HW2

學號：P12922005

姓名：林焜詳

Q1. Model

Model

Architecture

本次作業規定使用mT5（Xue等人，2020年）模型。mT5的架構與一般的Transformer Encoder and Decoder 非常相似，不同的部分是activation採用了gated-gelu，且在Pretrain時期不使用dropout。這邊使用mT5模型，詳細參數可看**Hyperparameters**。

Hyperparameters


```
{
  "_name_or_path": "google/mt5-small",
  "architectures": [
    "MT5ForConditionalGeneration"
  ],
  "classifier_dropout": 0.0,
  "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": "4.34.1",
  "use_cache": true,
  "vocab_size": 250112
}
```


How it works on Text Summarization

mT5模型是一個基於Transformer Encoder to Decoder 的多語言文本到文本生成的模型，藉由模型的生成能力，幫我們做到文本摘要的工作。

要如何應用在文本摘要任務上，我簡單的拆分成幾個步驟：

1. **預訓練（Pretraining）**：mT5在mC4數據集上進行預訓練，這個數據集包含了多種語言的大量網絡抓取的文本。預訓練的目標是文本到文本生成，即模型學會將一種形式的文本（例如問題）轉換成另一種形式的文本（例如答案）。
2. **理解和編碼（Understanding and Encoding）**：在文本摘要任務中，模型首先需要理解輸入的段落，這涉及到使用其編碼器部分來編碼文本，捕捉句子的語義和上下文。
3. **生成摘要（Summarization Generation）**：mT5的解碼器部分隨後基於編碼器的輸出來生成摘要。它一步一步地生成新的詞，每次都考慮到前面已經生成的內容，以確保摘要的連貫性和相關性。
4. **條件生成（Conditioned Generation）**：生成摘要時，模型被“條件化”來專注於輸入的段落，這意味著解碼器在生成文本時會嘗試保持輸入內容的主旨和重點。
5. **微調（Fine-tuning）**：雖然mT5已經在mC4數據集上進行了預訓練，但為了更好地執行特定任務如新聞摘要，它還需要在特定領域的數據上進行微調。在這個例子中，mT5在udn.com的新聞數據上進行了微調，這有助於模型更好地理解新聞文章的結構和常用語言。

藉由以上步驟，我們便可以訓練出一個不錯的摘要模型作為往後使用。

Preprocessing

我運用了預先訓練好的mT5分詞器來對數據進行分詞。這個分詞器將輸入的中文序列拆分成單個字符，接著根據其字典將每個字符轉換成對應的ID。

在完成分詞之後，分詞器會對序列進行截斷或填充，並在序列的開頭和結尾分別加上[BOS]（開始標記）和[EOS]（結束標記）。

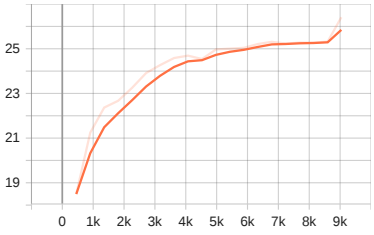
Q2. Training

Hyperparameters

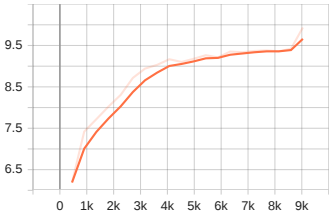
- Epoch: 20 → 嘗試起來是比較好的斷點，看起來在eval有筆記好的收斂結果，太少表現不夠好，太多則有點多餘了，improve不多。
- Batch size: 12 x 4 (累積梯度步數) → 充分運用算力，加速模型收斂速度
- Learning rate: 5e-5 → 5e-5是一般fine-tune會採用的大小
- Linear warm-up: 300步 → 隨機選一個合理的數
- Learning scheduler: cosine → 有更好的fine-tune效果

Learning Curves

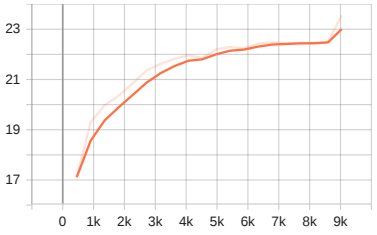
呈現的數值為 100 x rogue score



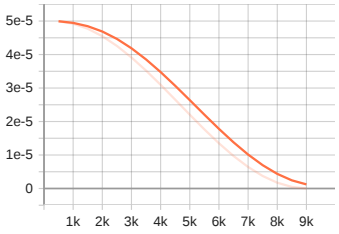
rouge-1 on validation set



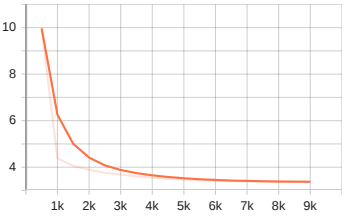
rouge-2 on validation set



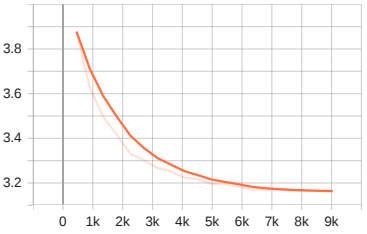
rouge-l on validation set



Learning rate



training loss



validation loss

Q3. Generation Strategies

Strategies

Greedy

Greedy會在每個字的輸出中選擇最高機率的那個作為輸出

Beam Search

Beam search是greedy search的一種改進算法。相較greedy擴大了搜索空間，但不至於如窮舉法大量，一般會使用beam size（k）選擇前k個當前最佳候選的詞組組合，每推衍一個新的字需保持k個候選詞組組合。

Top-k Sampling

Top-k sampling選取字的方式同greedy，但是在選之前會先將預測的logit作預處理，選top k個最好的詞預測機率，重新normalize成新的機率分佈。

Top-p Sampling

主要是要改善Top-k sampling在部分分布上會有無法適應的問題，top k 遇到平坦的distribution會讓選後的分布失去原本分不具有的分布特性。Top p 則改用總和閾值去動態取前k名作為新分部的運算。

Temperature

用以調整類型分佈不均衡的數據集會有的問題。藉由γ調整，若$\gamma = 1$，等於沒有調整，若$\gamma > 1$，則會放大小機率的樣本類別的機率，被採樣的機會會變高；若$\gamma < 1$，則反之。

$$p_i = \frac{N_i^{1/\gamma}}{\sum_{j \in [1,M]} N_j^{1/\gamma}}$$

Hyperparameters

Evaluation	rouge-1	rouge-2	rouge-l	rouge_combined	gen_len
GreedySearch	26.375	9.905	23.508	3.511	21.872
BeamSearch=3	27.687	11.032	24.509	3.752	23.452
BeamSearch=5	27.567	11.168	24.441	3.759	23.866
BeamSearch=7	27.344	11.056	24.214	3.725	24.206
Top-k=25	22.228	7.171	19.355	2.798	22.284
Top-k=50	20.720	6.466	18.065	2.584	22.411
Top-k=100	20.051	6.215	17.452	2.494	22.347
Top-p=0.8	23.202	8.023	20.398	2.994	22.111
Top-p=0.95	21.837	7.037	19.057	2.750	22.240
Top-p=0.95,T=0.5	25.625	9.417	22.642	3.377	21.832
Top-p=0.95,T=2.0	12.322	2.108	10.420	1.316	25.921

- 文字摘要是一個主觀且決定性的任務，所以其實表現上在指標上都不好，再仔細查看一些樣本的預測後，其實可以發現有些預測並不是答錯，只是可能換句描述，但是意思其實很接近，反而比較覺得像是對話任務會處理的不錯？
- 單使用一種方法，Beam search比其他的方式都好，整體上來看是比較聰明的做法。
- 另外若使用Top-p + Tempature的實驗可以發現，過度把溫度提高會讓預測分佈失效，因為新的分佈就長得很像均勻分布，對預測一點幫助都沒有。

Final generation strategy

最後選定Beam search 作為我的generation策略

Evaluation	rouge-1	rouge-2	rouge-l	rouge_combined	gen_len
BeamSearch=5	27.567	11.168	24.441	3.759	23.866