深度學習應用 - 作業三

tags: ADL

系級：電機所碩一　姓名：楊冠彥　學號：R11921091

Q1: Model

a. Model (1%)

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

根據助教作業說明，我使用Hugging Face的 google/mt5-small 模型，來做本次的Chinese News Summarization。mt5的模型涵蓋101種語言，且相似於T5。mt5的架構是一種標準的vanilla encoder-decoder transformer。

endoder會由無數相同的layer組成，每個layer會包括兩個sub-layer，一個用於multi-head self-attention機制，一個用於position-wise fully connected feed-forward 網路。另外，在每個sub-layer上都有殘差連接和layer normalization。

至於decoder的部分也由許多相同的layer組成。除了殘差連接和normalization這兩個sub-layer外，decoder層還插入了第三個sub-layer，這個sub-layer會對encoder的輸出執行multi-head attention。另外，在這個摘要產生title的任務中，一個簡單的linear layer被加到decoder的最上層，用來計算所有下一個可能token的logits。

config.json


```json
{
  "_name_or_path": "google/mt5-small",
  "architectures": [
    "MT5ForConditionalGeneration"
  ],
  "d_ff": 1024,
  "d_kv": 64,
  "d_model": 512,
  "decoder_start_token_id": 0,
  "dense_act_fn": "gelu_new",
  "dropout_rate": 0.1,
  "eos_token_id": 1,
  "feed_forward_proj": "gated-gelu",
  "initializer_factor": 1.0,
  "is_encoder_decoder": true,
  "is_gated_act": 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": "4.22.2",
  "use_cache": true,
  "vocab_size": 250100
}
```


b. Preprocessing (1%)

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

mt5的 tokenizer 是透過 SentencePiece 處理來源texts(maintexts) 和目標texts(titles) 的斷詞。那因為這邊我設定輸入文字最大長度為384，所以當輸入長度超過 384 時會將後面的文字截斷，若長度不足 384 則會補 上 token，而輸出最大長度我設定為 64，因此當輸出長度不足64時，就會補[PAD] token補到64字。另外，在計算訓練損失時將忽略target sequence中的所有 [PAD]，並且不使用額外的資料清理。

Q2: Training

a. Hyperparameter (1%)

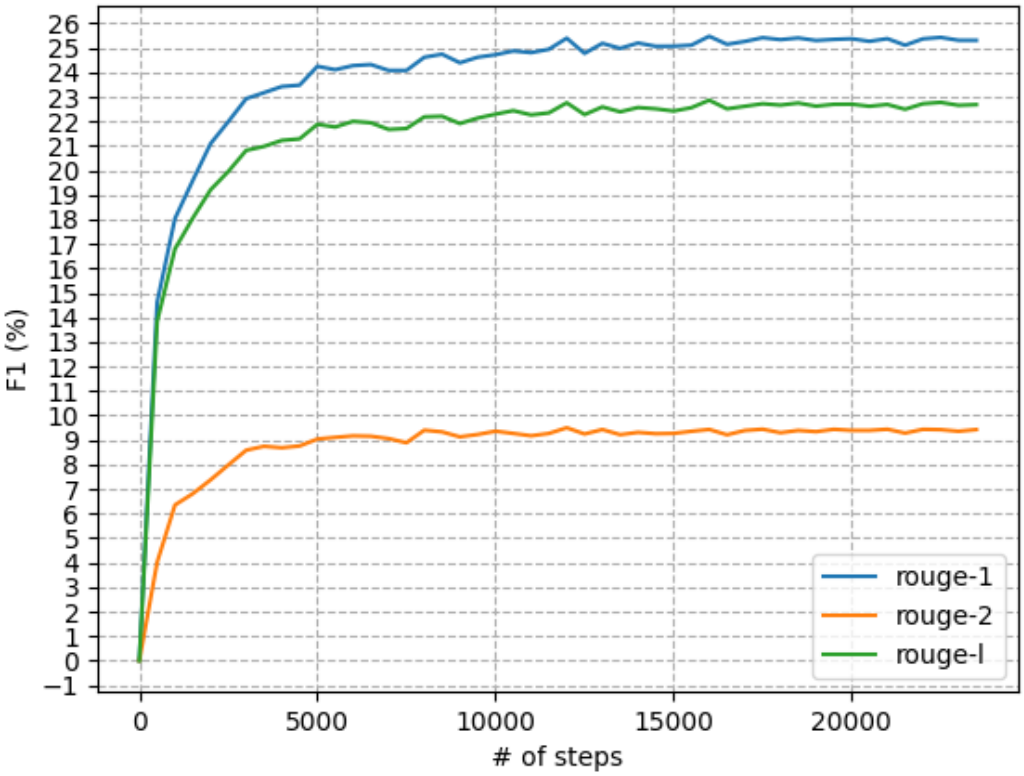
Describe your hyperparameter you use and how you decide it.

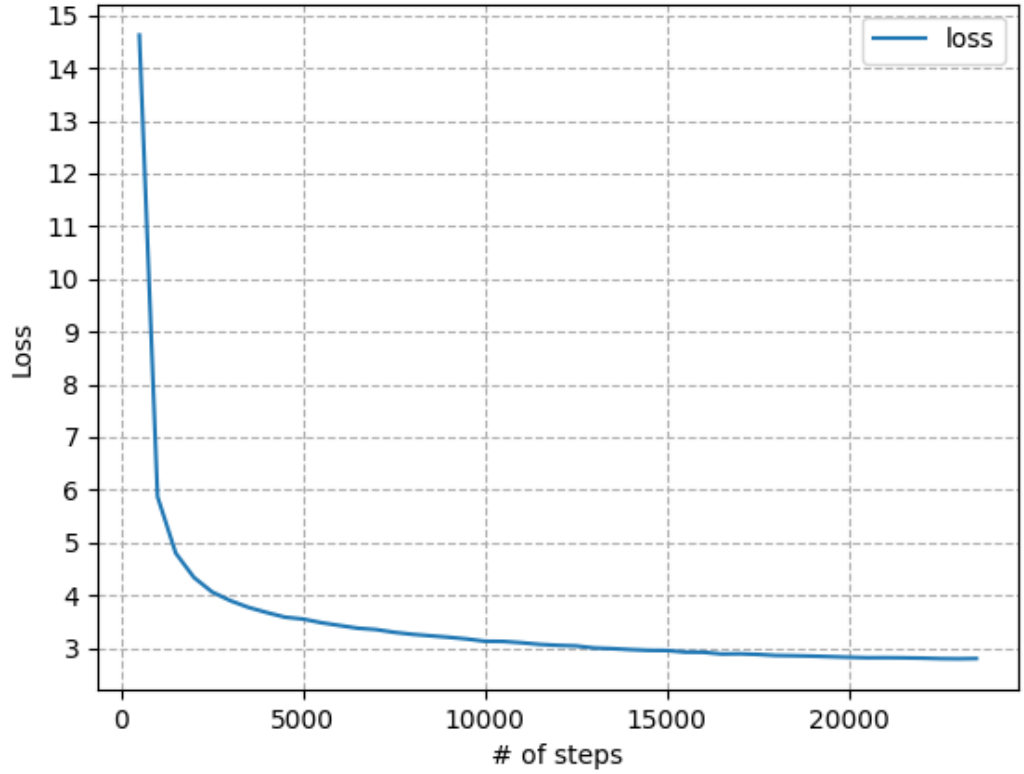
我選擇以下最為參數，因為我的GPU為 RTX3060 Laptop 僅有6G獨顯，因此我使用了助教投影片提及的adafactor做為optimization algorithm，相對於adam其減少了momentum的部分，因此記憶體使用較少，同樣由於GPU的限制，per_device_train_batch_size 最大只能設為2，learning rate一開始使用5e-5，但進步的非常慢，所以才更改為1e-4。

- Optimization algorithm: Adafactor
- Learning rate: 1e-4 = 0.0001
- Batch size: 2 (per_device_train_batch_size 2 * train_accumulation_steps 16)
- Num train epochs: 35
- max source length: 384
- max target length: 64
- learning rate scheduler type: linear
- warmup ratio: 0.1

b. Learning Curves (1%)

Plot the learning curves (ROUGE versus training steps)


F1 (%)
# of steps
rouge-1
rouge-2
rouge-l


Loss
# of steps
loss

Q3: Generation Strategies(6%)

a. Stratgies (2%)

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

Greedy

這項策略每次選擇機率最大的token作為輸出句子。一旦產生`<END>`，這個過程就會停止。

Beam Search

這項策略每個時間點觀察最高分的k條路徑，即candidate，下一個時間點，從這個時間點的k條中找出機率最大的那條，期望透過這樣相對於greedy更多的局部最佳擴展，即更大的搜尋空間，來找到全域最佳。值得注意的是當beam size=1，即k=1時，將等同於 greedy 搜尋。

Top-k Sampling

這項策略是每個時間點先從字典中找出目前預測機率最高的k個token，再將機率重新計算分布在這些token上，這個機率分布可寫成 $P(x|x_1 \dots x_{t-1})$ ，接著就是按機率取樣x_t 。

Top-p Sampling

這項策略是每個時間點先從字典中找出目前機率總和至少為 p 且數量最少的 $tokenset$，再將機率重新計算分布在這些 token 上，這個機率分布可寫成 $P(x|x_1 \dots x_{t-1})$ ，接著就是按機率從機率最大的前幾名累加到 p 的 sample 中取樣 x_t 。

Temperature

softmax是一種均勻化或集中化機率分佈的方法。應該根據我們是想要更集中於特定字詞，還是更多樣化的輸出來設置參數，tempetature愈高，機率分佈就會愈平緩，也就是會更加多樣化；temperature愈小，機率分佈就會愈集中，也就是愈偏向特定幾個token。機率公式如下：

$$P(x) = \frac{e^{s_w/\tau}}{\sum_{w' \in V} e^{s'_w/\tau}}$$

$\tau : temperature$

b. Hyperparameters (4%)

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

以下表格統一四捨五入到小數點後第五位，詳細可見繳交檔案中的statistic資料夾

Greedy vs Sampling

rouge	Greedy	do_sample
rouge-1_f	0.253	0.21077
rouge-1_p	0.28322	0.2221
rouge-1_r	0.24148	0.21126
rouge-2_f	0.0941	0.06945
rouge-2_p	0.1033	0.07216
rouge-2_r	0.09148	0.07095
rouge-l_f	0.22685	0.18643
rouge-l_p	0.25411	0.19673
rouge-l_r	0.21657	0.18682

Beam Search

rouge	num_beams=1 (Greedy)	num_beams=5	num_beams=6	num_beams=7	num_beam=8
rouge-1_f	0.253	0.26715	0.26781	0.26765	0.26713
rouge-1_p	0.28322	0.28798	0.28779	0.2873	0.28612
rouge-1_r	0.24148	0.2631	0.26408	0.26406	0.264
rouge-2_f	0.0941	0.1064	0.10729	0.1072	0.10713
rouge-2_p	0.1033	0.11353	0.11438	0.11428	0.11398
rouge-2_r	0.09148	0.1061	0.10693	0.10681	0.10691
rouge-l_f	0.22685	0.2379	0.23876	0.23839	0.23799
rouge-l_p	0.25411	0.25668	0.25684	0.25623	0.25522
rouge-l_r	0.21657	0.23427	0.23539	0.2351	0.23514

Top-k Sampling

rouge	k=30	k=60
rouge-1_f	0.21404	0.20434
rouge-1_p	0.22844	0.21607
rouge-1_r	0.21281	0.2039
rouge-2_f	0.0697	0.06492
rouge-2_p	0.07329	0.06785
rouge-2_r	0.07064	0.06581
rouge-l_f	0.18813	0.181
rouge-l_p	0.20113	0.19145
rouge-l_r	0.18706	0.18069

Top-p Sampling

rouge	p=0.5	p=1.0
rouge-1_f	0.24564	0.21077
rouge-1_p	0.26915	0.2221
rouge-1_r	0.2381	0.21126
rouge-2_f	0.08883	0.06945
rouge-2_p	0.09551	0.07216
rouge-2_r	0.08797	0.07095
rouge-l_f	0.2179	0.18643
rouge-l_p	0.23902	0.19673
rouge-l_r	0.21117	0.18682

Temparature (Sampling) + Beam Search(num_beams=6)

rouge	temparature=0.5	temperature=1.0
rouge-1_f	0.24279	0.26471
rouge-1_p	0.28267	0.28548
rouge-1_r	0.23003	0.26036
rouge-2_f	0.09211	0.10393
rouge-2_p	0.1061	0.11106
rouge-2_r	0.0887	0.10335
rouge-l_f	0.21879	0.23586
rouge-l_p	0.25534	0.25459
rouge-l_r	0.20665	0.23207

Final Generation Strategy

最後我使用 beam search(num_beams=6) 的策略。

Bonus: Applied RL on Summarization (2%)

a. Algorithm (1%)

Describe your RL algorithms, reward function, and hyperparameters.

我使用梯度策略去達到增強式學習，reward function則為rouge-l/0.22，hyperparameters則如下：

- Optimization algorithm: Adafactor
- Learning rate: 1e-3 = 0.001
- Batch size: 32 (per_device_train_batch_size 1 * train_accumulation_steps 32)
- Num train epochs: 3
- max source length: 384
- max target length: 64
- learning rate scheduler type: linear
- weight decay: 0.01

b. Compare to Supervised Learning (1%)

Observe the loss, ROUGE score and output texts, what differences can you find?

RL 的訓練目標與監督學習略有不同。最初的 finetuning 方法是在訓練資料中 fitting 每個 token 的產生。而 RL 方法是 fitting 產生句子的 rouge score 。下面的實驗結果是在 beam size = 6 的情況下測試的。 rouge 的分數則為 f1 score 。

Learning type	loss	rogue-1_f1	rouge-2_f1	rouge-l_f1
Supervised Learning	4.46413	0.22523	0.086	0.20565
Reinforcement Learning	3.11241	0.24031	0.09315	0.21669

在實驗中，RL 算法獲得了更顯著的rouge 分數的表現，在我觀察RL的輸出和finetuning後，也發現RL的輸出標題比另一個更符合人類的習慣。

Reference

- LeeMeng - 進擊的 BERT：NLP 界的巨人之力與遷移學習
- Transformer github
- Hugging Face
- Revisiting Pre-Trained Models for {C}hinese Natural Language Processing
- multilingual-t5 Github