ADL HW3 Report

B08902029 資工三 陳咏誼

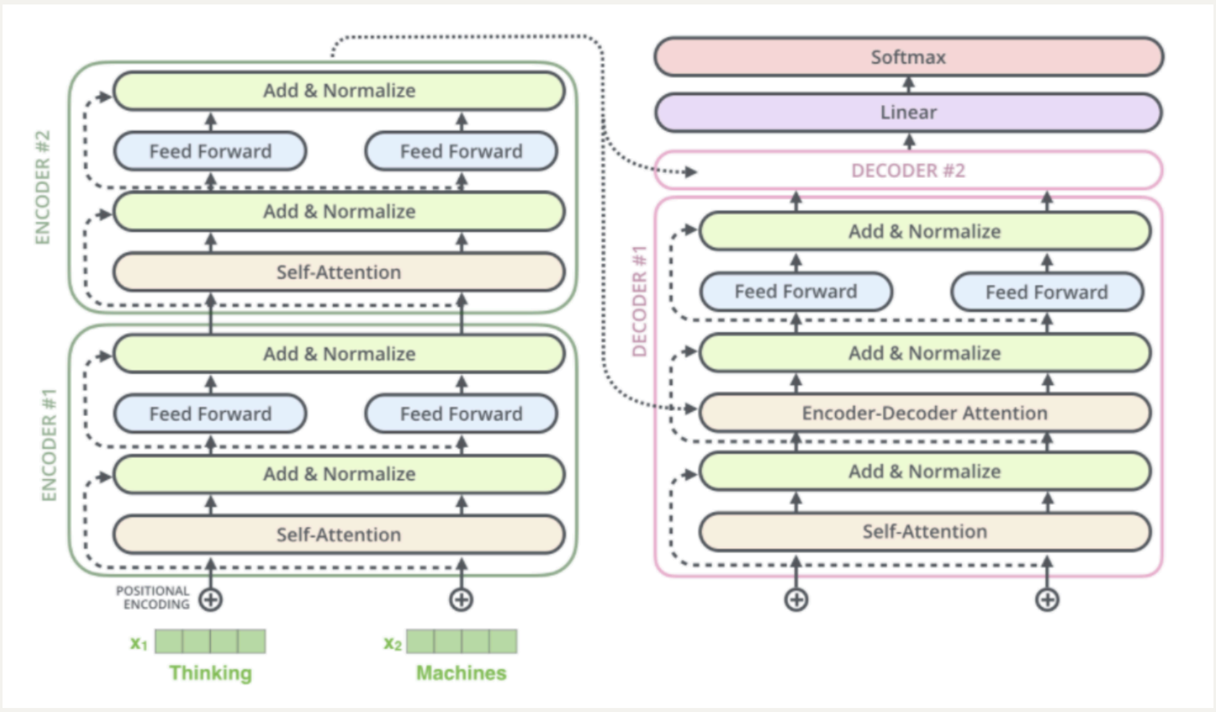
Q1: Model

Model (1%): Describe the model architecture and how it works on text summarization.

I use google/mt5-small as the pretrained model. The architecture of mt5 is almost the same as t5 except for the following difference:

- Use Gated-GELU activation instead of ReLU
- No dropout for unlabeled data
- No parameters sharing between embedding layers and classification layers
- Larger `d_model`, smaller `num_heads` and `d_ff`

The picture is describing t5 architecture, and it's basically also mt5 architecture. mt5 is an Encoder-Decoder model, which is pretrained on C4 dataset containing multiple NLP task, such as translation, summarization, semantic textual similarity. The special point of mt5(t5) is that it treats every ask as seq-to-seq task.



When finetuning on text-summarization task, it takes the encoding of maintext as input_ids and takes the encoding of title as labels.

Preprocessing (1%): Describe your preprocessing (e.g. tokenization, data cleaning and etc.)

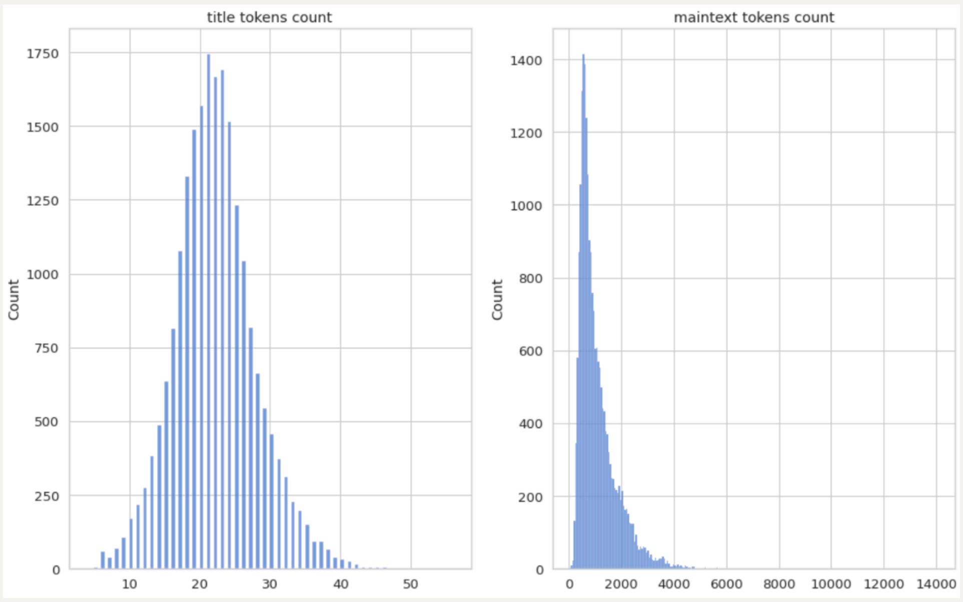
The model takes three parameters when being fine-tuned: `input_ids`, `attention_mask`, and `labels`

- `input_ids`, `attention_mask`:
    - First clean up the unnecessary characters like `\t` and `\n` of MAINTEXT, and use `google/mt5-small` tokenizer to obtain the embeddings.
- `labels`:
    - First clean up the unnecessary characters like `\t` and `\n` of TITLE, and use `google/mt5-small` tokenizer to obtain the embeddings. Then, transfer the `pad_id` within the input_ids of title embedding to `-100` so that the loss of padding would not be calculated.

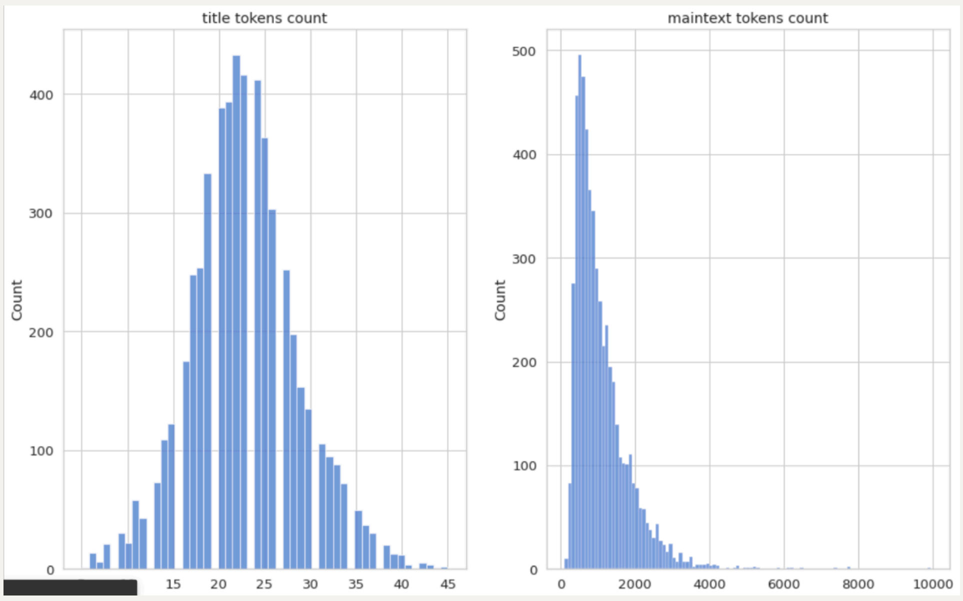
Q2: Training (2%)

Hyperparameter (1%): Describe your hyperparameter you use and how you decide it.

- batch_size: 1 * 8 (actual batch_size * accu_grad)
    - Test with batch_size = {2, 4, 8, 16}, and 8 has the best score.
- lr: 5e-5
    - Test with lr = {5e-4, 5e-5}, and 5e-5 has the best score.
- num_epoch: 25
    - Observe when the score stops growing. The best score shows at the 20-th epoch.
- warmup_ratio: 0.1
- title_max_len = 64, maintext_max_len = 2048
    - Analyze the distribution of training data and testing data.
        - TRAIN:

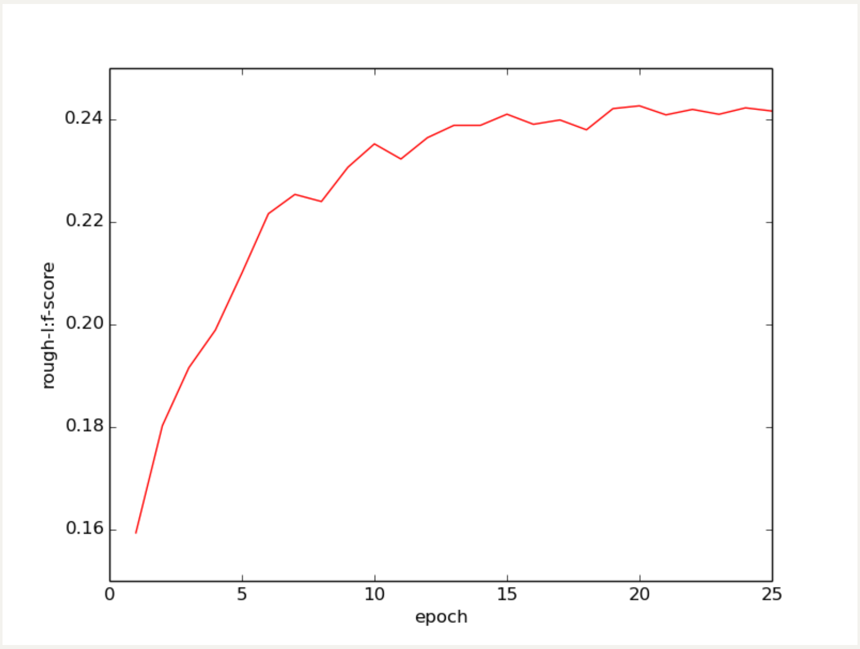


        - TEST:

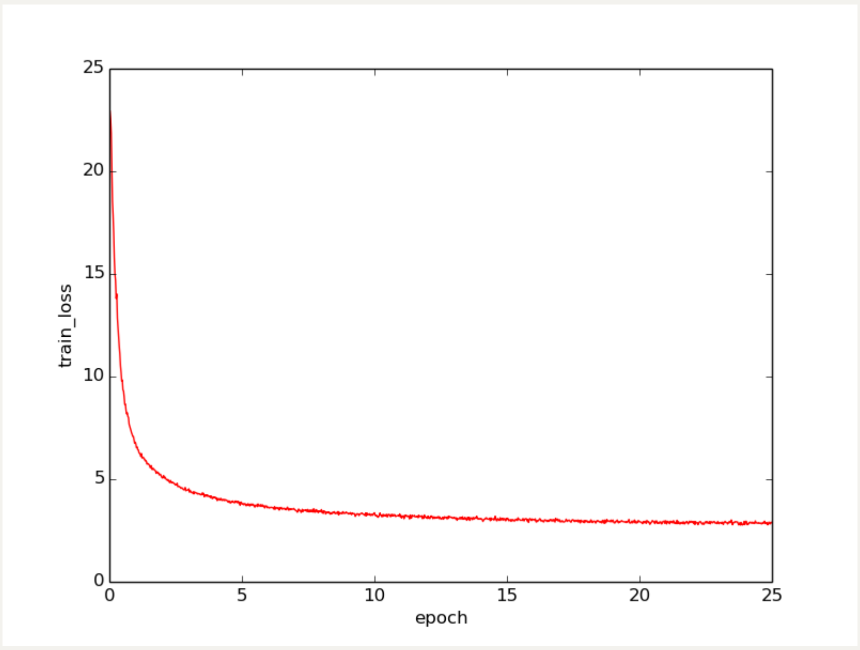


Learning Curves (1%): Plot the learning curves (ROUGE versus training steps)

Rough-L: f-score:



Train Loss:



Q3: Generation Strategies(6%)

Stratgies (2%)

- Greedy Search
    - At each generating step, simply select the word with the highest probability as its next word:
- Beam Search
    - Keep tracking the most likely `num_beams` of hypotheses at each time step, and eventually choose the hypothesis that has the overall highest probability, so as to reduces the risk of missing hidden high probability word sequences.
- Top-k Sampling
    - Filter the K most likely next words, and redistribute the probability mass of only those K next words.
- Top-p Sampling
    - At each generating step, choose a word from the smallest possible set of words whose cumulative probability exceeds the probability P, so less P avoids choosing the words that are too "normal" and "meaningless".
- Temperature

- Change the equation of softmax from $\frac{e^{w}}{\sum_{w' \in V} e^{w'}}$ to $\frac{e^{w/\tau}}{\sum_{w' \in V} e^{w'/\tau}}$. The higher τ is, the smoother the result of softmax is. The default value of τ is 1, that is, the original softmax.

Hyperparameters (4%)

Greedy and Beam Search:

It shows that `num_beams` = 5 and `num_beams` = 10 tie. In view of running time, I think `num_beams`=5 would be better. And we can learn that the words with top 6-10 possibility are tended not to be in the optimal path.

num_beams	rouge-1			rouge-2			rouge-l		
	r	p	f	r	p	f	r	p	f
1	0.2508	0.2769	0.2557	0.0904	0.1022	0.0928	0.2161	0.2398	0.2207
2	0.2667	0.2934	0.2718	0.1075	0.1183	0.1092	0.2355	0.2592	0.2398
5	0.2735	**0.2947**	**0.2758**	0.1113	**0.1202**	0.1120	0.2415	**0.2602**	**0.2434**
10	**0.2742**	0.2911	0.2746	**0.1126**	0.1200	**0.1127**	**0.2424**	0.2573	0.2426

Top-k Sampling

K = 5 has better performance than K = 20. I think the reason is that K = 20 allows the words which are not good enough to be sampled.

K	rouge-1			rouge-2			rouge-l		
	r	p	f	r	p	f	r	p	f
5	0.2350	0.2492	0.2349	0.0774	0.0837	0.0776	0.2020	0.2148	0.2020
20	0.2178	0.2261	0.2155	0.0678	0.0716	0.0674	0.1865	0.1942	0.1847

Top-p Sampling

P = 0.5 has better performance than P = 0.9 since P = 0.5 avoids choosing the words with general meaning. And from this comparison, we can judge that some words sampled by K = 20 in last experiment have lower cumulative probabilities than 0.5 .

P	rouge-1			rouge-2			rouge-l		
	r	p	f	r	p	f	r	p	f
0.5	0.2253	0.2396	0.2258	0.0770	0.0837	0.0776	0.1956	0.2089	0.1963
0.9	0.1829	0.1868	0.1796	0.0578	0.0602	0.0571	0.1610	0.1646	0.1580

Temperature

I set K = 5 to do the experiment since it has best performance among the 4 configurations of sampling above.
The result shows that the lower temperature leads to better performance. This makes sense since lower temperature makes the distribution less smooth. The word with high possibility are tended to be sampled.

temperature	rouge-1			rouge-2			rouge-l		
	r	p	f	r	p	f	r	p	f
0.7	0.2426	0.2625	0.2449	0.0823	0.0911	0.0837	0.2083	0.2261	0.2105
1	0.2350	0.2492	0.2349	0.0774	0.0837	0.0776	0.2020	0.2148	0.2020
1.5	0.2207	0.2293	0.2179	0.0672	0.0712	0.0666	0.1876	0.1954	0.1853

Final generation strategy

In view of the time limit, I finally choose to use Beam-decoding with num_beams=5.

Bonus: Applied RL on Summarization (2%)

Algorithm (1%): Describe your RL algorithms, reward function, and hyperparameters.

My RL algorithm is policy-gradient with taking the f scoreof rouge-l as reward, baseline = 0.2 and
$\gamma = 0.999$ on checkpoint of MT5 that has been trained with the hyperparameters in Q2.

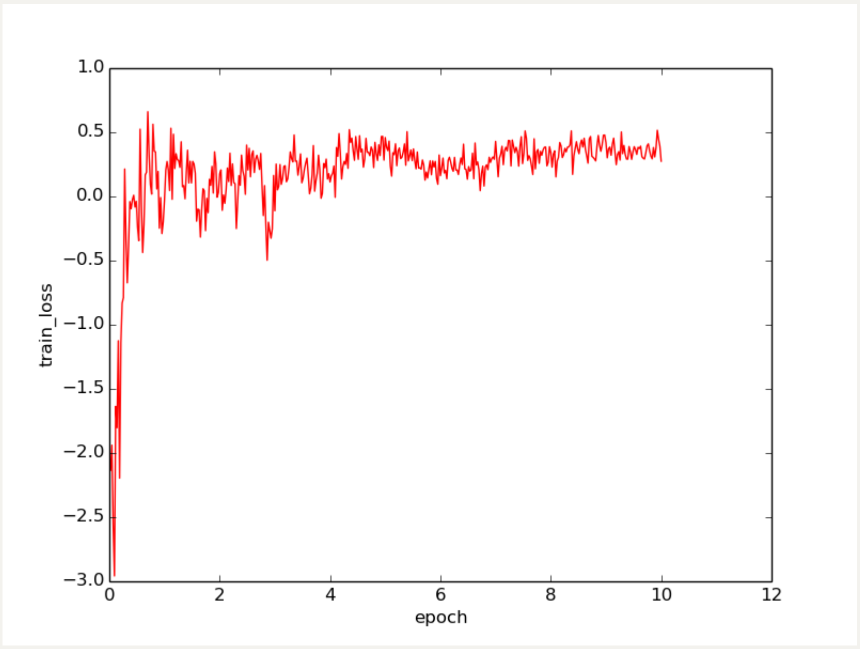
The generated word can be regarded as action. In each epoch, I sample words from the logits outputed from model and compute the f score of rouge-l as the total reward r_n. The reward of each word is calculated by $R_t = (\gamma^{n-t} * r_n) - 0.2$.

The train loss of each single data is $\Sigma_{1 \le t \le n} R_t \times \log$ (prob of w_t).

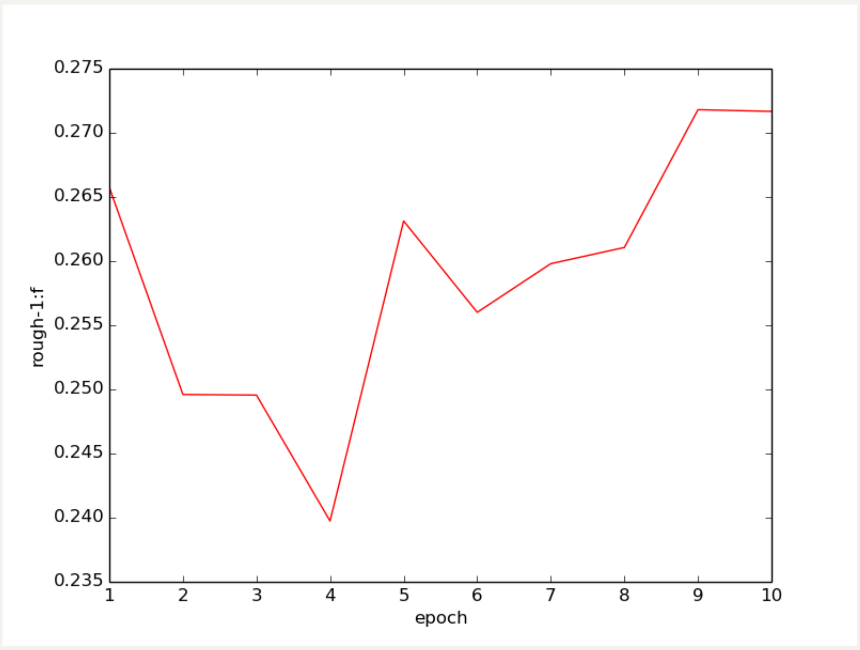
Compare to Supervised Learning (1%): Observe the loss, ROUGE score and output texts, what differences can you find?

I found that the performance reaches the worst at epoch=4 and the score starts to grow, so the fina performance might be better if I train it with more epochs.

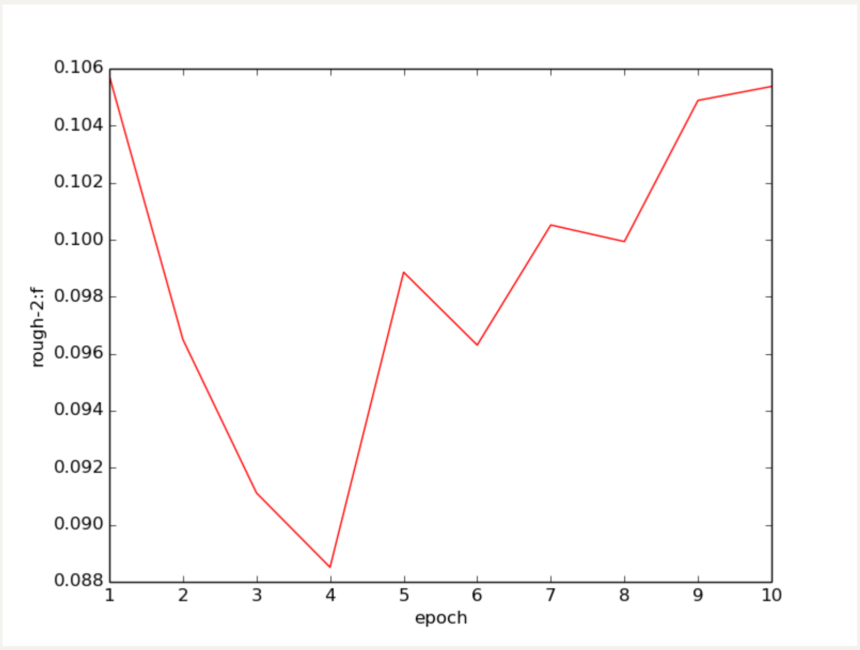
- Train Loss:



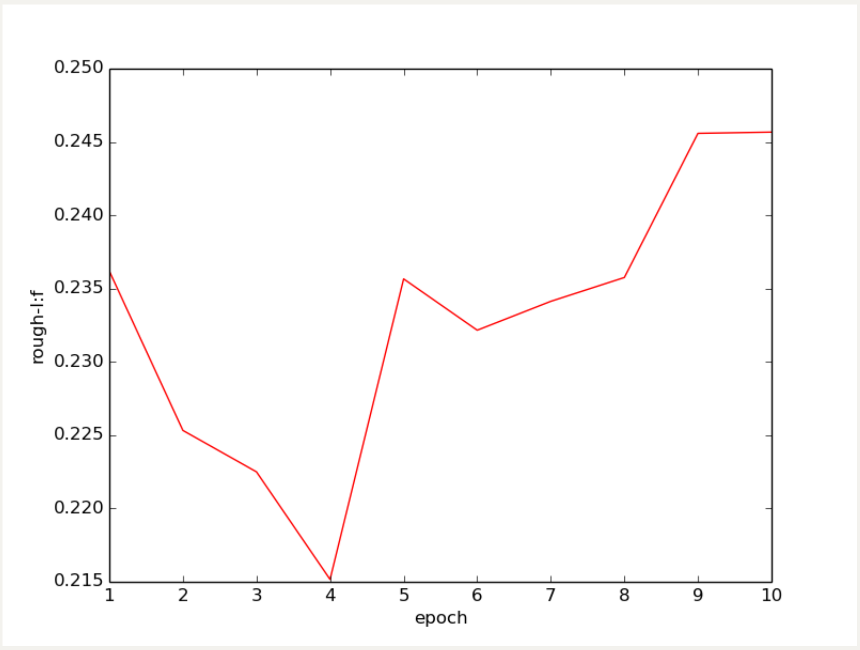
- Rouge-1 f score:



- Rouge-2 f score:



- Rouge-l f score:



Method	rouge-1			rouge-2			rouge-l		
	r	p	f	r	p	f	r	p	f
Supervised	**0.2742**	0.2911	**0.2746**	**0.1126**	**0.1200**	**0.1127**	**0.2424**	0.2573	0.2426
Supervised + RL	0.2570	**0.3170**	0.2716	0.1047	0.1157	0.1053	0.2323	**0.2872**	**0.2456**

I cannot tell obvious difference from the table above. I use `train_rl.sh` to train RL method with lr=5e-5, num_epochs=10, batch_size=8. Maybe other configuration would improve the result.