CSIE5431 ADL HW2

陳禹翰 B10705050

October 22, 2024

Q1: Model (2%)

Model (1%)

Describe the model architecture and how it works on text summarization.

<u>**Model Architecture**</u>
Encoder-Decoder Structure

- **Encoder**: The encoder processes the input text and generates hidden states that capture the contextual information.
- **Decoder**: The decoder uses these hidden states to produce the output text. It employs self-attention mechanisms to focus on relevant parts of the input while generating each token in the output.

Components

- **Self-Attention Mechanism**: This allows the model to weigh the importance of different words in the input sequence when generating outputs.
- **Feed-Forward Networks**: Each layer in both encoder and decoder contains feed-forward networks that further process the attention outputs.
- **Task-Specific Heads**: The `T5ForConditionalGeneration` class includes an additional linear layer (`lm_head`) on top of the decoder to generate the final output tokens based on the decoder's hidden states.

Text-to-Text Framework
T5's unique feature is its text-to-text framework, where every NLP task is treated as converting one piece of text into another. For instance:

- **Input**: "summarize: The quick brown fox jumps over the lazy dog."
- **Output**: "A fox jumps over a dog."

This allows for a unified approach across different tasks, simplifying model training and application56.

<u>**How T5 Works for Text Summarization**</u>

- **Input Preparation**: The input text is prefixed with a task descriptor (e.g., "summarize:") to inform the model of the desired operation.
- **Tokenization**: The input is tokenized using a specific tokenizer that converts text into numerical IDs compatible with the model.

- **Forward Pass**:
  - The tokenized input is fed into the encoder, which generates hidden states.
  - These states are passed to the decoder, which generates output tokens sequentially.
- **Output Generation**: The model produces a sequence of tokens that represent the summary. This is done through autoregressive generation, where each token is generated based on previously generated tokens and the encoder's hidden states.
- **Decoding Strategy**: Various strategies can be employed during decoding, such as greedy search or beam search, to enhance output quality and coherence.

Preprocessing (1%)

Describe your preprocessing (e.g., tokenization, data cleaning, etc.).

For my preprocessing steps in this text summarization project, I first load the dataset from a file, where each line contains a JSON object with the article's main text and its corresponding title. Here's how I approach the process:

1. **Data Loading and Limiting**:
   - I read the dataset line by line, and for each line, I convert the JSON object into a dictionary. I store each dictionary in a list.
   - If I want to limit the number of training samples, I use the `max_train` argument to stop reading more lines after reaching the desired amount.
2. **Tokenization**:
   - I use a tokenizer to convert the main text of the article (input) and the title (target summary) into token sequences.
   - For each article's main text, I tokenize it with a fixed maximum input length (`max_input_length`). If the text exceeds this length, I truncate it; if it's shorter, I pad it to ensure uniform input sizes across the dataset.
   - Similarly, I tokenize the title with a maximum output length (`max_output_length`), truncating or padding it as necessary to create a consistent target summary length.
3. **Attention Masks and Input Preparation**:
   - Along with tokenization, I generate attention masks, which allow the model to focus on non-padded tokens during training.
   - The tokenized input and output sequences, as well as the attention masks, are returned in PyTorch tensor format to ensure compatibility with the model during training.

Q2: Training (2%)

Hyperparameter (1%)

Describe the hyperparameters you use and how you decide on them.

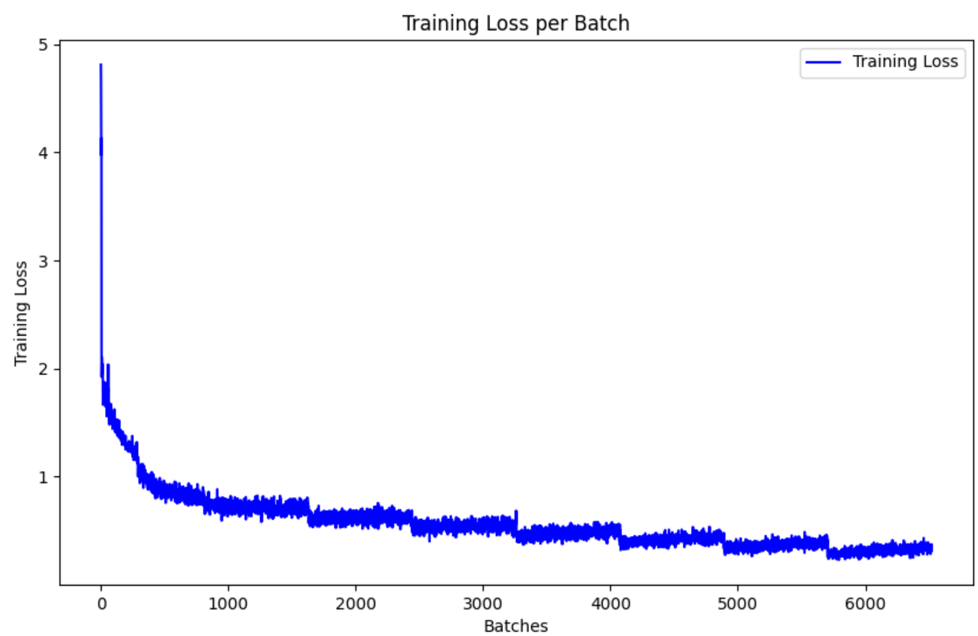
Name	Value	Reason/Description
Model	"google/mt5-small"	Multilingual support
Batch Size	24	Maximize memory usage, bigger batch size for better training stability
Learning Rate	5e-5	Common starting point for transformers
Epochs	8	Save every epoch, select optimal model, conserve resources.
Max Input Length	512	Handles long documents like news articles
Max Output Length	100	Ensures concise summaries
Gradient Accumulation Steps	5	Increases batch size without memory overflow
Validation Split	10%	Tracks performance on unseen data
FP16	Disabled	Avoids instability
Adafactor	Enabled	Memory efficiency

Table 1: Hyperparameters for Text Summarization

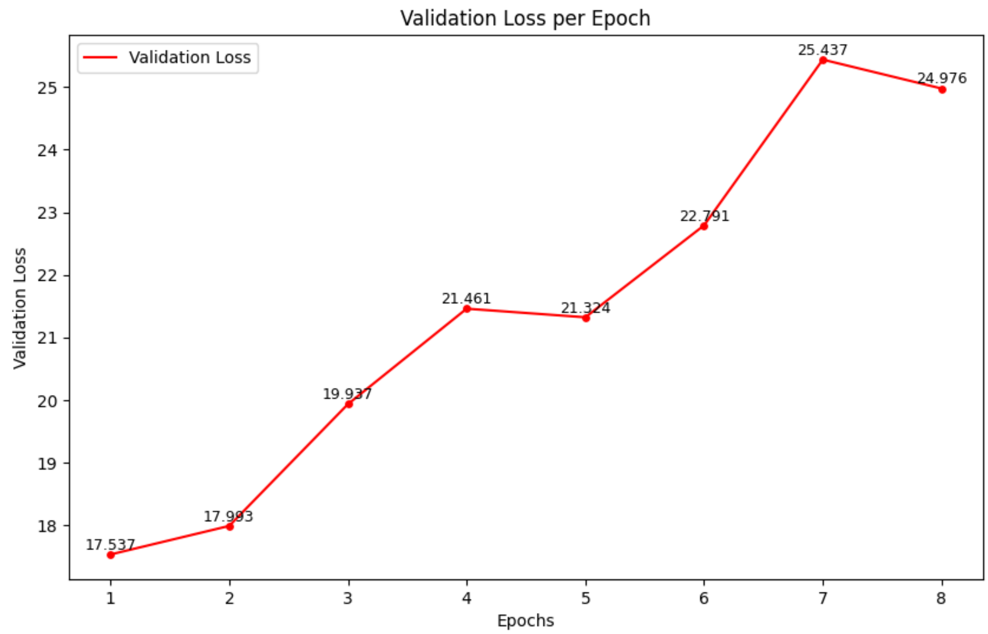
Learning Curves (1%)

Plot the learning curves (ROUGE versus training steps).

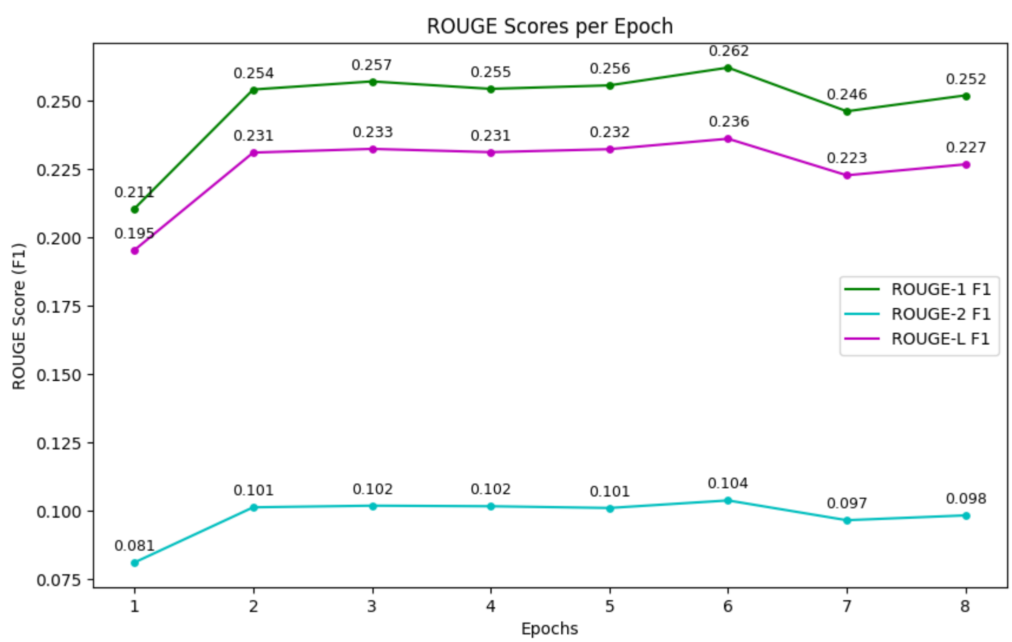
- Training Loss Per Batch



- Validation Loss Per Epoch



– ROUGE Scores Per Epoch



Epoch	ROUGE-1 F1	ROUGE-2 F1	ROUGE-L F1
5	0.2323	0.0872	0.2106
6	0.2337	0.0869	0.2117
7	0.2182	0.0805	0.1967

Table 2: Public data ROUGE F1 Scores for Epochs 5, 6, and 7

Q3: Generation Strategies (6%)

Strategies (2%)

Describe the detail of the following generation strategies: Greedy, Beam Search, Top-k Sampling, Top-p Sampling, Temperature.

Strategy	**Value**	**Description**
Beam Search	`num_beams=5`	Explores 5 sequences at once, improves coherence
Early Stopping	`early_stopping=True`	Stops when end token is generated
Temperature	`temperature=1.0`	Controls randomness, 1.0 keeps it neutral
Nucleus Sampling	`top_p=0.95`	Samples from the top 95% cumulative probability tokens
Top-k Sampling	`top_k=50`	Limits choices to top 50 most probable tokens to control variability

Table 3: Generation Strategies and their Values

Hyperparameters (4%)

Try at least 2 settings of each strategy and compare the results. What is your final generation strategy? (You can combine any of them).

Default Strategy:

- `num_beams=5`,
- `early_stopping=True`,
- `temperature=1.0`,
- `top_p=0.95`,
- `top_k=50`

Strategy	**ROUGE-1 F1**	**ROUGE-2 F1**	**ROUGE-L F1**
Default	0.2337	0.0869	0.2117
Temperature = 0	0.2182	0.0805	0.1967
Greedy	0.2043	0.0666	0.1838
Num Beams = 3	0.2315	0.0836	0.2096
Num Beams = 10	0.2355	0.0887	0.2134
Num Beams = 15	0.2365	0.0897	0.2140
Num Beams = 20	0.2367	0.0901	0.2140
Top-k = 30	0.2337	0.0869	0.2117
Top-k = 70	0.2337	0.0869	0.2117
Top-p = 50	0.2337	0.0869	0.2117
Top-p = 75	0.2337	0.0869	0.2117

Table 4: ROUGE F1 Scores for Different Generation Strategies in Epoch 6

From my test results, we can see that setting temperature to 0 or using Greedy Algorithm is no better than the default setting, and making changes to top-k or top-p does not make so much of a difference. However, it seems that `num_beams` indeed matters. The higher `num_beams` is, the higher the score. Moreover, the increase of f1-score seems to converges at `num_beams=20`.

Hence, the final strategy I will choose is

- `num_beams=20`,
- `early_stopping=True`,
- `temperature=1.0`,
- `top_p=0.95`,
- `top_k=50`

Bonus: Applied GPT-2 on Summarization (2%)

- **Model (1%)**
  Describe GPT-2 architecture and hyperparameters you use.

  GPT-2 architecture
  GPT-2, developed by OpenAI, is a generative pre-trained transformer model characterized by its large scale and capability for various natural language processing tasks. Below is a detailed overview of its architecture and functional components.

  Transformer Decoder

  * **Type**: GPT-2 is a **decoder-only** transformer model, which means it utilizes only the decoder part of the transformer architecture as described in the "Attention is All You Need" paper.
  * **Layers**: The model consists of multiple layers (12 in the smallest version, up to 48 in larger variants), each containing:
    · **Multi-Head Self-Attention Mechanism**: This allows the model to attend to different parts of the input sequence simultaneously, enhancing its ability to understand context.
    · **Feed-Forward Neural Networks**: Each layer includes a feed-forward network that processes the output from the attention mechanism.

  Key Features

  * **Parameters**: The largest GPT-2 model contains **1.5 billion parameters**, significantly larger than its predecessor, GPT-1, which had around 117 million parameters.
  * **Context Length**: GPT-2 can process sequences of up to **1024 tokens**, doubling the context length compared to GPT-1's 512 tokens.
  * **Vocabulary Size**: The vocabulary consists of **50,257 unique tokens**, allowing for nuanced language generation.

- **Compare to T5 Model (1%)**
  Observe the loss, ROUGE score, and output texts. What differences can you find?

Feature	**GPT-2**	**google/mt5-small**
Type	Decoder-only	Encoder-decoder
Parameters	Up to 1.5 billion	Approximately 300 million
Training Data	WebText (8 million pages)	C4 dataset
Objective	Next-token prediction	Text-to-text transformation
Attention Mechanism	Masked self-attention	Self-attention in both encoder & decoder
Context Length	Up to 1024 tokens	Variable, optimized for tasks

Table 5: Comparison of GPT-2 and google/mt5-small Features

While both models leverage transformer architectures, GPT-2 focuses on generating coherent text through a unidirectional approach, whereas google/mt5-small employs a bidirectional encoder-decoder framework that allows it to perform a broader range of NLP tasks effectively. This fundamental difference in architecture underpins their respective strengths in handling language-related tasks.