ADL Homework 2 Report

B10705016

Q1: Model (2%)

Model (1%)

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

MT5-small 模型使用了 Seq2Seq 的 Transformer 架構, 用於解決文字到文字的任務, 例如這次作業是把文章內容摘要成新聞標題。

Transformer 有兩部分:

- 編碼器:讀取原文, 理解內容, 生成 embedding。
- 解碼器:根據被編碼的內容, 逐字生成摘要。關鍵是 self-attention 機制, 讓模型在處理每個詞時, 都能考慮整個句子的其他詞, 生成更連貫的摘要。

訓練時, 我們用原文和對應的摘要, 讓模型學習怎麼抓取重點。
生成時, 模型一步步預測下一個詞, 直到完成摘要。

Preprocessing (1%)

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

我在原程式碼中的 preprocess_function 函式進行預處理。這個函式負責分詞和準備訓練所需的資料。

預處理的步驟:

- 輸入的分詞:
  - 為每個輸入文本添加前綴(對於像 T5 這類模型可能需要)。
  - 使用 tokenizer 將輸入文本轉換為 token IDs, 並根據參數進行 truncation 和 padding。
- 目標摘要的分詞:
  - 使用 tokenizer 的目標模式, 將摘要分詞為 token IDs。
  - 根據 max_target_length 進行截斷和填充。
- 標籤的準備:
  - 如果有填充, 且 ignore_pad_token_for_loss 設為 True, 會將標籤中的填充 token IDs 替換為 -100, 確保在計算 loss 時忽略這些 token。

此外, 我原本嘗試將訓練資料集中內文長度小於 10 或大於 1024 或重複的內容之資料去除, 但發現只會去除掉 40 餘筆的資料, 因此最後決定直接不做額外處理了。

Q2: Training (2%)

Hyperparameter (1%)

Describe your hyperparameter you use and how you decide it.

- max_source_length: 512
- max_target_length: 64
- num_beams: 4
- per_device_train_batch_size=4
- per_device_eval_batch_size=4

首先，batch size 是基於上次作業的經驗設定的。

其次，實驗後發現 beams 數字越大效果越好，但到了 4 以上就沒有顯著差異，因此把 beams 設在 4。

最後，我對資料集的標題和內文進行了長度的統計，其結果如下：

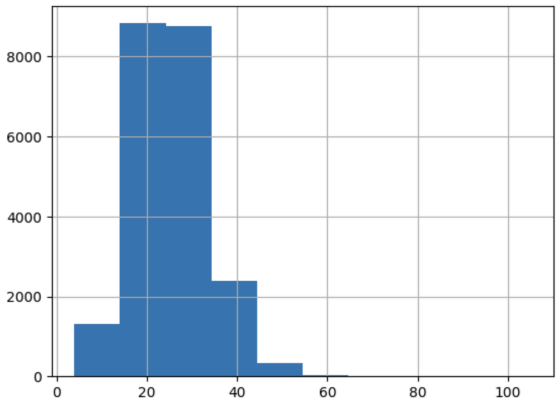


圖1：訓練資料集標題長度分佈

如圖所示，大部分的文章標題都落在 60 以內，因此我將 max_target_length 設在 64。

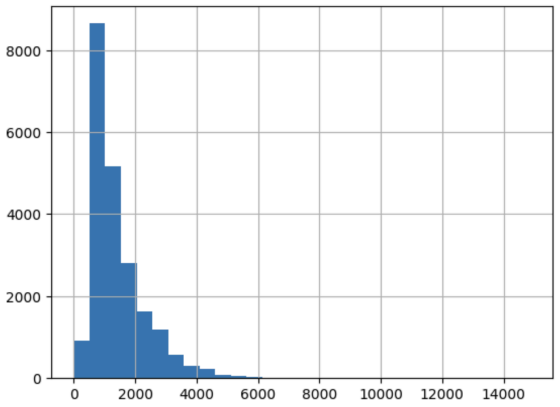


圖2：訓練資料集標題長度分佈

如圖所示，大部分的文章內文長度落在 2000 以內，原想將 max_source_length 設為 1024，但考慮到訓練成本，加上內文在 tokenization 過後長度應該會下降，我折衷地將其設為 512。

Learning Curves (1%)

Plot the learning curves (ROUGE versus training steps)

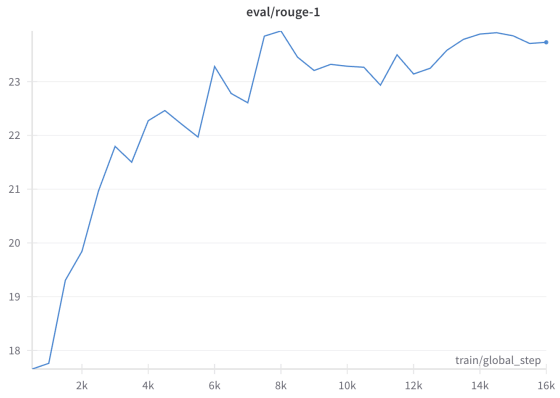


圖3：rouge-1 score 折線圖

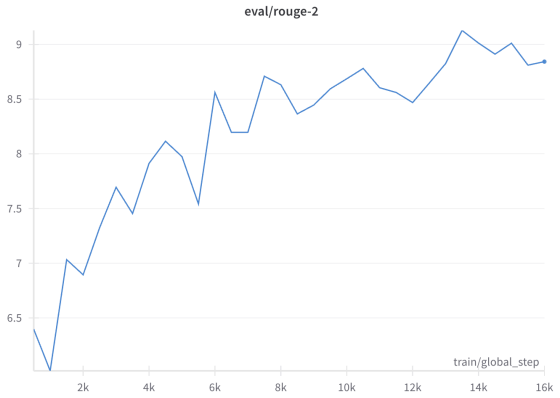


圖4：rouge-2 score 折線圖

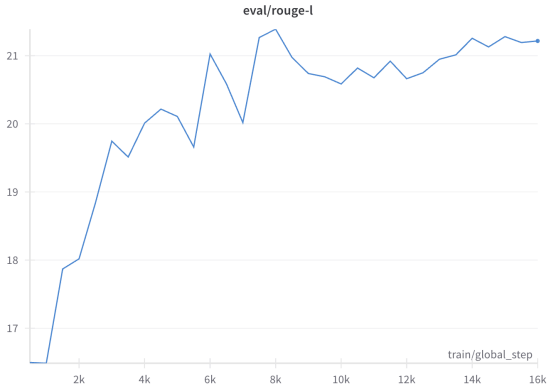


圖5：rouge-L score 折線圖

Q3: Generation Strategies(6%)

Strategies (2%)

Describe the detail of the following generation strategies

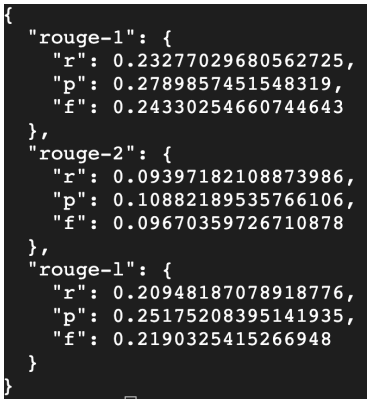
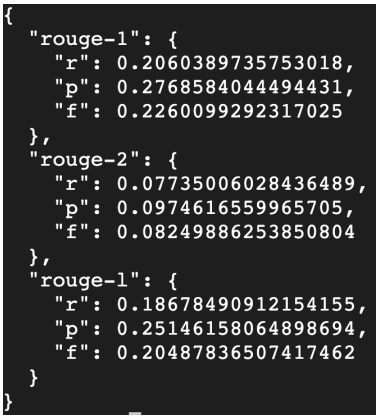
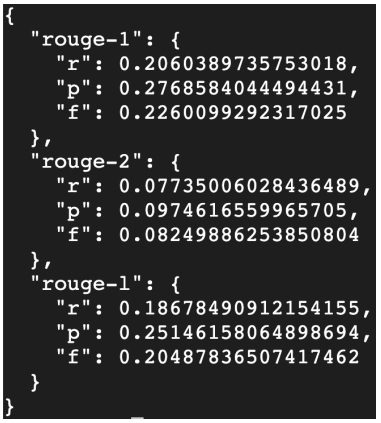
策略	流程	優點	缺點
Greedy	最簡單的生成策略。每一步生成時，都選擇當前機率最高的下一個詞，然後將其作為輸出的一部分，並將其作為後續生成的輸入。	簡單快速：計算量小，速度快。 局部最佳：每一步都選擇當前最有可能的詞。	錯過全局最佳解：因為只考慮當前步驟，可能導致整體結果不是最優的。 缺乏多樣性：生成的文本可能較為單調，缺乏創意。
Beam Search	一種平衡效率和效果的生成策略。每一步保留一定數量的序列（beam），而不是只保留機率最高的一個。細節流程如下： 1. 初始化：開始時，只有一個空的部分序列。 2. 擴展：在每一步，對每個部分序列，計算所有可能的下一個詞，並擴展成新的序列。 3. 截斷：在新生成的序列中，根據總概率選擇前 K 個（K 為束寬）序列，捨棄其餘的。 4. 重複：重複擴展和截斷，直到生成結束符或達到最大長度。	全局考慮：能夠考慮到更多的可能性，尋找更好的整體序列。 可調節：束寬越大，考慮的可能性越多，但計算量也越大。	計算成本：比貪婪搜尋需要更多的計算資源。 仍有局限：可能仍然無法探索到所有可能的序列。
Top-k Sampling	Top-k Sampling 是一種引入隨機性的生成策略，用於增加生成文本的多樣性。它在每一步中，從機率最高的前 K 個詞中隨機採樣下一個詞。細節流程如下： 1. 排序詞彙表：根據每個詞的機率，從高到低排序。 2. 選取前 K 個詞：僅考慮機率最高的 K 個詞，其餘的詞的機率設為 0。 3. 重新正規化：將剩餘的機率重新正規化，使其總和為 1。 4. 隨機採樣：根據新的機率分佈，隨機選擇下一個詞。	增加多樣性：因為引入了隨機性，生成的文本可能會有不同的變化。 控制隨機程度：通過調整 K 的大小，可以控制多樣性的程度。	可能產生不合理的詞：如果 K 值過大，可能會選到不合適的詞。
Top-p Sampling	Top-p Sampling 也是一種隨機採樣策略，但與 Top-k 不同的是，它根據累積機率來選擇候選詞。細節流程如下：	自適應候選集大小：根據輸出分佈動態調整候選詞的數量。	計算複雜度稍高：需要計算累積機率。

	1. 排序詞彙表：根據每個詞的機率，從高到低排序。 2. 計算累積機率：從機率最高的詞開始，累加機率，直到累積機率達到或超過預設的閾值 p（例如 0.9）。 3. 選取候選詞：將累積機率範圍內的詞作為候選詞，其餘詞的機率設為 0。 4. 重新正規化：將候選詞的機率重新正規化，使其總和為 1。 5. 隨機採樣：根據新的機率分佈，隨機選擇下一個詞。	更平衡：相比 Top-k，能更好地權衡機率質量和多樣性。	
Temperature	溫度是一種調節模型輸出機率分佈的方法，用於控制生成文本的隨機性。 調整機率分佈：在生成時，將模型輸出的 logits 除以一個溫度參數 T（$T > 0$）。 $T < 1$：降低溫度，機率分佈更尖銳，模型更傾向於選擇高機率的詞，隨機性降低。 $T > 1$：提高溫度，機率分佈更平坦，低機率的詞有更大的機會被選擇，隨機性增加。	靈活性：可以與其他策略結合使用，如 Top-k 或 Top-p 採樣。 控制文本質量與多樣性：調節溫度可以在保證文本合理性的同時，增加多樣性。	極端值的影響： $T \to 0$：模型會變得極為貪婪，只選擇機率最高的詞。 $T \to \infty$：機率分佈變得均勻，所有詞的機率相同，生成的文本可能不連貫。

Hyperparameters (4%)

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

表格請見下頁。

Beams Search (num_beams=4)	Top-K Sampling	Greedy
		
predict_results = trainer.predict(predict_dataset, metric_key_prefix="predict", max_length=data_args.val_max_target_length, num_beams=4,)	predict_results = trainer.predict(predict_dataset, max_length=data_args.val_max_target_length, do_sample=True, top_k=50, num_beams=1, metric_key_prefix="predict")	predict_results = trainer.predict(predict_dataset, metric_key_prefix="predict", max_length=data_args.val_max_target_length, num_beams=1,)

我最後使用的策略是單純的 Beam Search, 並且束寬 (num_beams) 4, 因為他是所有我嘗試的策略中得出最佳結果的策略。