ADL HW3 Report

R10922124 林家毅

Q1

Model

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

Config:

{
"_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_max_distance": 128,
"relative_attention_num_buckets": 32,
"tie_word_embeddings": false,
"tokenizer_class": "T5Tokenizer",
"torch_dtype": "float32",

"transformers_version": "4.18.0",
"use_cache": true,
"vocab_size": 250100
}

mT5 是 encoder-decoder transformer 架構，同時具有 bidirectional attention encoder 以及 auto-regressive generating decoder 兩個好處，適合用在各種 sequence to sequence 的 tasks，只需要在 fine-tune 時加上 prompt prefix，例如 “summarize: ”

Preprocessing

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

google/mt5-small 的 tokenizer 是 SentencePiece tokenizer，是基於 Unigram segmentation algorithm，將 sentence 切成 subword 來做 tokenize。

Q2

Hyperparameter

Describe your hyper-parameter you use and how you decide it.

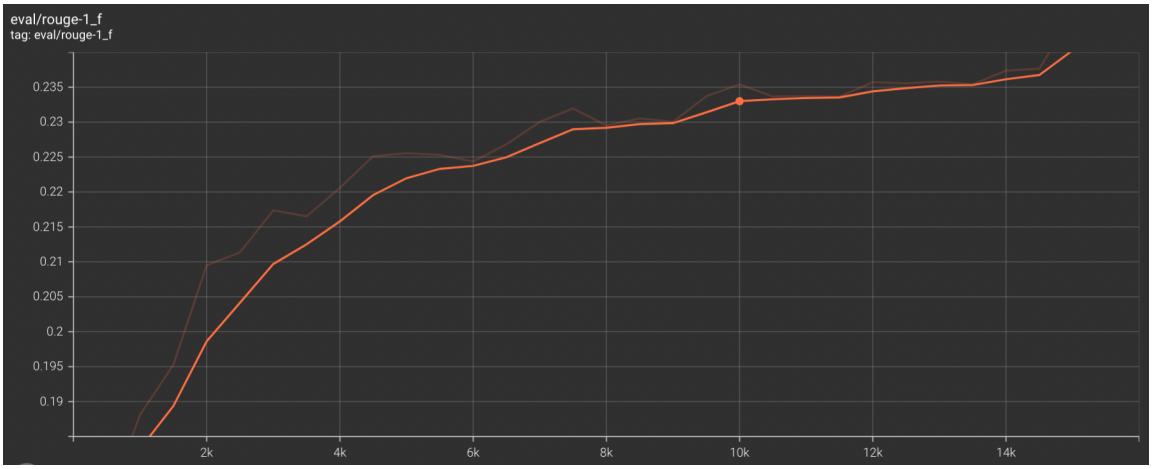
- Learning rate: 5e-5
- Total update steps: 35000 (經過測試覺得可以穩定達到 baseline)
- Training batch size: 4 (不設太大避免 CUDA out of memory)
- Optimizing algorithm: adafactor (節省 memory)
- Max source length: 256 (節省 memory)
- Max target length: 64 (節省 memory)

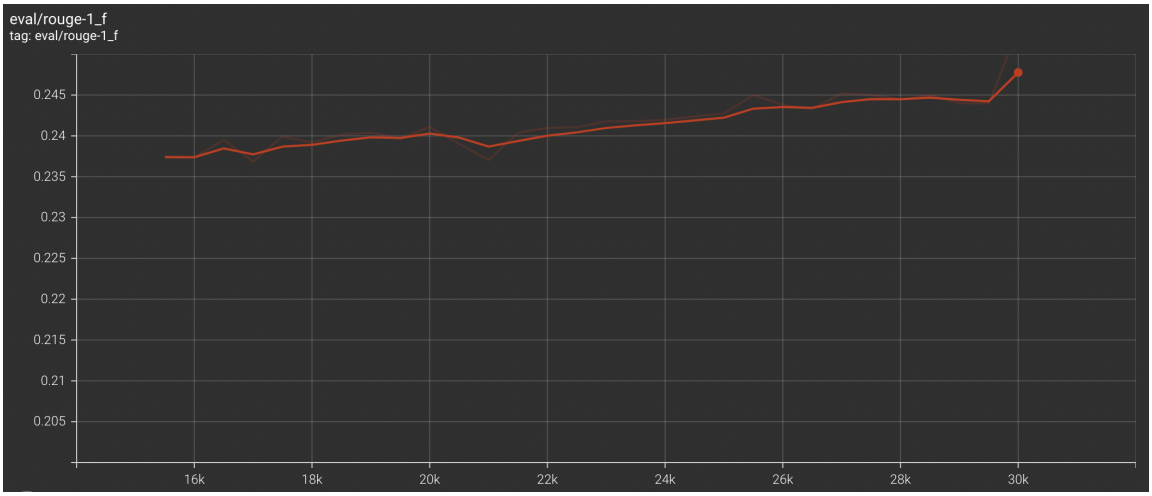
Learning Curves

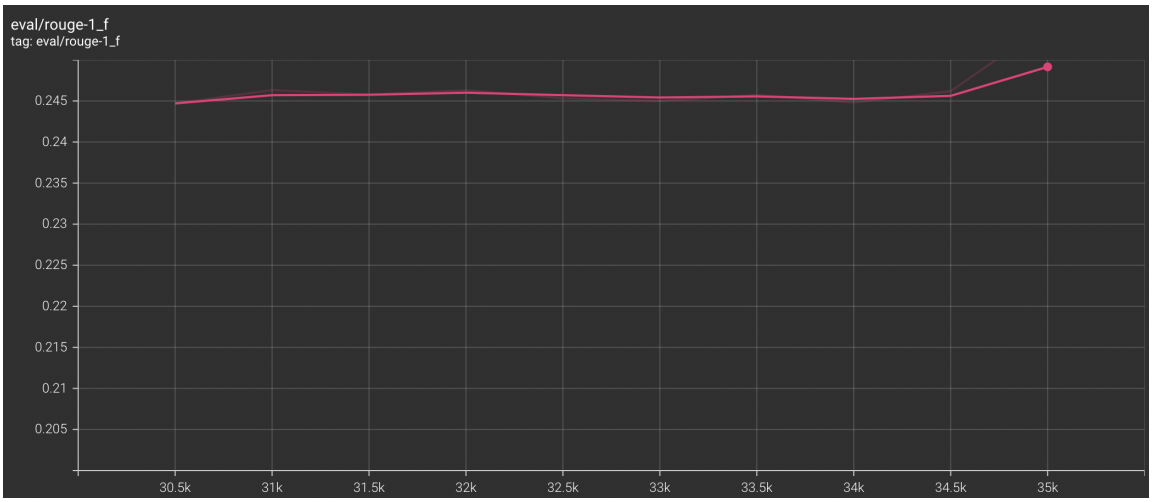
Plot the learning curves (ROUGE versus training steps)

每 500 steps 做一次 evaluation，整個 training process 共 35000 steps，F1-scores 如下 (因為分成三次做 training，因此每種 score 有三張圖片)

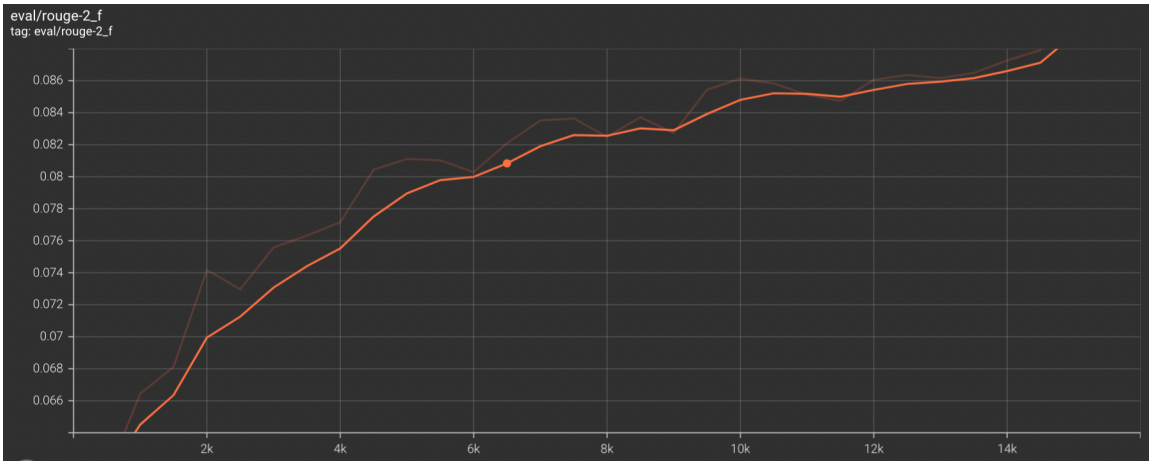
- Rouge-1

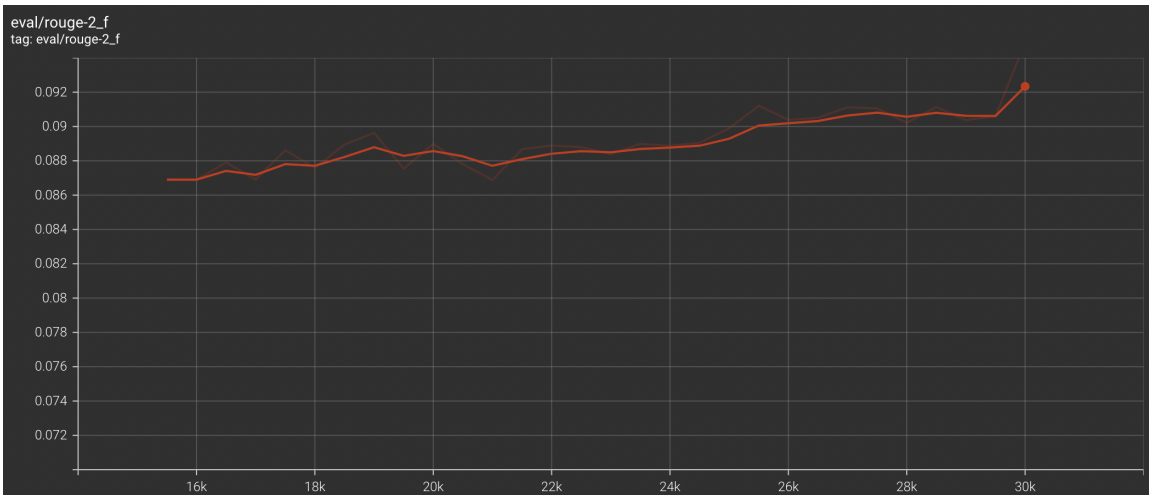

eval/rouge-1_f
tag: eval/rouge-1_f
0.235
0.23
0.225
0.22
0.215
0.21
0.205
0.2
0.195
0.19
2k
4k
6k
8k
10k
12k
14k

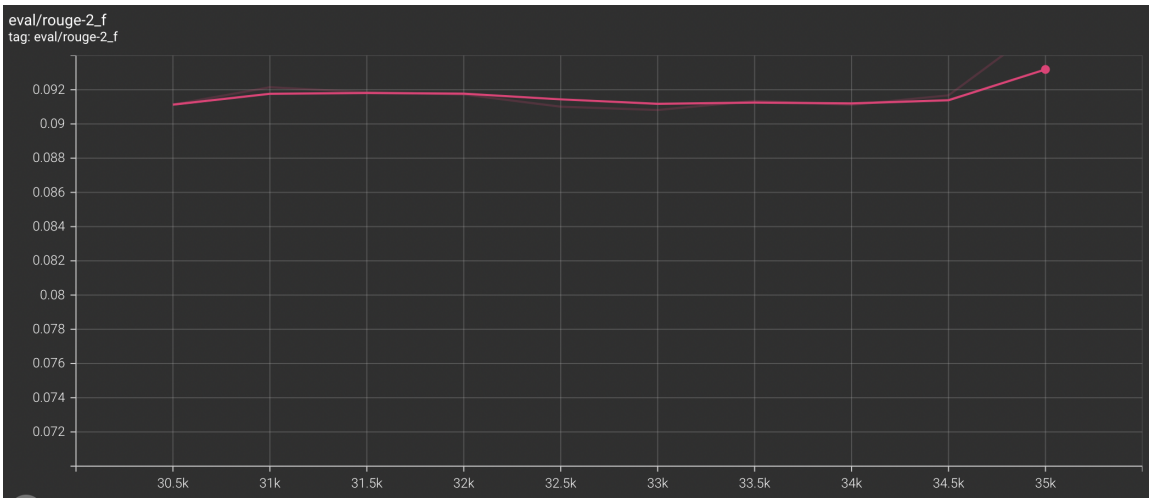

eval/rouge-1_f
tag: eval/rouge-1_f
0.245
0.24
0.235
0.23
0.225
0.22
0.215
0.21
0.205
16k
18k
20k
22k
24k
26k
28k
30k


eval/rouge-1_f
tag: eval/rouge-1_f
0.245
0.24
0.235
0.23
0.225
0.22
0.215
0.21
0.205
30.5k
31k
31.5k
32k
32.5k
33k
33.5k
34k
34.5k
35k

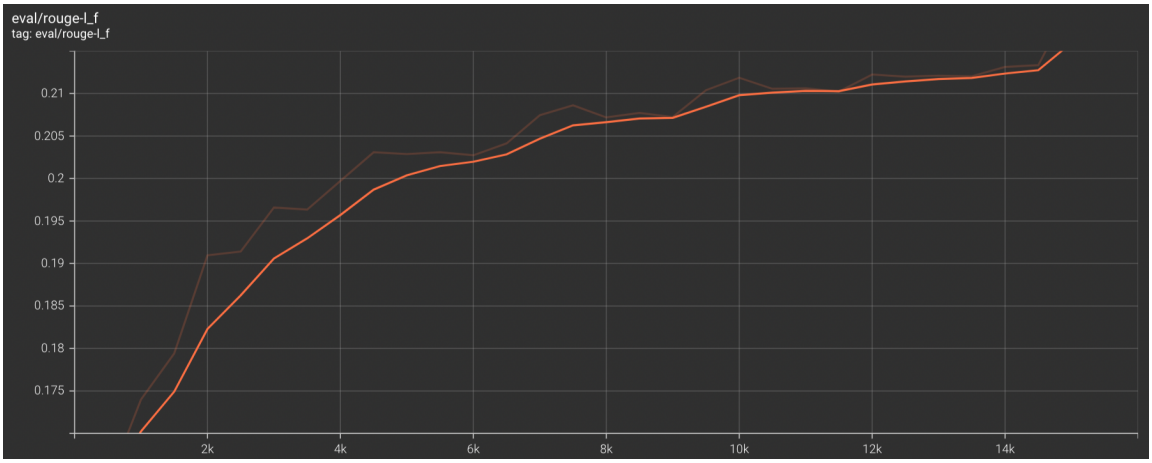
- Rouge-2

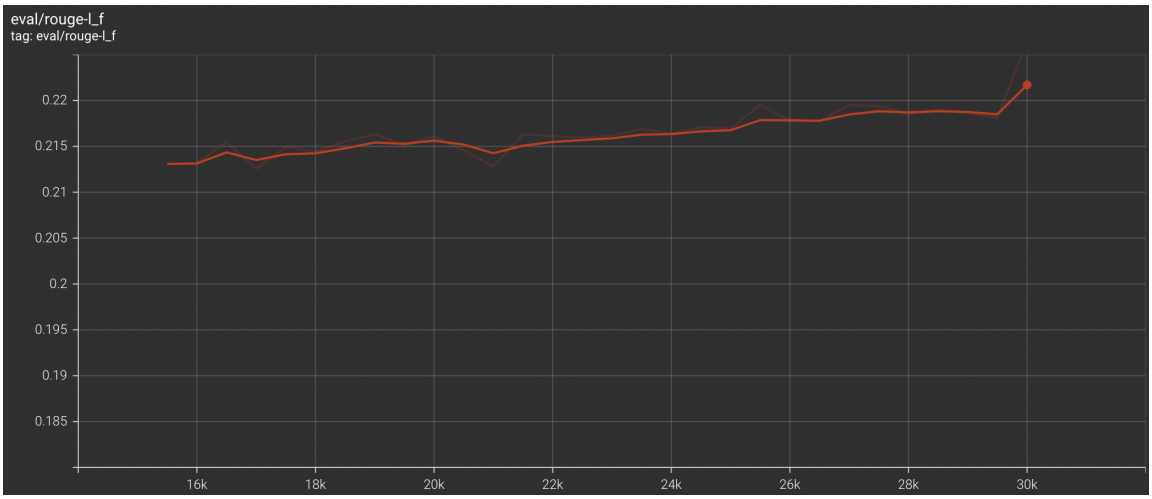

eval/rouge-2_f
tag: eval/rouge-2_f
0.086
0.084
0.082
0.08
0.078
0.076
0.074
0.072
0.07
0.068
0.066
2k
4k
6k
8k
10k
12k
14k

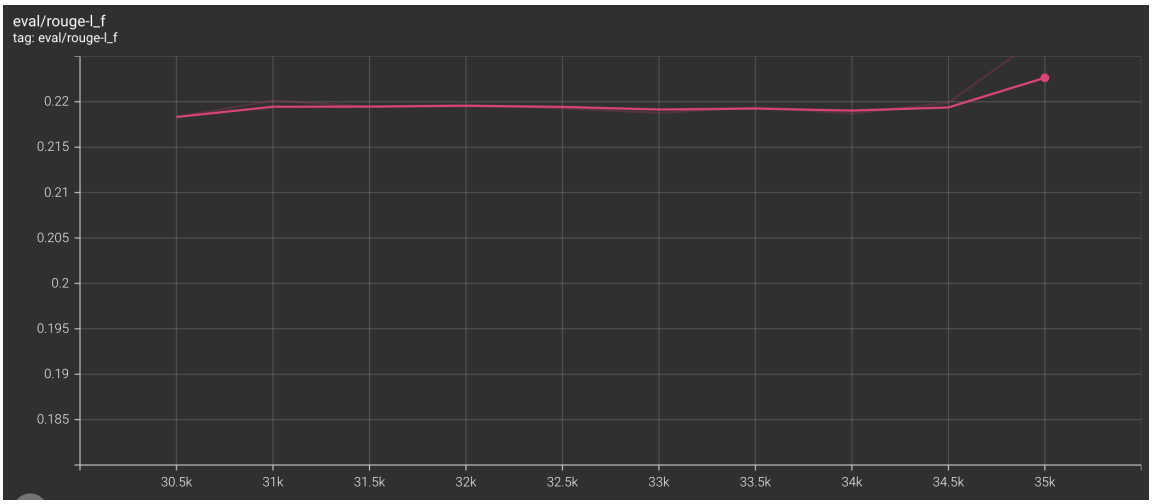

eval/rouge-2_f
tag: eval/rouge-2_f
0.092
0.09
0.088
0.086
0.084
0.082
0.08
0.078
0.076
0.074
0.072
16k
18k
20k
22k
24k
26k
28k
30k


eval/rouge-2_f
tag: eval/rouge-2_f
0.092
0.09
0.088
0.086
0.084
0.082
0.08
0.078
0.076
0.074
0.072
30.5k
31k
31.5k
32k
32.5k
33k
33.5k
34k
34.5k
35k

- Rouge-l







Q3

Strategies

Describe the detail of the following generation strategies:

- Greedy

  以 $P(w_t|w_1...w_{t-1})$ 最大者作為 w_t

- Beam Search

  每次都選擇 num_beams 個 $P(w_t|w_1...w_{t-1})$ 最大的當作候選，下一個 time step 再從這個 time step 的結果選擇 num_beams 個機率最大的候選，依此類推

- Top-k Sampling

  從 $P(w_t|w_1...w_{t-1})$ 最高的前 k 種可能 sample 出 w_t

- Top-p Sampling

  從 $P(w_t|w_1...w_{t-1})$ 最高的前幾種可能 (機率加總為 p) sample 出 w_t

- Temperature

  對 softmax 的結果做 sharpen or smoothing (<1 為 sharpen, >1 為 smoothing)

Hyperparameters

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

- Greedy

	Greedy	Sample
rouge-1, r	0.1947	0.1481
rouge-1, p	0.2694	0.1672
rouge-1, f	0.2174	0.1520
rouge-2, r	0.0690	0.0390
rouge-2, p	0.0884	0.0420
rouge-2, f	0.0747	0.0390
rouge-l, r	0.1769	0.1316
rouge-l, p	0.2454	0.1484
rouge-l, f	0.1976	0.1349

greedy 相較於 sample 的 performance 較好，因為 sample 可能會 sample 到不太好的選擇，導致最後 decode 出來的結果不好

- Beam Search

	Greedy	num_beams = 3	num_beams = 5

	Greedy	num_beams = 3	num_beams = 5
rouge-1, r	0.1947	0.2079	0.2165
rouge-1, p	0.2694	0.2622	0.2706
rouge-1, f	0.2174	0.2241	0.2322
rouge-2, r	0.0690	0.0757	0.0835
rouge-2, p	0.0884	0.0926	0.1010
rouge-2, f	0.0747	0.0801	0.0880
rouge-l, r	0.1769	0.1873	0.1956
rouge-l, p	0.2454	0.2365	0.2449
rouge-l, f	0.1976	0.2018	0.2098

beam search 的效果會比用 greedy 更好，因為會保留 num_beams 種最有可能的選擇，到最後才決定結果，而不像 greedy 總是選擇當下機率最大的；另外，num_beams 為 5 的時後會比為 3 的時候還要更好

- Top-k Sampling

	Greedy	top_k = 3	top_k = 5
rouge-1, r	0.1947	0.1914	0.1852
rouge-1, p	0.2694	0.2438	0.2300
rouge-1, f	0.2174	0.2071	0.1984
rouge-2, r	0.0690	0.0616	0.0579
rouge-2, p	0.0884	0.0730	0.0670
rouge-2, f	0.0747	0.0644	0.0599
rouge-l, r	0.1769	0.1702	0.1651
rouge-l, p	0.2454	0.2169	0.2050
rouge-l, f	0.1976	0.1841	0.1767

top_k sampling 對 performance 沒有明顯的幫助，performance 比用 greedy 差

- Top-p Sampling

	Greedy	top_p = 0.8	top_p = 0.9
rouge-1, r	0.1947	0.1673	0.1588
rouge-1, p	0.2694	0.1969	0.1838
rouge-1, f	0.2174	0.1750	0.1649

	Greedy	top_p = 0.8	top_p = 0.9
rouge-2, r	0.0690	0.0492	0.0455
rouge-2, p	0.0884	0.0549	0.0515
rouge-2, f	0.0747	0.0500	0.0466
rouge-l, r	0.1769	0.1484	0.1410
rouge-l, p	0.2454	0.1749	0.1634
rouge-l, f	0.1976	0.1552	0.1464

top_p sampling 對 performance 沒有明顯的幫助，performance 比用 greedy 差

- Temperature

	Greedy	temperature = 0.8	temperature = 1.2
rouge-1, r	0.1947	0.1671	0.1313
rouge-1, p	0.2694	0.1967	0.1412
rouge-1, f	0.2174	0.1751	0.1313
rouge-2, r	0.0690	0.0499	0.0305
rouge-2, p	0.0884	0.0563	0.0317
rouge-2, f	0.0747	0.0511	0.0300
rouge-l, r	0.1769	0.1491	0.1169
rouge-l, p	0.2454	0.1755	0.1259
rouge-l, f	0.1976	0.1561	0.1169

temperature 無論 <1 或 >1 都沒有得到比較好的結果，performance 比用 greedy 差

- Final generation strategy

  num_beams = 5，其他參數不調整