

Applied Deep Learning HW3

r11944049 呂瑋浩

November 2022

1 Model and Preprocessing

使用google的mt5-small模型，在mc4上pretrain，涵蓋101種語言，基於T5 model，同樣為encoder-decoder架構，並在模型上微調，稱為**T5.1.1**，主要改進為：使用GeGLU替代ReLU、在Unlabeled data訓練時不使用Dropout。

mT5會使用Encoder將input輸出為Hidden state，Decoder根據Hidden state生成文字，在tokenizer上使用SentencePiece來達成跨語言的特性，其中它將Whitespace視為一個符號，像是”Hello world.”變成”Hello.world.”，這樣做的好處是能學到些語言上需要空格的特性。

在此次作業中，我使用了huggingface的script進行訓練，將jsonl轉為json後，使用資料中的”maintext” 與”title”欄位進行訓練。

```
{
  "_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": "4.23.1",
  "use_cache": true,
  "vocab_size": 250100
}
```

Figure 1: Configurations of mT5

2 Training

Training detail

- Context Selection
 - Batch size: 16 (`per_gpu_train_batch_size 2 * gradient_accumulation_steps 8`)
 - Num_train_epochs: 5
 - Learning_rate: 1e-3
 - Optimizer: adafactor

使用與HW2相同的Batch size，並且訓練5個Epoch(約四小時)，由於model架構較大，使用較高的Learning rate與adafactor來縮短所需的Epoch。

Learning curve

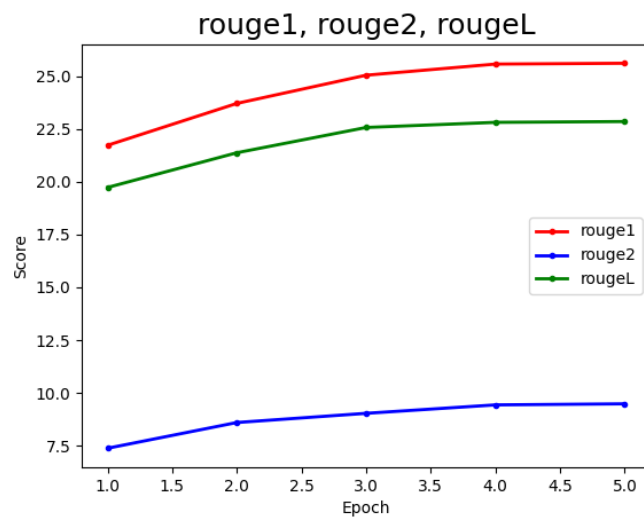


Figure 2: Learning curve

最後Eval的結果

- Rough-1: 26.4398
- Rough-2: 9.8887
- Rough-L: 23.6084

3 Generation Strategies

Strategies

- Greedy: 選擇機率最高的字。
- Beam Search: 對於n beams，會保留n條path，並選擇機率最高的path，相對於greedy考慮到全局最佳。
- Top-k Sampling: 對於機率最高前k個字進行sampling。
- Top-p Sampling: 改進top-k對於機率很集中於某個字的狀況，相加每個字的機率值至超過p%，並對這些字進行sampling。
- Temperature: 將預測出來的分布丟進softmax以更改diversity，使用 τ 來控制分布，當 τ 越靠近1，就會越接近softmax

$$P(w_t) = \frac{e^{s_w/\tau}}{\sum_{w \in V} e^{s_w/\tau}}, \text{ where } \tau \text{ is hyperparameter}$$

Hyperparameters

	greedy	Beam Search			Top-k	
		3	5	7	3	5
rouge-1	23.419	25.009	25.196	25.166	22.494	21.906
rouge-2	8.544	9.802	10.045	10.167	7.786	7.476
rouge-L	21.188	22.622	22.792	22.779	20.070	19.559
	0.85	Top-p		Temperature		
		0.90	0.95	0.85	0.90	0.95
rouge-1	20.318	19.934	19.282	20.033	19.594	19.196
rouge-2	6.763	6.627	6.187	6.625	6.285	6.161
rouge-L	18.152	17.776	17.174	17.812	17.528	17.137

Table 1: rouge score of generation strategy

由table1可得知beam search的表現明顯比其他策略好，而在Beam search中，隨著n愈大所需的記憶體空間與時間也愈大，當n設定為5後，後續的上升幅度就趨緩了，也因為所需的記憶體空間大小超過8G，所以最後使用5-beam為最後的策略。