R10723050 財金碩二陳韻帆
Q1 Model

1.Model

使用 transformer encoder decoder 架構

先將 maintext(source) 丟入 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 層產出機率。

$$src_{input\_ids} = tokenizer(maintext)$$
$$encouder_{output} = encouder(src_{input\_ids})$$
$$decoder_{input\_ids}[:, 1:] = target_{input\_ids}[:, :-1]$$
$$decoder_{output} = decoder(decoder_{input\_ids}, encouder_{output})$$
$decoder_{output} = decoder(postdecoder_{output}, encouder_{output})$, while doing generate

2.Preprocessing

Tokenization 的部分，t5 的 tokenizer 使用的是 sentencepiece，是使用 byte pair encoding, unigram language model 去做 subword 的切割，再配合 viterbi algorithm 去找到最佳的 tokenize 方法。

資料的處理是將資料丟入 load_dataset 函式中，再把傳出的 dataset 用 map 函式送入 preprocess 函式中做 tokenize 最後丟入 data_loader 中準備用於訓練。

Q2 Training

1.Hyperparameter

batch size 設定為 4，嘗試過更小的 2 評分結果較差。

為減少 gpu 記憶體使用量，設定 gradient accumulation 並且不使用 AdamW 的 optimizer 而使用 Adafactor。

經過不同數值測試，最終將 gradient accumulation steps 設定為 4，並設定 learning rate 為 4e-4。

Q3 Generation Strategies

1.Stratgies

greedy: 每個 decode 的 timing 直接選擇當下條件機率最大的選項。

Beam Search: 若 num beam=n，每個 decode 的 timing 選擇當前機率最高的 n 條路徑保留，到最後選擇機率最高的路徑。

Top_k sampling: 每個 decode 的 timing 保留目前條件機率最高的 k 個條件機率做 sampling。

Top_p sampling: 每個 decode 的 timing 將條件機率由條件機率最高項依序做 sum，直到條件機率大於等於 p 停止，從有做 sum 的集合中做 sampling。

Temperature: 調整 softmax 時的計算方式，進而讓最終產出的機率分配不同。越高的 Temperature 值會讓最終的機率分配越接近 uniform，反之則會有較多較極端的機率值。

2.Hyperparameters

greedy V.S. do_sample

直接使用 greedy 策略結果比 do_Sample 好上許多，因為直接使用 do_Sample 比較有可能選到機率小，不合適的選擇，進而導致之後的選擇一起出問題。

	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	0.1050
rouge-2_f	0.0880	0.0549
rouge-l_r	0.1943	0.1595
rouge-l_p	0.2649	0.1832
rouge-l_f	0.2145	0.1636

num_beams=5 V.S. num_beams=10

使用 num_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 num_beams 調高到 10 分數也未必較高，是因為 num_beams 高時會一直選擇語意表達相對安全，不會錯誤的選擇，但實際上人類並不會這樣思考去表達語意，造成最後的語意還是與人類有差距。

	num_beam=5	num_beam=10
rouge-1_r	0.2338	0.2355
rouge-1_p	0.2986	0.2913
rouge-1_f	0.2510	0.2496
rouge-2_r	0.0941	0.0957
rouge-2_p	0.1180	0.1166
rouge-2_f	0.0997	0.1002
rouge-l_r	0.2095	0.2115
rouge-l_p	0.2681	0.2618
rouge-l_f	0.2250	0.2240

top_k=8 V.S. top_k=40

使用 top_k 時比較不會抽到太極端的奇怪選擇，可看出結果比單純用 do_sample 好，但 top_k 太大時，留下的選擇會過多，最後變成接近 do_sample 較不好的答案。

	top_k=8	top_k=40
rouge-1_r	0.2048	0.1839
rouge-1_p	0.2527	0.2122
rouge-1_f	0.2174	0.1894
rouge-2_r	0.0679	0.0561
rouge-2_p	0.0795	0.0618
rouge-2_f	0.0701	0.0560
rouge-l_r	0.1798	0.1603
rouge-l_p	0.2281	0.1854
rouge-l_f	0.1907	0.1651

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，整體分數比使用 top_k 時好，但若 p 設定太大，仍會納入太多選擇，使結果變差。

	top_p=0.65	top_p=0.9
rouge-1_r	0.2057	0.1920
rouge-1_p	0.2578	0.2283
rouge-1_f	0.2204	0.2008
rouge-2_r	0.0708	0.0614
rouge-2_p	0.0845	0.0689
rouge-2_f	0.0738	0.0623
rouge-l_r	0.1820	0.1697
rouge-l_p	0.2281	0.2013
rouge-l_f	0.1948	0.1771

top_p=0.65+Temperature=0.5 V.S. top_p=0.65+Temperature=1.5

使用較小的 top_p(保留較少的選項) 配合較小的 Temperature(使機率分配較不均勻) 會使最後容易取得機率較高的選項，可以在不同的範例中測試適合的組合方式，在此例中小的 top_p 配合小的 Temperature 分數較佳。

	top_p=0.65+Temperature=0.5	top_p=0.65+Temperature=1.5
rouge-1_r	0.2196	0.1817
rouge-1_p	0.2920	0.2093
rouge-1_f	0.2407	0.1862
rouge-2_r	0.0813	0.0550
rouge-2_p	0.1030	0.0609
rouge-2_f	0.0872	0.0548
rouge-l_r	0.1963	0.1589
rouge-l_p	0.2617	0.1834
rouge-l_f	0.2153	0.1907

最終選擇策略: num beam = 5