Q1: Model

Model

模型的架構我使用了hugginface上的Seq2SeqLM，其主要是由T5的encoder-decoder所組成的，只需要在每一篇文章的前面加上"summarize: "就可以讓T5完成summarization的task

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_num_buckets": 32,
  "tie_word_embeddings": false,
  "tokenizer_class": "T5Tokenizer",
  "torch_dtype": "float32",
  "transformers_version": "4.17.0",
  "use_cache": true,
  "vocab_size": 250100
}
```


Preprocessing

先在每一篇文章的前面加上prefix "summarize: "，之後tokenizer會將主要的文章切成tokens，切完之後padding或truncate成1024長度的sequence，而標題則是padding或truncate到128長度的sequence

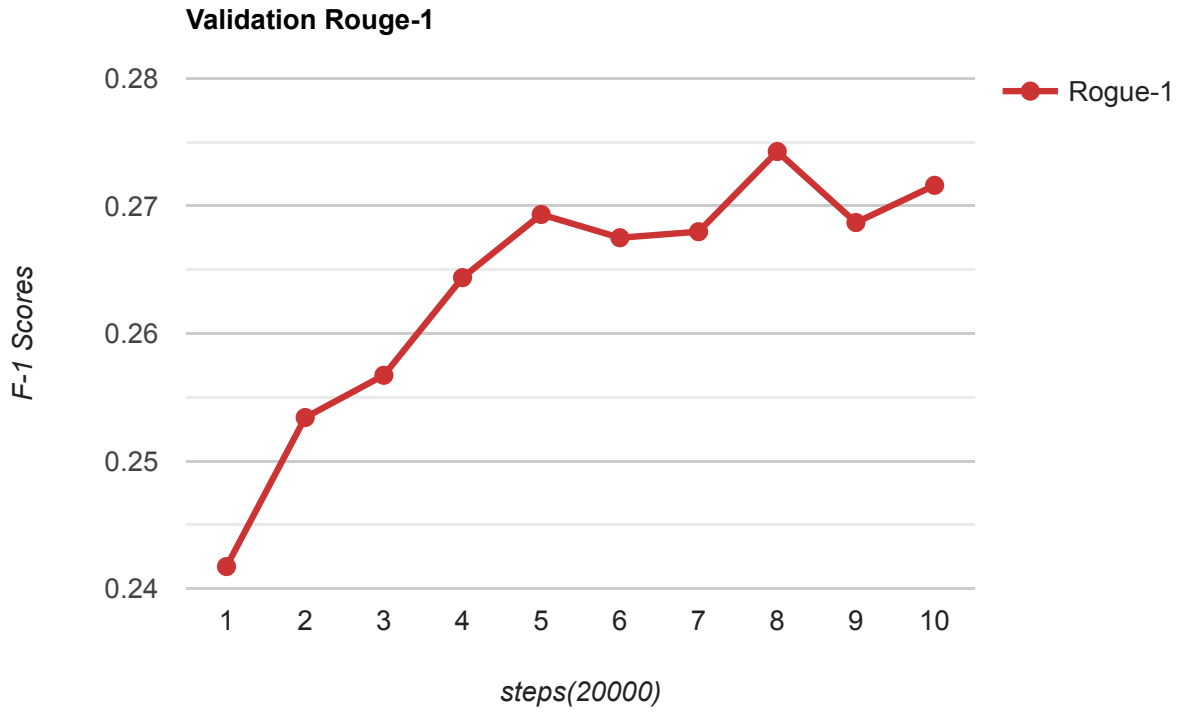
Q2: Training

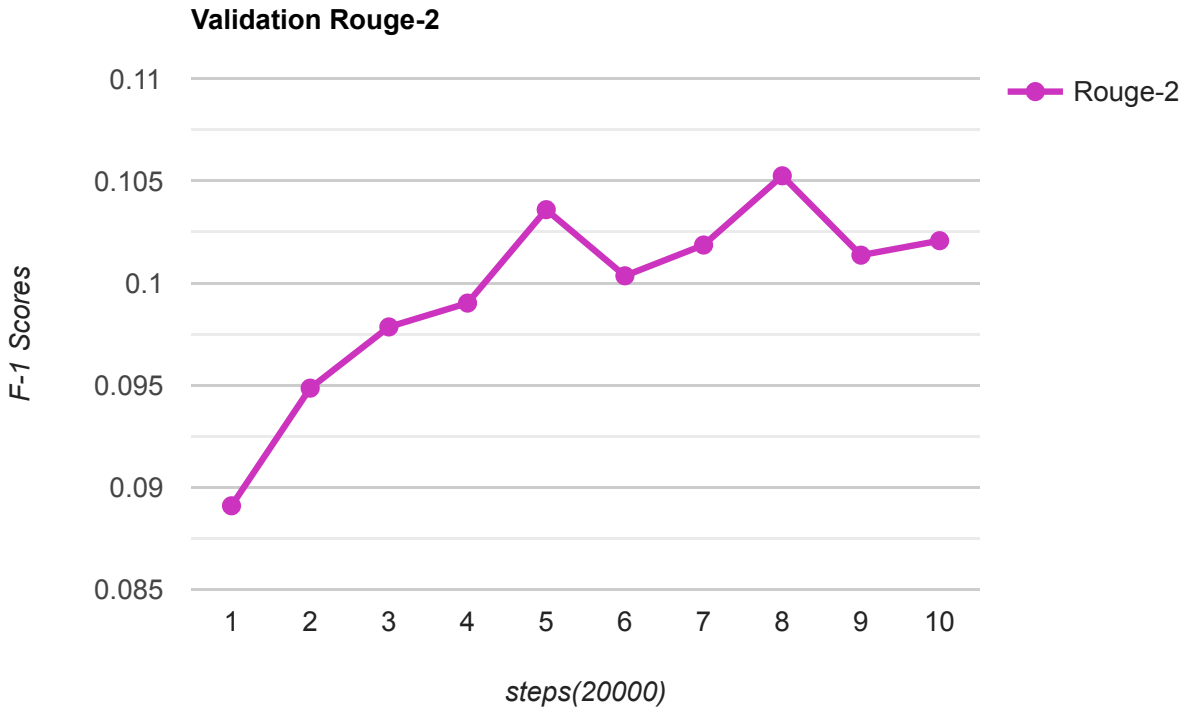
Hyperparameter

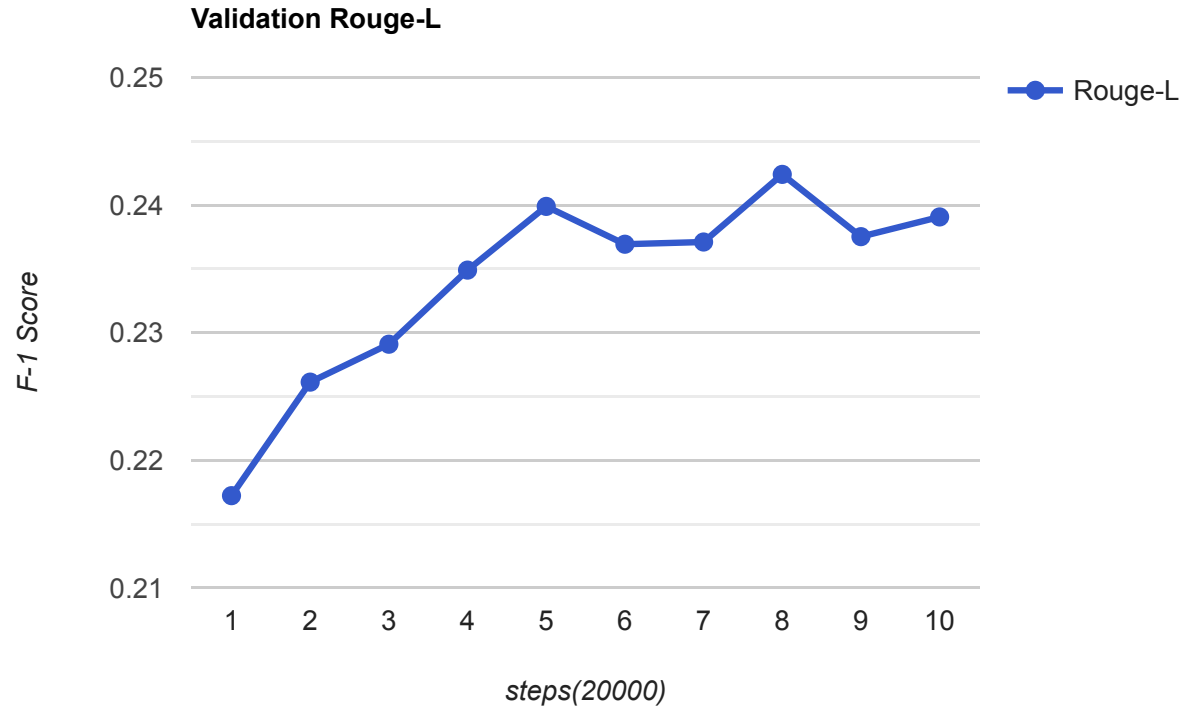
- **Optimizer:** Adam
- **Learning Rate:** 4e-5
- **Epochs:** 20
- **Batch Size:** 2 (per_gpu_train_batch_size 1 * gradient_accumulation_steps 2)

基於空間上的考量，我將batch size壓低，且為了觀察rouge的變化，讓epochs延長至20，大概train了20個小時，但其實大概4個epoch就可以過baseline了

Learning Curves







上面是將compute_metrics換成助教給的get_rouge套在validation dataset後的結果，generate strategy為預設的greedy

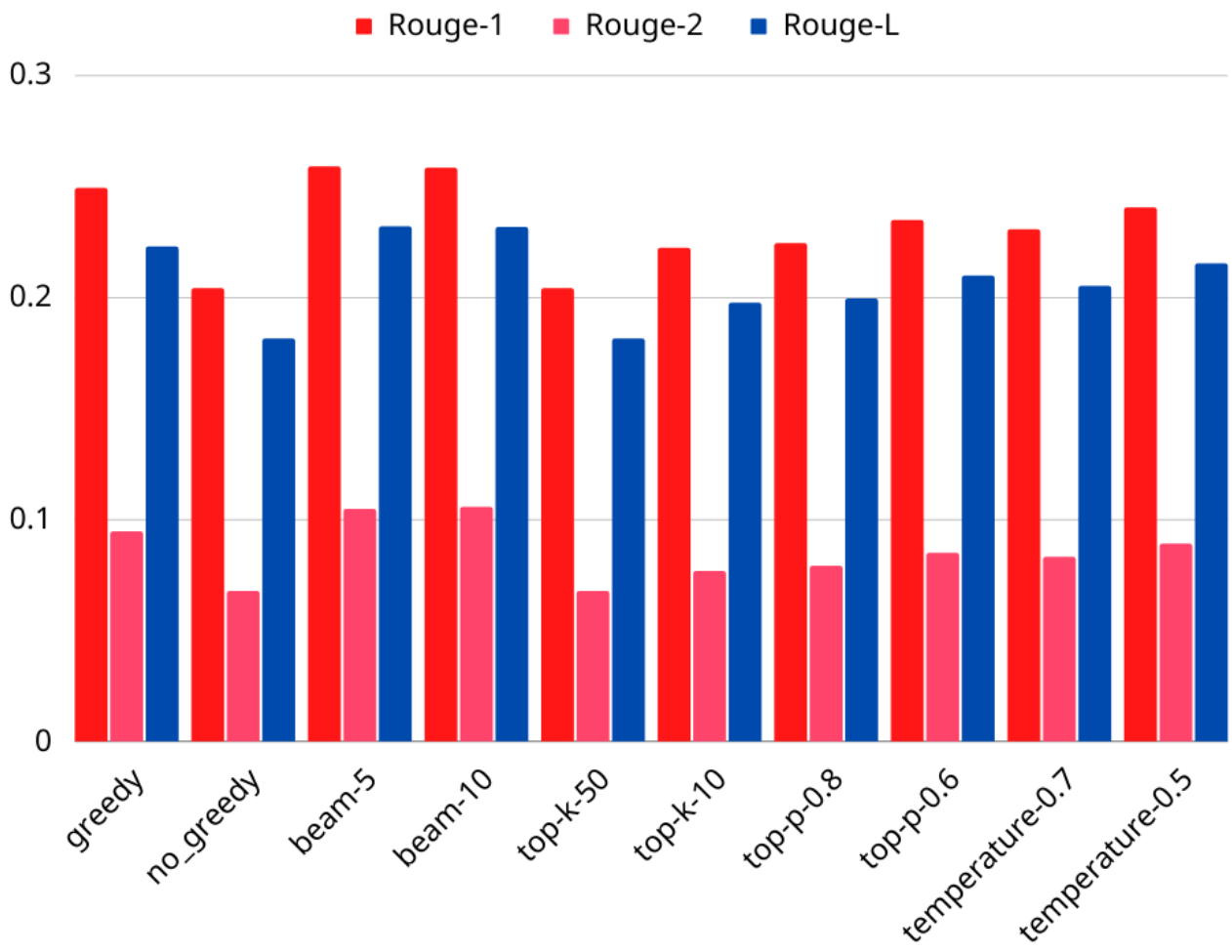
Q3: Generation Strategies

Stratgies

- **Greedy:** 每次預測都拿機率最大的word
- **Beam Search:** 紀錄前num_beams個最大的sequences，最後再從中選擇最大的
- **Top-k Sampling:** 每次sample出前k個最大的words，最後再重新分配機率，隨機從中選出word
- **Top-p Sampling:** 每次sample出總和機率剛好超過p的前幾大words，最後再重新分配機率，隨機從中選出word
- **Temperature:** 在計算softmax前將每一項都乘以t(0<t<1)，使得softmax完的分佈大的更大，小的更小

Hyperparameters

Strategy	parameter	Rouge-1	Rouge-2	Rouge-L
Greedy	Off	0.2039530095530	0.0677330203463	0.1813097078029
Greedy	On	0.2441508274611	0.0910778042630	0.2183053082929
Beam Search	5	0.2588432817909	0.1047904422755	0.2318888512388
Beam Search	10	0.2583083903714	0.1056036136550	0.2316008578000
Top-k Sampling	10	0.2222457883136	0.0767138758858	0.1974708899383
Top-k Sampling	50	0.2039530095530	0.0677330203463	0.1813097078029
Top-p Sampling	0.6	0.2347298908247	0.0848652542332	0.2097564484366
Top-p Sampling	0.8	0.2243164777285	0.0791240063926	0.1993601593659
Temperature	0.5	0.2404122507668	0.0890243282595	0.2151795065146
Temperature	0.7	0.2305121459105	0.0831232267147	0.2051488098084



上面是我各自試過不同演算法的結果，可以發現beam search在size為5的情況下表現會最好，因此我選擇拿beam search來當作我生成的演算法