Report

1. Model

In the 2b_t5 model, I used the t5-small-chinese-cluecorpussmall model, which is a Chinese T5 model trained with CLUECorpusSmall data. In 2b_gpt2, I used the gpt2-distil-chinese-cluecorpussmall model, also trained on CLUECorpusSmall.

T5: This model includes an encoder and decoder, which excels in tasks like question answering, translation, and summarization, all of which are text-to-text tasks. It combines unsupervised and supervised learning methods during pretraining. T5 converts all NLP tasks into a text-to-text format, allowing a single model to handle a wide range of tasks without needing individual task-specific training.

GPT2: This model uses only a decoder and is mainly good at text generation tasks (generating text based on the context of previous text). It was pretrained using unsupervised learning.

2. Dataset

T5: In the __getitem__ function, I process the data by encoding the text and summary and converting them into PyTorch tensors, resulting in token ID lists for both the text and the summary. These are padded to the same length (padding token IDs: text uses tokenizer.pad_token_id, and summary uses -100, ensuring that the model doesn't calculate loss for the padding parts). The model receives the text as input and the summary as its corresponding label.

GPT2: I used three functions to process the data: process, process2, and collator_fn. (Training: process + collator_fn; Evaluation: process2 + collator_fn)

(1) process: Model input: CLS + text token ID list + CLS + SEP + summary token ID list + SEP (encoded). Model label: [-100] * len(CLS + text token ID list + CLS) + SEP + summary token ID list + SEP (encoded).

I designed this sequence to help the model learn that CLS represents the start/end of a text, and SEP marks the start/end of the summary. In evaluation, by inputting CLS + text token ID list + CLS, the model should understand that it needs to generate a summary.

I set the label as: [-100] * len(CLS + text token ID list + CLS) + SEP + summary token ID list + SEP. By using -100, I ensure that the model calculates the loss only for the summary portion, aiming to generate more accurate summaries.

(2) process2: Model input: CLS + text token ID list + CLS (encoded), Model label: [-100] * len(CLS + text token ID list + CLS) + SEP + summary token ID list + SEP (encoded).

(3) collator_fn: Converts each element of the model's input and label into PyTorch tensors and applies padding. The padding token ID for the labels is set to -100 to ensure that the model does not compute the loss for the padded portions."

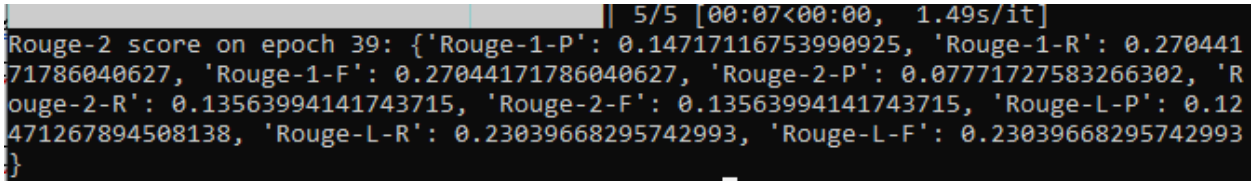
3. Training

For both the T5 and GPT2 models, I used the same hyperparameters: BATCH_SIZE = 64, LEARNING_RATE = 1e-3, and EPOCHS = 40. The training steps were identical as well:
- optimizer.zero_grad(): Clears the gradients from the previous iterations.
- outputs = model(**batch) / outputs = model(input_ids=input_ids, attention_mask=attention_mask, labels=labels): Passes the batch content as parameters to the model.
- loss = outputs.loss: Retrieves the loss value.
- loss.backward(): Computes the gradients based on the loss.
- optimizer.step(): Updates the model parameters according to the calculated gradients.

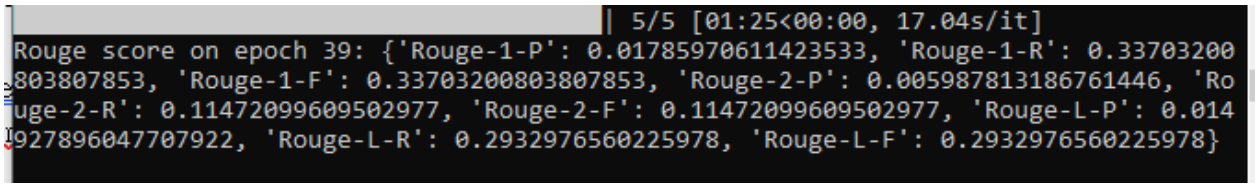
4. Evaluation

I used ROUGE to evaluate the models. Below is the ROUGE score for both the T5 and GPT2 models (only the final epoch is shown due to the large number of results).

T5:



GPT2:



Based on the ROUGE scores, I found that the T5 and GPT2 models performed similarly in ROUGE-R and ROUGE-F metrics. However, GPT2's ROUGE-P was noticeably lower than T5's,

which might suggest that GPT2 generated more content not included in the reference summaries.