NTU ADL 2023 Fall - HW2

written by Chih Han, Yang. B10902069

Model

Model Description

This is the Multilingual Text to Text Transfer Transformer, commonly known as MT5. For this assignment, we will engage in the fine-tuning process of this pre-trained model specifically for a text summarization task. In this task, the model is responsible for producing an appropriate title for a given paragraph. This task aligns with the typical sequence-to-sequence nature, specifically the text-to-text task that the T5 model excels at. The T5 model comprises two crucial components: a transformer encoder and a transformer decoder. The encoder takes the paragraph sequence as input and generates a sequence of outputs. Subsequently, the decoder employs an autoregressive approach to generate the title. The first layer of the decoder performs self-attention. On top of that, the second layer of the decoder engages in “cross attention” with the encoder’s output, incorporating key-value pairs from the encoder’s output and queries from the decoder’s first-layer self-attention output. Following this cross-attention step, the output proceeds through additional layers before producing the final result. It is important to note that the self-attention mechanism in the decoder is referred to as “masked self-attention” due to the autoregressive behavior. This implies that the decoder utilizes its own previous output as input to generate the next token; thus, it lacks visibility of any future tokens, only attending to prior tokens.

Data Preprocessing

Data preprocessing is crucial for fine-tuning a sequence-to-sequnce model. We should avoid using dirty data to affect the model’s performance. The data preprocessing is done in the following steps:

1. Drop the duplicate data. If a dataset has excessive duplicate data, we should remove them to avoid overfitting on those data.
2. Drop some useless characters. In my code, I dropped space and newline characters from the input paragraphs. These two characters are irrelevant to the actual context. However, we shouldn’t overly clean up too many seemingly irrelevant tokens. The model performs worse based on my attempts. My guess is that some tokens, such as “”" or “,” are actually crucial to constructing the sentence.
3. Add a special token to notify the model to start the generative process.
4. Tokenize the data with sentencepiece tokenization. The approach employed by SentencePiece is Byte-Pair Encoding tokenization (BPE). BPE determines the vocabulary by iteratively merging the most frequent token pairs into a new vocabulary set, ensuring diversity and comprehensiveness. This process is guided by a set of merging rules. Subsequently, each word is split into characters, and these rules are applied until no further merging can occur.

Hyperparameter

This is the hyperparameter set for this fine-tuning process:

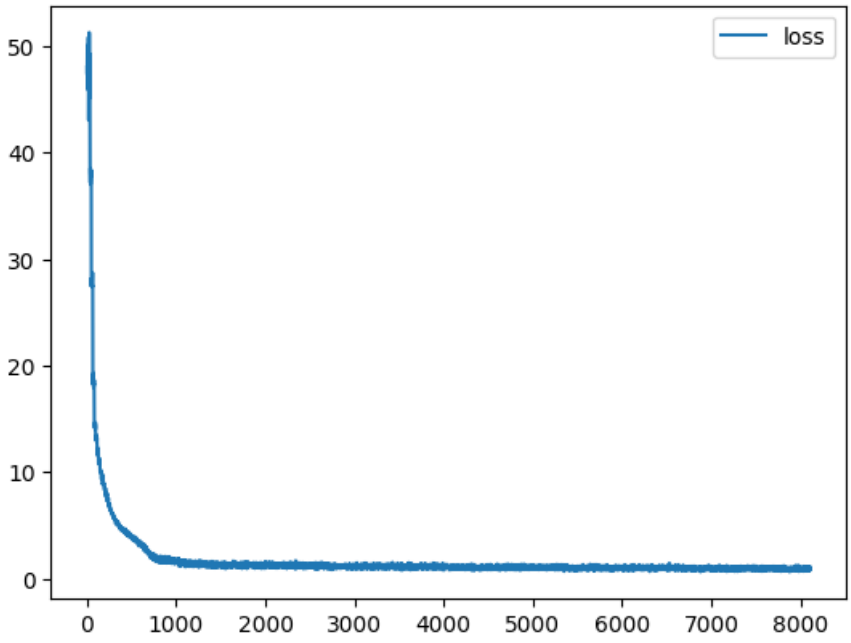
- max_output_length = 64 : The maximum length of the title labels in the dataset is no longer than 64.
- max_input_length = 256 : In our conventional thinking, longer input sequences tend to yield better results because the model can take into account more information. However, this necessitates a larger model with the capacity to process this additional information. In this assignment, we utilize the “mt5-small” model. If we set the maximum input to, for instance, 1024, it produces erratic results and the loss cannot be effectively optimized.
- learning_rate = 1e-3 : Using linear learning rate scheduler.
- num_train_epochs = 6 : I attempted 3 epochs previously, which displayed the potential for more

optimized models. Consequently, doubling the number of epochs yielded an improved result.

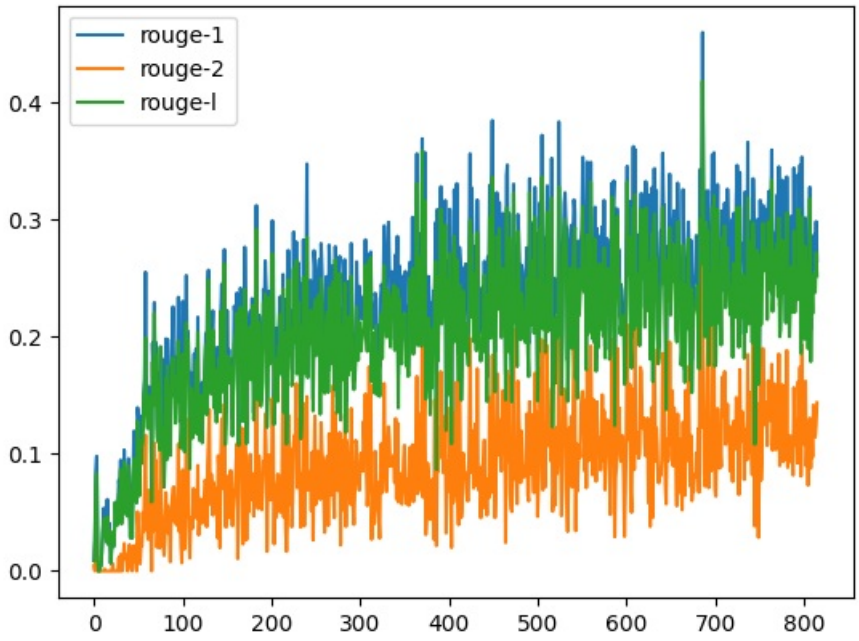
- batch_size = 16 : Different paragraphs from the dataset can have diverse contexts and lengths. I tried to set the batch size as large as the GPU could afford to generate a comprehensive result.
- seed = 20231104

Learning Curves

- Loss Learning Curve: Cross Entropy



- Rouge Learning Curve: Rouge Score
  - blue line : unigram
  - orange line: bigram
  - green line: longest common subsequence gram



Generation Strategies

Describe the details of the following generation strategies:

1. Greedy: In this strategy, the model simply selects the token with the highest probability at each step. It makes the locally optimal choice at each step, aiming to maximize the likelihood of the entire sequence.

2. Beam Search: The beam search algorithm chooses the top N tokens with the highest probability from each decoder's vector outputs and concatenate them into a generated sequence of text. It explores multiple possibilities in parallel, which could improves the greedy algorithm, aiming to maximize the likelihood of the entire sequence from those beam candidates.
3. Top-k Sampling: In top-k sampling, the model **samples** from the top k most likely tokens at each step. This allows for some randomness in the generated sequences while still maintaining control over the likelihood of the chosen tokens. It helps to introduce diversity in the generated text because the algorithm doesn't choose the optimal token.
4. Top-p Sampling: Top-p sampling selects from the smallest set of tokens whose cumulative probability mass exceeds a predefined threshold p(0<p<=1). This strategy dynamically adjusts the number of considered tokens based on their cumulative probabilities, which can lead to more diverse and controlled outputs.
5. Temperature: Temperature is a technique that adjusts the probabilities outputted by the model. A higher temperature value (>1) makes the distribution more uniform and deterministic, leading to more randomness in the generated text. A lower temperature value (<1) sharpens the distribution, making the output more deterministic.

Hyperparameters

After training the model, I perform evaluation on the first 100 features of the data with 5 different strategies: (the following scores = f1-score * 100)

1. Greedy

```
greedy_config = GenerationConfig(max_length=args.max_output_length)
```

```
rouge-1: 20.40792833902558
rouge-2: 7.236797986414389
rouge-l: 18.42256309905828
```

2. Beam Search

```
beam_config = GenerationConfig(max_length=args.max_output_length,
                               num_beams=10,
                               length_penalty=1,
                               no_repeat_ngram_size=2,
                               early_stopping=True)
```

```
rouge-1: 24.3631487179364
rouge-2: 10.309077519928305
rouge-l: 22.436342505238038
```

3. Sampling with Temperature

```
sampling_temp_config = GenerationConfig(max_length=args.max_output_length,
                                        do_sample=True,
                                        top_k=0,
                                        temperature=0.7)
```

```
rouge-1: 18.40069198195147
rouge-2: 5.184157273858043
rouge-l: 16.713770074517704
```

4. Top-K Sampling

```
topK_sampling_config = GenerationConfig(max_length=args.max_output_length,
                                        do_sample=True,
                                        top_k=model.config.vocab_size//5)
                                    # Set k to the top 20% of the vocab size
```

```
rouge-1: 10.780672671962291
rouge-2: 1.918568360183683
rouge-l: 9.843856822193116
```

5. Top-P Sampling

```
topP_sampling_config = GenerationConfig(max_length=args.max_output_length,
                                        do_sample=True,
                                        top_p=0.95,
                                        top_k=model.config.vocab_size//5)
    # Set p to 95% of the cumulative probability of the vocab set
    # Used along side with top-K technique
```

```
rouge-1: 12.373567040062678
rouge-2: 3.61369356590339
rouge-l: 11.752682100349887
```

My ultimate generation strategy employs beam search with a beam width of 10. This approach consistently yields the highest Rouge score when compared with the label. This choice is justified, as beam search is designed to seek an optimized output, minimizing randomness in the generated text.

Useful Links

1. Homework Description Slides
2. 2. About the accelerate package:
   - accelerate
   - Basic tutorials
3. About mT5 model:
   - mt5 paper
   - mt5-small
   - mt5 documentation
   - t5 Tokenizer
   - MT5ForConditionalGeneration source code
4. fine-tuning mt5:
   - Metric - twrouge
   - Deal with CUDA out of memory
   - About model.train() model.eval()
5. Generation strategy:
   - About generation strategy
   - generate
   - GenerationConfig