ADL HW2 Report　R12944062 李泓賢

Q1：

(A)Model

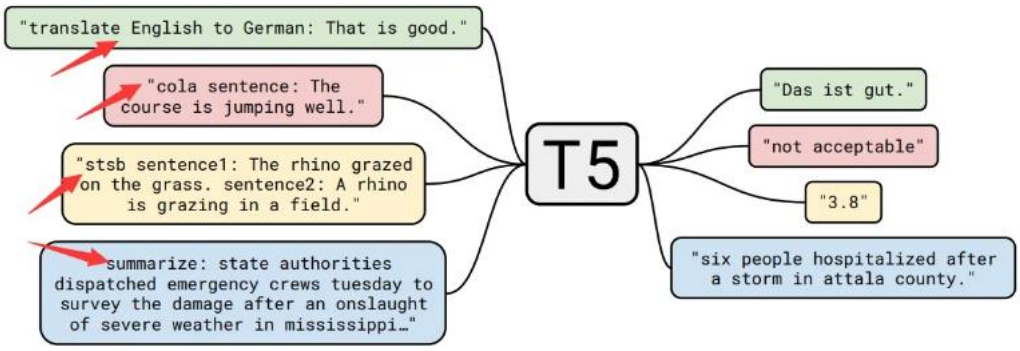


Figure 1: A diagram of our text-to-text framework. Every task we consider—including translation, question answering, and classification—is cast as feeding our model text as input and training it to generate some target text. This allows us to use the same model, loss function, hyperparameters, etc. across our diverse set of tasks. It also provides a standard testbed for the methods included in our empirical survey. "T5" refers to our model, which we dub the "**T**ext-**t**o-**T**ext **T**ransfer **T**ransformer".

T5 是一個 encoder-decoder 的 transformer，他可以執行很多種不同的任務，但在此之前必須加上不同的 prefix，例如本次的作業就需要在前面加上"summarize: "。在計算 loss 時，只需要輸入 input_ids 以及 labels，model 就會將 labels shift right(其實也就只是在前面加一個<pad>)作為 decoder 的輸入，之後將算出來的 logits 以及 groundtruth 做 cross-entropy，如此一來得到 loss。

基於上述 loss 的產生方式，代表每個 method 的 loss 產生方式都相同，因此我們不需要針對每個 method(ie. Beam = 2, Beam = 3)都做四個小時的訓練，反而只需要訓練出一個 model 然後對於每個 method 瘋狂 inference 即可(利用.generate())。

(B)Preprocessing

mt5 使用的 tokenizer 是 sentencepiece，這是一個 google 開源的工具，他最大的特色是不需要預先分詞，可以直接在原始的句子上訓練；此外他也實現了兩種 subword 算法(bpe 與 unigram language model)。

Q2：

(A)Hyperparameter

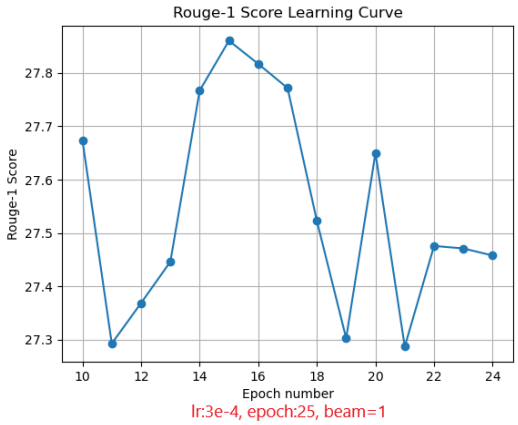
其實我有嘗試過蠻多種不同的組合，以下一一說明：首先我 learning rate 是設 3e-5(跟之前 hw1 一樣的數字)，後來看了 hugging face t5 model 的 doc 看到這一段：

Additional training tips:

- T5 models need a slightly higher learning rate than the default one set in the `Trainer` when using the AdamW optimizer. Typically, 1e-4 and 3e-4 work well for most problems (classification, summarization, translation, question answering, question generation). Note that T5 was pre-trained using the AdaFactor optimizer.

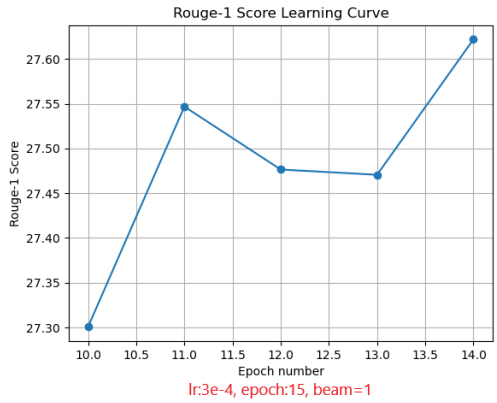
因此我嘗試 3e-4，發現 performance 提升了非常多(rouge-1 大概提升了 1.多)，於是我最後決定使用 3e-4。

至於 epoch 的選擇，由於 TA 只有建議我們訓練 4 個小時就可以通過 baseline，並沒有建議多少個 epoch 才能通過，因此我一開始設定 25 個 epoch(扣除 inference 時間大約五個小時 on my gtx3060)，learning curve 如下：



lr:3e-4, epoch:25, beam=1

我發現線條非常奇怪，可能有 overfitting 的問題，於是我猜是 epoch 太多了，因此我又嘗試看看降低 epoch 數至 15(扣除 inference 時間大約四個小時)：



lr:3e-4, epoch:15, beam=1

我認為 15 個 epoch 應該是一個不錯的選擇，因此最終採用 epoch=15。

至於 train batch size，我任何的實驗組合都採用 8 這個數字。

基於以上的原因，我最終的參數以及成效如下：

Model：google/mt5-small

Performance：(Public.jsonl with beam=3)

{"eval_rouge-1": 28.6013, "eval_rouge-2": 11.4134, "eval_rouge-l": 25.3686}

Loss Function：CrossEntropyLoss()

Optimization：AdamW()

Learning Rate：3e-4

Trainning Batch Size：8

Epoch：15

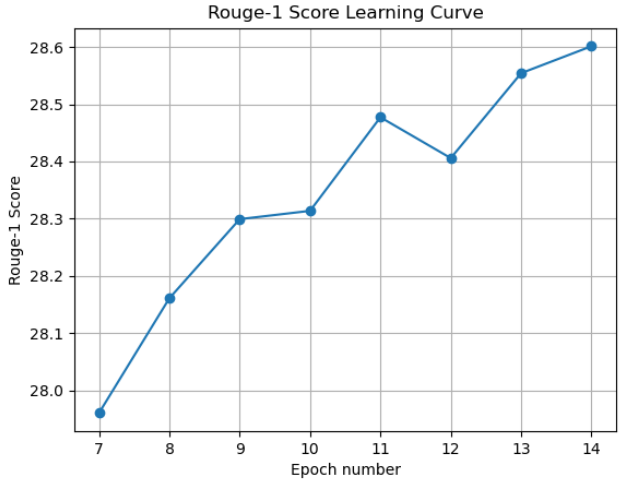
Max_source_length：256

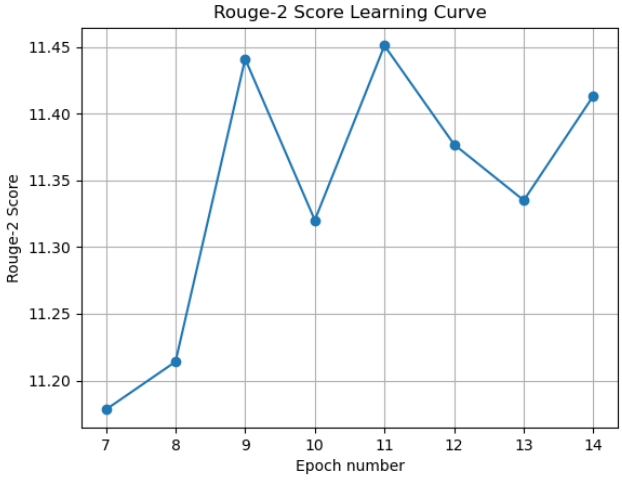
Max_target_length：64

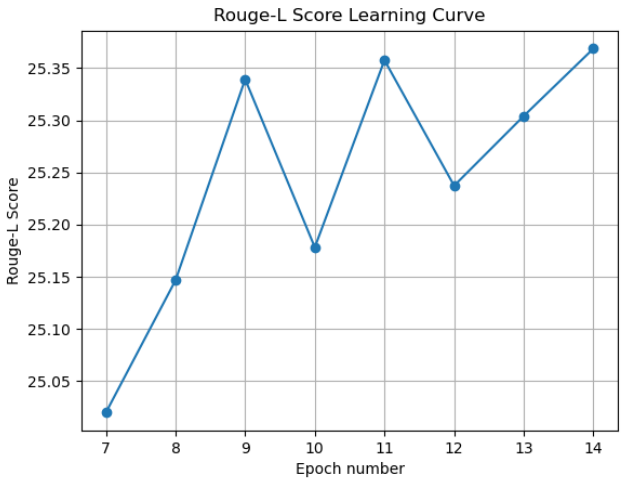
(B)Learning Curves

先說明一下我遇到的問題，我發現我如果從第 6 個 epoch 開始 inference 仍然會跳出”hypothesis is empty”的 error，所以我不確定是不是 6 個 epoch 還不足以讓模型 predict 出非空的 sentence；又或者 6 個 epoch 其實夠只是我運氣很差。我先暫時不考慮第二個可能性，因此後續所有實驗組合我都是等模型訓練 7 個 epoch 之後(也就是第 8 個 epoch)才開始做 inference，然後就沒有遇到這個問題了。

三個 rouge-score 的圖請見下一頁：


Rouge-1 Score Learning Curve
Rouge-1 Score
28.6
28.5
28.4
28.3
28.2
28.1
28.0
7
8
9
10
11
12
13
14
Epoch number

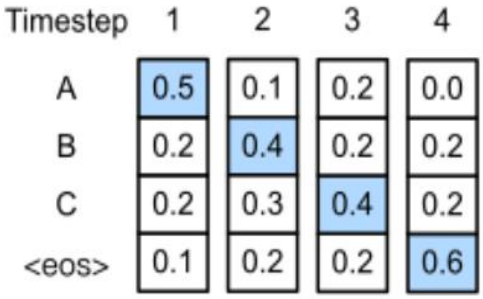

Rouge-2 Score Learning Curve
Rouge-2 Score
11.45
11.40
11.35
11.30
11.25
11.20
7
8
9
10
11
12
13
14
Epoch number


Rouge-L Score Learning Curve
Rouge-L Score
25.35
25.30
25.25
25.20
25.15
25.10
25.05
7
8
9
10
11
12
13
14
Epoch number

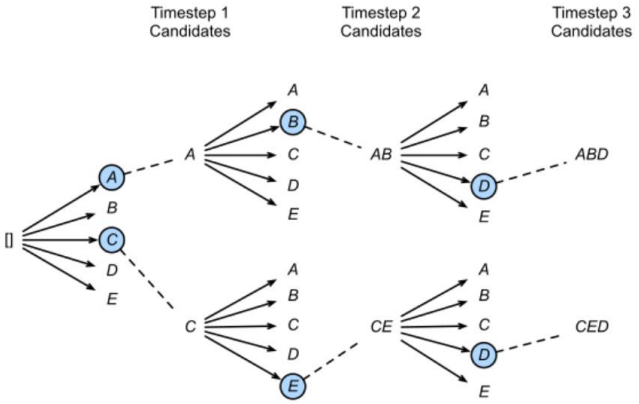
Q3：

(A)Strategies

- Greedy：每一個時間點 t，都根據 1:t-1 選出機率最高的詞，能有一定程度的表現，但不一定是最佳解。如下圖會得出 ABC。



- Beam Search：是一種對 greedy 的改進，也可以說 greedy 是 beam search 的一種特例(beam=1)，就是多一點選擇範圍，beam=n 即共有 n 個選擇。下圖為 beam=2，即每個時間點 t 都會保留從過去到現在(1:t-1)機率最高的 2 個選擇：



- Top-k Sampling：即在 sampling 之前減小採樣空間，將採樣空間限縮成前 k 個機率最高的 token，然後重新計算得到新的機率分布。若 k 太大容易採樣出長尾詞，若 K 太小容易退化成 Beam Search。
- Top-p Sampling：也是在 sampling 之前減小採樣空間，model 會選擇最少的 token 數且加總機率>=p 的那些 token，然後重新計算得到新的機率分布。
- Temperature：可以將 temperature t 想像成一個 scaler，也就是 softmax 中的分子與分母都除以一個 t，之後才將此值送入 softmax function 中，公式如下圖：

$$P(x_i|x_{1:i-1}) = \frac{\exp(u_i/t)}{\sum_j \exp(u_j/t)}$$

(B)Hyperparameters

我嘗試了不只下表的 decoder method，下表僅列舉了一些比較具有比較意義且效果較接近 optimal 的方式：

method	rouge-1	rouge-2	rouge-L
greedy	27.347	10.2774	24.2683
beam=2	28.2559	11.0308	25.0746
beam=3	28.6013	11.4134	25.3686
beam=4	28.5918	11.5466	25.3869
top_p=0.05	27.2638	10.2364	24.1617
top_p=0.15	26.9758	9.9942	23.882
top_p=0.25	26.6953	9.827	23.5735
top_k=1	27.347	10.2774	24.2683
top_k=2	26.5673	9.5746	23.4238
top_k=3	26.19	9.3496	23.0519
temp=0.05	27.3031	10.2068	24.1748
temp=0.15	27.2169	10.1635	24.1291
temp=0.25	27.0509	10.0453	23.9633
k=40, p=0.05	27.347	10.2774	24.2683

從表中可以看出，beam=3 會有最好的 rouge-1 表現，因此我選擇這個參數作為我的 decoder method。

此外，top_k=1 與 greedy 具有相同的效果，這也十分合理。我也觀察到 top_p 與 top_k 的 score 會隨著 p 或 k 的數值增加而減小，這是因為這兩個方法採取了 sampling，因此每個機率低的 token 都有機會被採樣到，所以若是機率低的 token 被採樣到，那麼模型的 score 肯定會下降。

另外，我也試著結合 top_k=40 與 top_p=0.05，發現這得出的效果比單單只使用 top_p=0.05 還要高一些(rouge-1, 27.347>27.2638)，因此我得到的結論是：結合 top_p 與 top_k 兩者可以得到更好的效果。