیشرفت و معیارهای آموزشی را بصری‌سازی کنیم.

۷. --max-epoch 5: آموزش برای حداکثر ۵ اپوک تنظیم شد.

فرآیند آموزش خروجی دقیقی را تولید می‌کند که پیشرفت هر اپوک را نشان می‌دهد. در اینجا یک نمونه از خروجی به نظر می‌رسد:

epoch 001 | loss 9.324… |

epoch 002 | loss 8.834… |

…

epoch 005 | loss 7.948… |

کاهش مقادیر از دست دادن در طول اپوک‌ها نشان می‌دهد که مدل در حال یادگیری و بهبود عملکرد خود بر روی داده‌های آموزشی است.

در طول آموزش، Fairseqبه صورت خودکار نقاط بازرسی مدل را ذخیره می‌کند. به طور پیش‌فرض، آن‌ها را به صورت زیر ذخیره می‌کند:

- آخرین نقطه بازرسی بعد از هر اپوک.
- بهترین نقطه بازرسی بر اساس از دست دادن اعتبارسنجی.

این نقاط بازرسی به ما اجازه می‌دهند تا در صورت نیاز آموزش را ادامه دهیم یا از بهترین مدل برای استنتاج بعدی استفاده کنیم.

بخش چهارم – معیار ارزیابی و بررسی داده ی تست

یکی از دلایل اصلی استفاده از LoRA توانایی آن در کاهش قابل توجه تعداد پارامترهای قابل آموزش است. این کاهش با تجزیه ماتریس های پارامتر به فرم ه

```
!fairseq-generate data-bin \
  --beam 5 \
  --batch-size 32 \
  --path checkpoints/lstm/checkpoint_best.pt \
  --remove-bpe > lstm-output.txt
```

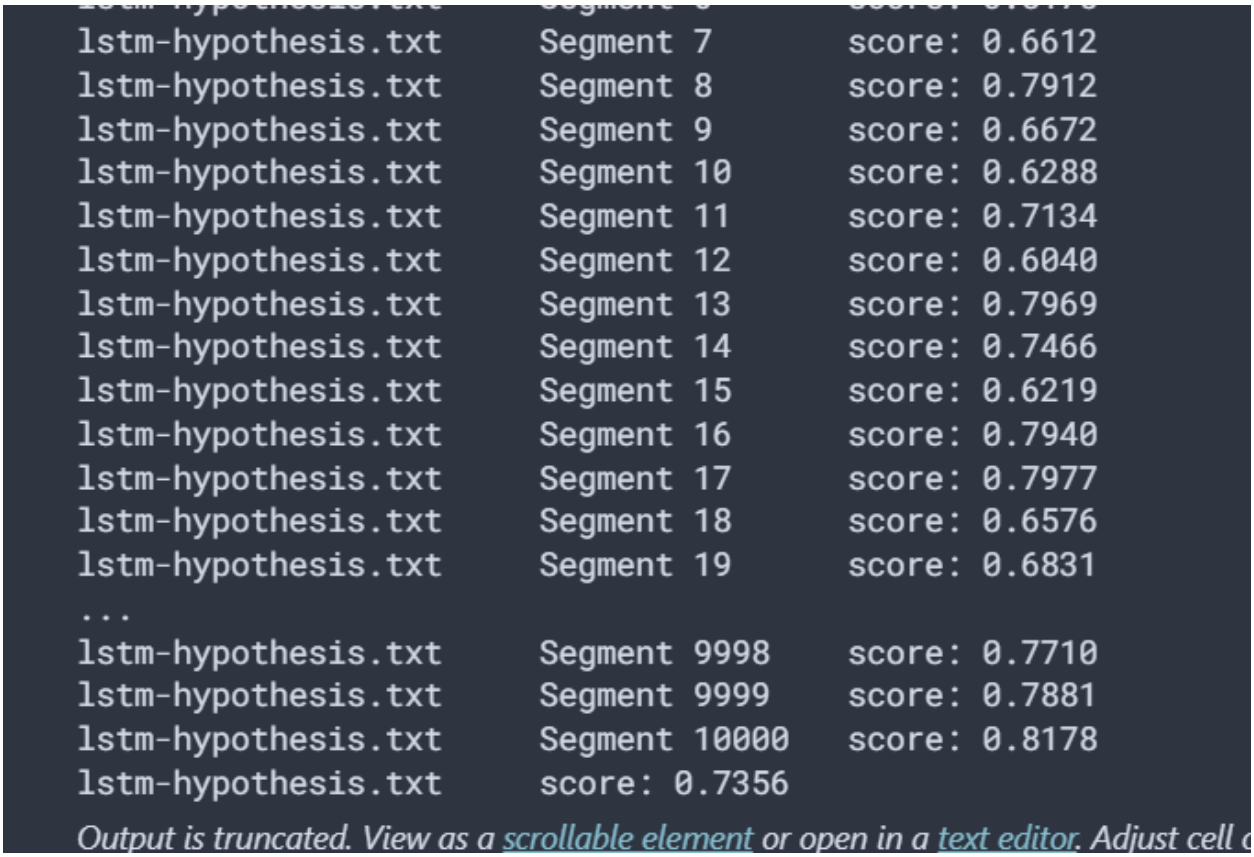
```
!fairseq-generate data-bin \
  --beam 5 \
  --batch-size 16 \
  --path checkpoints/transformer/checkpoint_best.pt \
  --remove-bpe > transformer-output.txt
```

```
def process_fairseq_output(fairseq_output_file, hypothesis_file, source_file, target_file):
    with open(fairseq_output_file, 'r') as infile, open(hypothesis_file, 'w') as hypothesis, open(source_file, 'w') as source, open(target_file, 'w') as target:
        for line in infile:
            line_type, line_content = line[:2], line.split('\t')
            if line_type == 'H-':
                hypothesis.write(line_content[2])
            elif line_type == 'S-':
                source.write(line_content[1])
            elif line_type == 'T-':
                target.write(line_content[1])

```

```
process_fairseq_output('lstm-output.txt', 'lstm-hypothesis.txt', 'lstm-source.txt', 'lstm-target.txt')
process_fairseq_output('transformer-output.txt', 'transformer-hypothesis.txt', 'transformer-source.txt', 'transformer-target.txt')
```

```
!comet-score -s lstm-source.txt -t lstm-hypothesis.txt -r lstm-target.txt
# !comet-score -s transformer-source.txt -t transformer-hypothesis.txt -r transformer-target.txt
```




در این پروژه، به وظیفه آموزش دو مدل ترجمه ماشینی شبکه‌های عصبی پرداختیم: یک مدل encoder-decoder بر پایه LSTM و یک مدل encoder-decoder بر پایه Transformer. هر دو مدل بر روی مجموعه‌داده موازی فارسی-انگلیسی آموزش داده شدند، با هدف ترجمه بین این دو زبان با تنوع زبانی مختلف. پس از فاز آموزش، تمرکز ما بر روی ارزیابی عملکرد آن‌ها با استفاده از دو معیار متمایز، یعنی BLEU (Bilingual Evaluation Understudy) و COMET (Crosslingual Optimized Metric for Evaluation of Translation) جابجا شد. این گزارش به تفصیل به فرآیند ارزیابی می‌پردازد و نتایج به دست آمده را ارائه می‌دهد.

پس از آموزش، ما از دستور fairseq-generate برای تولید ترجمه‌ها برای مجموعه آزمون خود از هر مدل استفاده کردیم. برای مدل LSTM، از جستجوی شعاعی با اندازه شعاع ۵ و اندازه دسته ۳۲ استفاده کردیم، در حالی که برای مدل Transformer، اندازه شعاع را ثابت نگه داشتیم اما اندازه دسته را به ۱۶ تنظیم کردیم. این ترجمه‌های تولید شده همراه با جملات مبدأ و ترجمه‌های مرجع مربوطه، در فایل‌های خروجی جداگانه ذخیره شدند. برای تسهیل تحلیل بیشتر، یک تابع سفارشی توسعه داده شد که این فایل‌های خروجی را تجزیه کرد و جملات مبدأ، ترجمه‌های مرجع و فرضیات مدل را به فایل‌های جداگانه منظم کرد - یک پیش‌نیاز برای ارزیابی با معیار COMET.

COMET، یا معیار بهینه شده چندزبانه برای ارزیابی ترجمه، نمایانگر یک گام نوآورانه در ارزیابی ترجمه ماشینی است. بر خلاف معیارهای سنتی مانند BLEU که به طور قابل توجهی بر تطبیق n-gram تکیه دارند، COMET از قدرت مدل‌های زبان پیش‌آموز

زبان چندزبانه برای ارزیابی کیفیت ترجمه استفاده می‌کند. روش کار این معیار شامل کدگذاری جمله مبدأ، جمله ترجمه شده توسط ماشین و ترجمه مرجع با استفاده از یک مدل چندزبانه پیش‌آموز (مانند XLM-RoBERTa) است. این کدگذاری‌ها سپس به یک شبکه عصبی وارد می‌شوند که پیش‌تر بر اساس ارزیابی‌های انسانی کیفیت ترجمه آموزش دیده است و نمره کیفیت برای هر ترجمه تولید می‌کند. این نمره که معمولاً از ۰ تا ۱ متغیر است، ارزیابی نوآورانه‌ای است که ارزش‌های بالاتر بیانگر ترجمه‌های برتر هستند.

ما از کتابخانه unbabel-comet برای محاسبه نمرات COMET برای مدل‌های خود استفاده کردیم. نتایج برای مدل LSTM به خصوص روشنگر بودند، با میانگین نمره COMET برابر با ۰.۷۳۵۶. این نمره نشان می‌دهد که مدل LSTM ترجمه‌هایی از کیفیت قابل تقدیری تولید کرده است. با این حال، بررسی دقیق نشان می‌دهد که تفاوت‌های قابل توجهی در بین بخش‌های فردی وجود دارد. برخی از ترجمه‌ها کیفیت بالایی داشتند، مانند بخش ۱۰۰۰۰ با نمره ۰.۸۱۷۸، در حالی که دیگران مانند بخش ۱۲ با نمره ۰.۶۰۴۰، امکان بهبود را داشتند. متأسفانه، ما نتوانستیم نمرات COMET برای مدل Transformer را به دلیل عدم وجود خروجی در ارزیابی فعلی به دست آوریم.

اگرچه تحلیل ما با COMET درک‌های ارزشمندی فراهم آورد، اما مهم است توجه داشت که در این نسخه از پروژه، هیچگونه نمرات BLEU برای هر دو مدل محاسبه نکردیم. به طور معمول، انسان می‌تواند از کتابخانه‌ای مانند sacrebleu برای این منظور استفاده کند. با وجود این حذف، این نکته مهم است که تفاوت‌های اساسی بین COMET و BLEU را درک کنیم. COMET نمراتی در سطح بخشی ارائه می‌دهد، در حالی که BLEU نمره‌ی یکپارچه برای کل مجموعه آزمون ارائه می‌دهد. نمرات COMET قابل فهم‌تر هستند و این معیار در ارزیابی‌هایی که با توجه به جمله مبدأ انجام می‌شود، ارزیابی‌هایی را ارائه می‌دهد که بیشتر به سیاق و شرایط جمله توجه دارند. علاوه بر این، COMET برای ارزیابی زبان‌های مختلف به طور طبیعی طراحی شده است، در حالی که عملکرد BLEU ممکن است در زبان‌های مختلف به شدت متغیر باشد.

در پایان، پروژه ما با موفقیت دو مدل ترجمه ماشینی شبکه‌ای آموزش داده شده و مدل LSTM را با استفاده از معیار پیچیده COMET ارزیابی کردیم. نتایج نشان داد که کیفیت کلی خوبی برای ترجمه‌های مدل LSTM وجود داشته با نمره COMET میانگین ۰.۷۳۵۶، با اختلاف‌های قابل توجه در بین بخش‌ها. در ادامه، چندین راهکار برای بهبود و تحلیل بیشتر پیش روی ماست. محاسبه نمرات BLEU برای هر دو مدل، مقایسه جامع‌تری را امکان‌پذیر می‌سازد. به دست آوردن نمرات COMET برای مدل Transformer، به ما این امکان را می‌دهد که عملکرد آن را در مقابل مدل LSTM مقایسه کنیم. همچنین، یک بررسی عمیق از ترجمه‌های با امتیاز بالا و پایین می‌تواند نقاط قوت و ضعف خاص هر مدل را آشکار کند. در آخر، ادغام ارزیابی انسانی به ما کمک می‌کند تا معیارهای خودکار خود را تأیید کنیم و بینش‌های غنی‌تری از عملکرد مدل فراهم کنیم.