Applied Deep Learning, Spring 2022 - Homework 3

郭柏志 R09521205

Q1: Model

1. Model:


```json
{
  "_name_or_path": "google/mt5-small",
  "architectures": [
    "MT5ForConditionalGeneration"
  ],
  "d_ff": 1024,
  "d_kv": 64,
  "d_model": 512,
  "decoder_start_token_id": 0,
  "dropout_rate": 0.1,
  "eos_token_id": 1,
  "feed_forward_proj": "gated-gelu",
  "initializer_factor": 1.0,
  "is_encoder_decoder": true,
  "layer_norm_epsilon": 1e-06,
  "model_type": "mt5",
  "num_decoder_layers": 8,
  "num_heads": 6,
  "num_layers": 8,
  "pad_token_id": 0,
  "relative_attention_num_buckets": 32,
  "tie_word_embeddings": false,
  "tokenizer_class": "T5Tokenizer",
  "torch_dtype": "float32",
  "transformers_version": "4.17.0",
  "use_cache": true,
  "vocab_size": 250100
}
```


Figure 1: mt5-small config.json

mt5 為 multilingual ransfer Text-to-Text Transformer 的簡寫，mt5 是一個 encoder-decoder 架構的 multi-task pre-training language model，只要任務資料的 source 與 target 轉換成 text to text 的形式，T5/mt5 就能同時 pre-train 多種不同的任務，因此單一 t5/mt5 模型在處理處理多種下游任務時只需要少量的訓練資料就能達到很好的預測結果。

mt5 的模型架構為 encoder-decoder Transformer，輸入句經過 tokenize 通過 emddeing layer 後轉為 embedding vector，embedding vector 通過 encoder 輸出的 hidden state 後續作為 key and value 用來與 decoder layer 中的 hidden state 計算 attention，decoder 則是將前一步的 output 作為當前的 input，在中間的 attention layer 與 encoder 輸出的 hidden state 計算 attention，decoder 輸出的 hidden state 會再經過 linear layer 與 softmax 轉為長度為字典大小的機率分佈，接著透過如 greedy, beam search 等 decode strategy 產生句子。在 summarization task 中 input 為 maintext，output 為 title (summarization)。

Encoder:

$$X = input\ context$$
$$input\ embedding = embedding\ layer(X)$$
$$hidden\ state_{encoder} = Transformer\ Encoder(input\ embedding)$$

Decoder:

$$input[1:t] = output[0:t-1]$$
$$hidden\ state_{decoder} = masked\ self\ attention(input)$$
$$hidden\ state_{decoder} = attention\ layer(hidden\ state_{decoder}, hidden\ state_{encoder},)$$
$$hidden\ state_{decoder} = Linear(hidden\ state_{decoder})$$
$$Logit = SoftMax(hidden\ state_{decoder})$$
$$output[t] = Decode\ Strategy(Logit)$$

2. Preprocessing:

mt5 tokenizer 透過 SentencePiece 處理 input(maintext) 與 output(title) 斷詞，輸入最大長度設定為 256，當輸入長度超過 256 時會將後面的文字截斷，若長度不足 256 則會補上 <pad> token，輸出最大長度設定為 64，當輸出長度不足 64 ，對於訓練資料若長度不足 64 則會補上 <pad> token，再將 <pad> token id 換成 -100 以在訓練時忽略不計 <pad> token 的 loss。

Q2: Training

1. Hyperparameter:

- max source length: 256
- max target length: 64
- per device train batch size: 4
- learning rate: 5e-5
- gradient accumulation steps: 4
- learning rate scheduler type: linear
- training epochs: 12

2. Learning Curves:

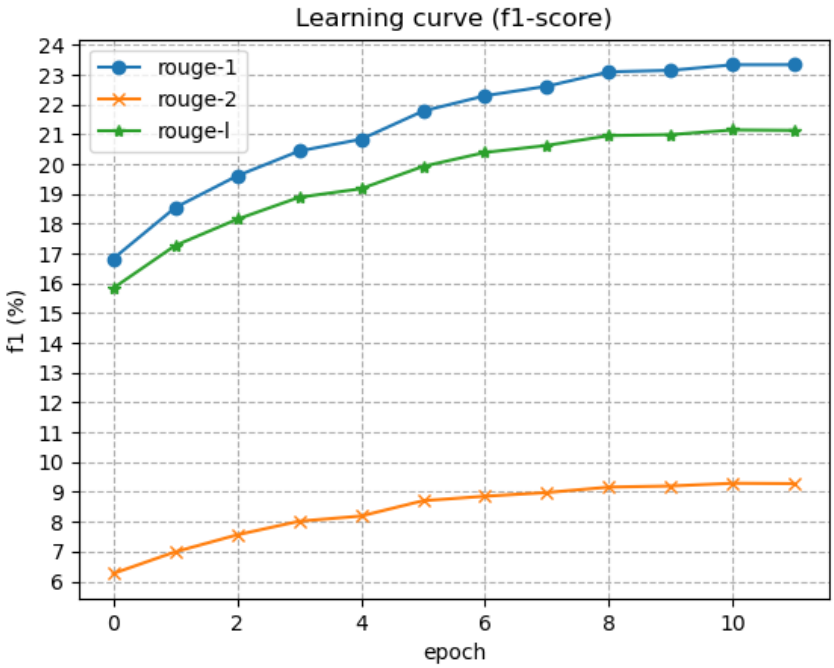


Figure 2: learning curve (rouge-f1)

Q3: Generation Strategies

1. Stratgies:

- **Greedy:** decoder 每個時間點都選擇當下機率最大的字作為預測的字，缺點在於每個時間點產生的字無法考慮到全局，當下生成的字可能與整個句子的語意沒有相關，而導致全局分數較低。

- **beam search:** 每個時間點追蹤目前最高分的 k 條 sequence，期望透過追蹤多條 sequence 能找到全局分數更高的 sequence。beams size 愈小，追蹤的 sequence 數量域少，愈接近 Greedy 的結果，beam size 愈大，選出的 sequence 愈接近在 dataset 中常出現的字，但可能因此選到跟當下 condition 關聯性較小的字。

- **Top-k Sampling:** Greedy 與 beam search 選擇的字通常還是字典中出現機率偏高的字，但人類說話時並不會刻意選擇最常出現的字，因此在選擇字時加入的隨機性，選擇當下機率分佈最高的前 k 個字隨機 sample，k 值愈低愈接近 Greedy，產生的字會是比較安全的但也比較容易局限在字典中出現機率教高的字，k 值愈高產生的字愈多樣，但同時選到與內容無關且在字典中出現機率偏低的字的機率也會提高。

- **Top-p Sampling:** Top-k Sampling 的機率分佈為 Narrow Distribution 時可能會把一些與 topic 毫無關聯的字包進 top k，當這些字被選中，後面的字也在生成的時候也會跟著被影響，而當機率分佈為 Broad Distribution 時則可能侷限了可以選擇的字，因此 Top-p 設定一個 threshold，只動態地採機率值累加超過這個 threshold 的前幾個字出來 sample，就能解決上述 top-k 的問題。

- **Temperature:** Temperature 並不是一種 decoding 的方法，它是一個調整機率密度函數的參數，當 temperature 愈高，機率分佈就會愈平緩均勻，temperature 愈低，機率分佈就會愈集中。

2. Hyperparameters:

Greedy

Greedy 其實可以視為 Beam Search beam number = 1 時的特例，可以發現 Greedy 的分數較低，原因是 Greedy 指能夠挑出局部最佳的單個字詞但無法找到全局最佳的文字組合。


```
{
    "rouge-1": {
        "r": 0.19935899221730402,
        "p": 0.27601886241984785,
        "f": 0.22105171510582355
    },
    "rouge-2": {
        "r": 0.07541032527575012,
        "p": 0.09733106745583633,
        "f": 0.08114478365968142
    },
    "rouge-l": {
        "r": 0.18068720470309763,
        "p": 0.25116652504661346,
        "f": 0.2004810324679967
    }
}
```


Figure 3: Greedy

Beam Search

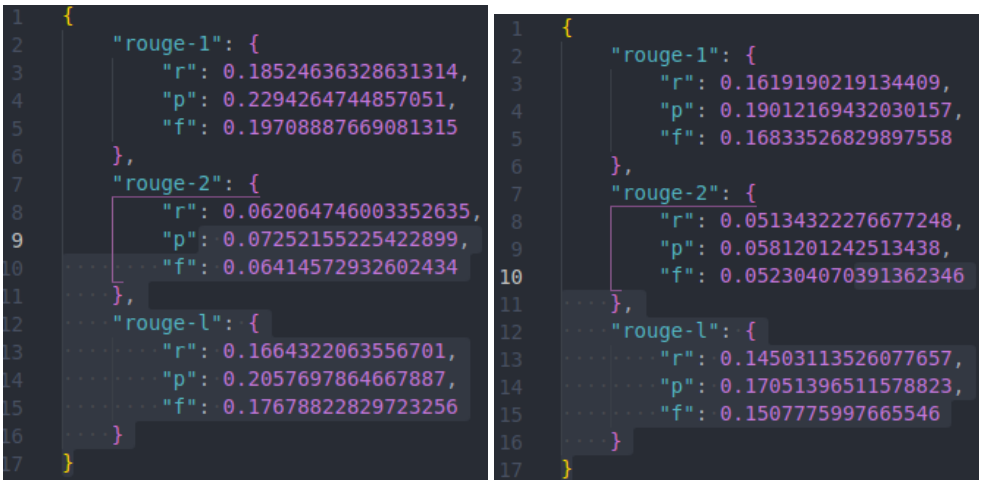
beam number = 5 與 beam number = 10 的結果相差不大，但在所有策略中 Beam Search 是結果最好的，因為其有較大的機會在全局找到更佳的文字組合。

(a) beam number = 5 (b) beam number = 10

Figure 4: Beam Search

Top-K Sampling

Top-K Sampling 是所有策略中分數最低的，其中當 top k 值愈大時分數愈低，原因可能來自 sample 的隨機性造成 decode 時容易選到與整 topic 沒有關係的字或是在字典中出現頻率低的字會導致後續的 decoding 更難以選出貼近 topic 的字，當 top k 愈大時此現象造成的影響也會跟著被放大。

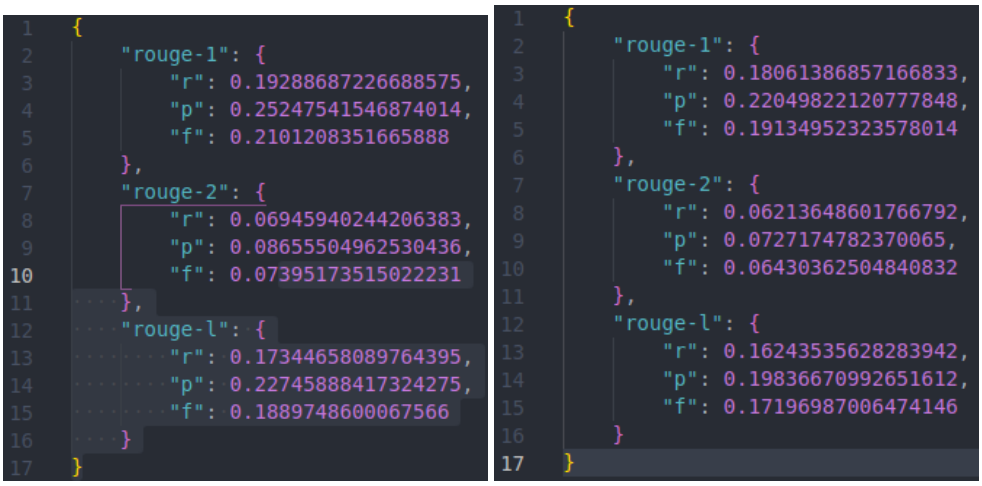


(a) top k = 10 (b) top k = 50

Figure 5: Top-K Sampling

Top-p Sampling

Top-p Sampling 雖然也是 sampling 的策略，但其分數好於 Top-K Sampling，因 Top-p Sampling 能動態地採機率值累加 top p 這個 threshold 的字來 sampling，能減低 Narrow Distribution 與 Broad Distribution 對 Top-K Sampling 的影響。

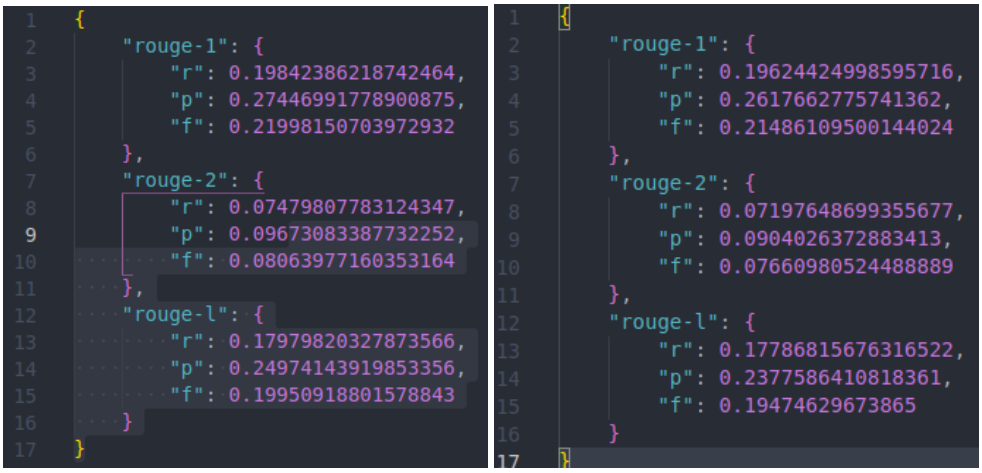


(a) top p = 0.4 (b) top p = 0.8

Figure 6: Top-P Sampling

Temperature

我採用 Top-p Sampling, top p = 0.4 來比較 Temperature 的影響，可以發現 temperature 較低時的分數較高一點，亦即機率分佈愈集中時對這個任務的效果是比較好的。



(a) temperature = 0.4 (b) temperature = 0.8

Figure 7: Temperature (top p = 0.4)