ADL Hw3 report

R10723059 財金碩二 胡祖望

November 30, 2022

1 Q1: Model

1.1 Model

model config.


```
{
  "_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": "4.24.0",
  "use_cache": true,
  "vocab_size": 250112
}
```


使用 transformer encoder decoder 架構先將 maintext(source) 🄴入 tokenizer 中，結果當作 encoder 的 input 輸入，將最後一層的 encoder output 當作 hidden layer 的 query 和 key 接到 decoder 中將 title(target) text tokenizer output 右移當作 decoder 的 input，做 masked 的 Multi-Head Attention 後輸出 value 連接 encoder 的 output 再進入 feed forward 層，經過數層後再輸入到 Linear 層與 softmax 層生成機率。
src input ids = tokenizer(main text)
encoder output = encoder(src input ids)

decoder input ids[:, 1 :] = target input ids[:, : −1]
decoder output = decoder(decoder input ids, encoder output)
decoder output = decoder(postdecoder output, encoder output)

1.2 Preprocessing

tokenization: t5 tokenizer 是 sentencepiece，是由 byte pair encoding 以及 unigram language model 組成，會將 sentence 切成 subword，利用 viterbi algorithm 找出機率最大的 path 來做 tokenize。資料的處理是將資料🄵入 loaddataset 函式中，再把傳出的 dataset 用 map 函式送入 preprocess 函式中做 tokenize 最後🄵入 dataloader 中准備用於訓練。

2 Q2: Training

2.1 Hyperparameter

batch size = 4
batch size 太小會造成結果不好，且可能造成 gradient 計算的誤差，因此設定 gradient accumulation steps，但由於電腦規格必須設大一點
🄵🄵少 gpu 記憶體使用量，設定 gradient accumulation 🄵且不使用 AdamW 的 optimizer 而使用 Adafactor。
gradient accumulation steps = 10
per device train batch size = 4
gradient accumulation steps = 10
per device eval batch size = 4
eval accumulation steps = 4
learning rate = 1e-4
warmup ratio = 0.1

3 Q3:Generation Strategies

3.1 Stratgies

Stratgies:
1. greedy: 每個 time step 永遠都選 P(w|w1...wt−1) 最大的那一個。
2. Beam Search: 每個 time step 都選 num beams 個候選（最好的前 num beams 個），下一個 time step 則從這一個 time step 的候選中找出前 num beams 機率最大的候選。
3. Top k: Sampling: 從 P(w|w1...wt−1) 的機率分配從機率最大的前 K 名 sample 出 wt。
4. Top p: Sampling: 從 P(w|w1...wt−1) 的機率分配從機率最大的前幾名累加到 p 的 word sample 出 wt。
5. Temperature: 對 softmax 做 sharpen 或 smoothing，小於 1 是 sharpen，大於則是 smoothing。低 Temperatur = 更確定，高 Temperatur = 更隨機

3.2 Hyperparameters

1. greedy: greedy VS no greedy(”do sample”)
greedy 會比起 sample 來的好，原因是 sample 還是會有一定的機率 sample 到比較不好的選擇，造成後續 decode 的結果也不好。

	greedy	do_Sample
rouge-1_r	0.2168	0.1821
rouge-1_p	0.2948	0.2093
rouge-1_f	0.2394	0.1871
rouge-2_r	0.0817	0.0548
rouge-2_p	0.0606	0.105
rouge-2_f	0.088	0.0549
rouge-l_r	0.1943	0.1595
rouge-l_p	0.2649	0.1832
rouge-l_f	0.2145	0.1636

2. beam search: greedy VS beams = 5 VS beams = 10
有 beam search 會比起 greedy 好，原因是如果在前一輪 time step 選錯，會連帶影響後面的 time step，因此 beam search 可以有更多選擇，防止該輪 time step 不是機率最高是最後的答案。使用 beams=5 時會保留相對較少的最佳路徑，但從結果可以發現即使將 beams 調高到 10 分數也未必較高，原因是 beams 高時會一直選擇語意表達相對安全，不一定是錯誤的選擇，但實際上人類不會這樣□話，造成最後的語意還是與人類有差距。

	greedy	num_beam=5	num_beam=10
rouge-1_r	0.2168	0.2338	0.2355
rouge-1_p	0.2948	0.2986	0.2913
rouge-1_f	0.2394	0.251	0.2496
rouge-2_r	0.0817	0.0941	0.0957
rouge-2_p	0.0606	0.118	0.1166
rouge-2_f	0.088	0.0997	0.1002
rouge-l_r	0.1943	0.2095	0.2115
rouge-l_p	0.2649	0.2681	0.2618
rouge-l_f	0.2145	0.225	0.224

3. top k: greedy VS top k=8 VS top k=40
有使用 top k 時比較不會抽到太極端的奇怪選擇，會比原本的 sample 還要好，但如果 k 過大則會留下的選擇會過多，最後與原本的結果差不多。

	greedy	do_Sample	top_k=8	top_k=40
rouge-1_r	0.2168	0.1821	0.2048	0.1839
rouge-1_p	0.2948	0.2093	0.2527	0.2122
rouge-1_f	0.2394	0.1871	0.2174	0.1894
rouge-2_r	0.0817	0.0548	0.0679	0.0561
rouge-2_p	0.0606	0.105	0.0795	0.0618
rouge-2_f	0.088	0.0549	0.0701	0.056
rouge-l_r	0.1943	0.1595	0.1798	0.1603
rouge-l_p	0.2649	0.1832	0.2281	0.1854
rouge-l_f	0.2145	0.1636	0.1907	0.1651

4. top p: greedy VS top p=0.65 V.S. top p=0.9

使用 top p 時可以看到較能配合機率分配的厚尾、偏鋒態等形狀做選擇，用 top p 會比較好找到候選的範圍。整體分數比使用 topk 時好會會比原本的 sample 還要好，若 p 設定太大，仍會納入太多選擇，使結果變差。若 p 設定太小則會與 greedy 相當。

	greedy	do_Sample	top_p=0.65	top_p=0.9
rouge-1_r	0.2168	0.1821	0.2057	0.1948
rouge-1_p	0.2948	0.2093	0.2578	0.2283
rouge-1_f	0.2394	0.1871	0.2204	0.2008
rouge-2_r	0.0817	0.0548	0.0708	0.0614
rouge-2_p	0.0606	0.105	0.0845	0.0689
rouge-2_f	0.088	0.0549	0.0738	0.0623
rouge-l_r	0.1943	0.1595	0.182	0.1697
rouge-l_p	0.2649	0.1832	0.2281	0.2013
rouge-l_f	0.2145	0.1636	0.1948	0.1771

5. Temperature: greedy VS top p=0.65+Temperature=0.5 VS top p=0.65+Temperature=1.5

temperature 在 top p(保留較少的選項) 影響較大，原因是 temperature(使機率分配較不均🄵) 會影響每個 timestep 候選的數量，temperature<1，會將機率 sharpen，造成機率大的更大，小的更小，導致候選數量🄵少，更容易 sample 到機率大的，比較像 greedy；而 smooth 反之，導致機率平均分🄵，因此會像是從 top p 中 random sample。在此例中小的 top p 配合小的 Temperature 分數較佳。

	greedy	do_Sample	top_p=0.65+Temperature=0.5	top_p=0.65+Temperature=1.5
rouge-1_r	0.2168	0.1821	0.2196	0.1817
rouge-1_p	0.2948	0.2093	0.292	0.2093
rouge-1_f	0.2394	0.1871	0.2407	0.1862
rouge-2_r	0.0817	0.0548	0.0813	0.055
rouge-2_p	0.0606	0.105	0.103	0.0609
rouge-2_f	0.088	0.0549	0.0872	0.0548
rouge-l_r	0.1943	0.1595	0.1963	0.1589
rouge-l_p	0.2649	0.1832	0.2617	0.1834
rouge-l_f	0.2145	0.1636	0.2153	0.1907

最終選擇策略: num beam = 5