ADL HW 3

B08902134, 曾揚哲

Q1 Model Architecture

The model I use is the adaptation version of T5 [1] to multi-lingual tasks, mT5 [2], which is basically follows standard encoder-decoder architecture [3] with multiple stack of attention blocks. The pretrained model I used, `google/mT5` is loaded from huggingface then fine tuned on the current task. The encode-decoder architecture enables model to be easily applied on summarization or generally speaking, all generative tasks. During training, the padded mini-batch text is fed into **t5**-encoder with expected title fed into the **t5**-decoder. In inference stage, the **t5**-encoder takes the text and the **t5**-decoder would generate the predicted logits.

Q2 Preprocessing

The data preprocessing procuder follows the original recipe of T5[1]. SentencePiece [4] tokenizer is used. Each context is truncated or padded to 256 what each title is to 64. No data cleaning techniques are applied.

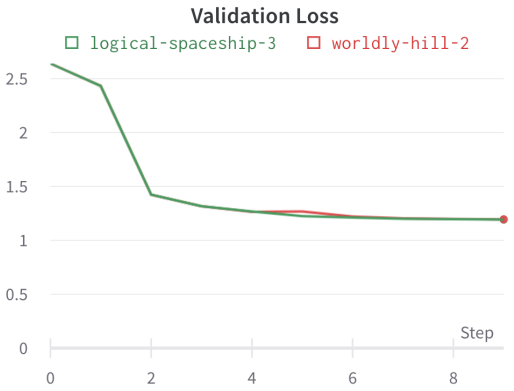
Q3 Hyperparameter

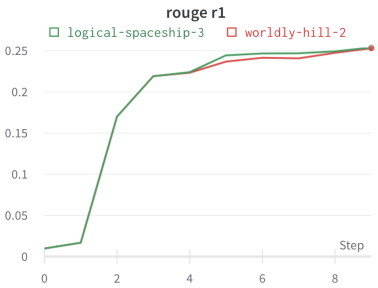
The hyperparamters is adapted from the original paper [2].

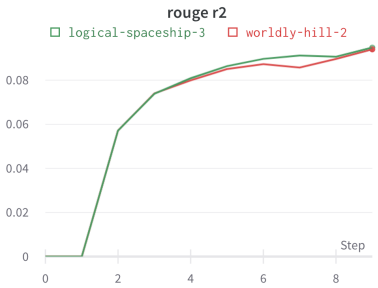
1. Learning rate: 0.001
2. Weight decay: 0.01
3. Epoch: 20
4. Batch size: 8

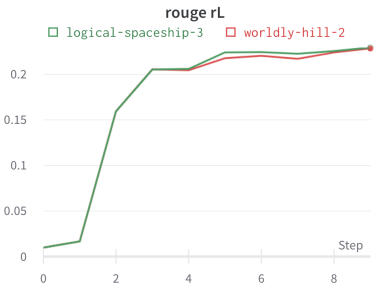
Q4 Plot

The green line is trained with the linear scheduler while the red line is trained with the cosine scheduler.









Q5 Strategies

Q5.1 Greedy

Greedy decoding is basically choosing the most probable word for each preceding word, whose output can be poor due to lack of backtracking and that the unexplored path may have higher probability.
The control variables for this experiment is as follow:

```
{'num_beams': 1,' 'top_k': 200, 'top_p': 1.0, 'temperature': 1.0}
```

do_sample	Rouge 1	Rouge 2	Rouge L	Time	GPU (MB)
False	0.2335	0.0822	0.2092	0:01:58	3011
True	0.2040	0.0663	0.1807	0:02:09	3075

In sight of this experiment, I found out that sampling from a distribution is worse and hence select `do_sample` = False.

Q5.1.1 Beam Search

Beam search is similar to greedy encoding while it keep track of k most probable sequences to find a better and hidden solution. Greedy encoding is basically beam search with $k = 1$. While small beam size may lead to ungrammatical, unnatural or incorrect sentences, adopting large beam size is computational expensive and exhaustive.
The control variables for this experiment is as follow:

```
{'do_sample': False, 'top_k': 200, 'top_p': 1.0, 'temperature': 1.0}
```

num_beams	Rouge 1	Rouge 2	Rouge L	Time	GPU (MB)
1	0.2335	0.0822	0.2092	0:01:57	3011
5	0.2489	0.0936	0.2233	0:05:42	6109
10	0.2506	0.0946	0.2241	0:07:09	9503

With the increased beam size from 1 to 5, the rouge score for each value increases much while the it does not when beam sized increased from 5 to 10. As beam size increase, the time and space for inference multiplies. For the following part, I will use `num_beams` = 5.

Q5.1.2 Top-K Sampling

Sample the word via distribution but restricted to the top-k probable words. Some special cases including $k = 1$ as greedy decoding and $k = V$ as pure sampling. As k increases, the output become more diverse but also risky, while as k decreases, the output is more generic.
The control variables for this experiment is as follow:

```
{'do_sample': True, 'num_beams': 1, 'top_p': 1.0, 'temperature': 1.0}
```

It appears that with higher k, the rouge score is lower. After all I'm not using sampling as final decoding method.

k	Rouge 1	Rouge 2	Rouge L	Time	GPU (MB)
1	0.2334	0.0822	0.2092	0:02:07	3075
50	0.2054	0.0677	0.1824	0:02:38	3075
100	0.2017	0.0663	0.1795	0:02:10	3075
200	0.2011	0.0655	0.1790	0:02:36	3075

Q5.1.3 Top-P Sampling

Sampling from a subset of vocabulary with the most probability. That is,

$$w_i \sim V^{(p)}$$

$$\text{where } V^{(p)} = \sup_{V' \subset V} \sum_{x \in V'} P(x|w_1, \dots, w_{i-1}) \geq p$$

The control variables are as follow:

```
{'do_sample': True, 'num_beams': 1, 'top_k': 0, 'temperature': 1.0}
```

p	Rouge 1	Rouge 2	Rouge L	Time	GPU (MB)
1.0	0.1996	0.0656	0.1782	0:03:17	3043
0.5	0.2267	0.0789	0.2023	0:07:02	3485
0.25	0.2329	0.0812	0.2079	0:07:09	3485
0.0	0.2334	0.0822	0.2092	0:07:11	3485

It appears that the lower p is, the better the performance is. Not sure why the inference time becomes twice longer when $p < 1.0$.

Q5.1.4 Temperature

The temperature hyperparamter applied to the softmax function is to control the diversity which is applicable to any decoding algorithm. The diversity of output increases τ increases, and vice versa.
The control variables are as follow:

```
{'do_sample': True, 'num_beams': 1, 'top_k': 0, 'top_p': 0.25}
```

temperature	Rouge 1	Rouge 2	Rouge L	Time	GPU (MB)
1.0	0.2329	0.0812	0.2079	0:05:11	3485
0.5	0.2329	0.0821	0.2090	0:05:17	3517
0.25	0.2334	0.0822	0.2091	0:05:08	3517
0.1	0.2334	0.0822	0.2092	0:05:05	3517

We could see that the lower temperature leads to higher Rouge-L score overall score.

Conclusion

I adopt beam search with `num_beams=5` as the final decoding method.

Q6 Bonus: Applied RL

Algorithm

I used policy gradient as my RL algorithm and take the following formula as reward function.

$$\text{rouge-1} \times 0.3 + \text{rouge-2} \times 0.5 + \text{rouge-L} \times 0.2$$

The hyperparamter used is same as trained with pure-DL and the RL training process is conducted after DL training is done, i.e., in an attempt to fine tune the model.
Let θ_{DL} be the trained network parameters by previous DL process.
For each training iteration, I sample the sentence generated and its corresponding reward. For each reward, calculate the exponential cumulative sum with $\gamma = 0.99$, i.e.,

$$R = \sum_{n=1}^{N} \gamma^{N-n} r_n$$

Then compute the loss with it and use `AdamW` optimizer to update the parameters of the model.

Compare to Supervised Learning

Training with RL, the loss could fluctuate around 0, either positive or negative, which is not the case in deep learning, or to be specific, maximum likelihood estimation. The text generated seems weird but the rouge score is slightly higher. The improvement is not obvious perhaps the reward function I designed didn't work properly in this task, and mentioned in the lecture, we probably should directly take `rouge-L` as the reward function. The reason I used weighted sum in my experiment is attempting to improve `rouge-2` more aggressively than two other terms, however, in vain. Most importantly, the loss calculation in `PyTorch` seems not supporting mixed precision training, and I have no choice but to fall back to original fp32 training, leading a prolonged training time appoximately twice as deep learning procedure.

w/ RL	Rouge 1	Rouge 2	Rouge L
False	0.2489	0.0936	**0.2233**
True	**0.2490**	**0.0951**	0.2221

References

[1] Raffel, Colin, et al. "Exploring the limits of transfer learning with a unified text-to-text transformer." arXiv preprint arXiv:1910.10683 (2019).

[2] Xue, Linting, et al. "mT5: A massively multilingual pre-trained text-to-text transformer." arXiv preprint arXiv:2010.11934 (2020).

[3] Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems 30 (2017).

[4] Kudo, Taku, and John Richardson. "Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing." arXiv preprint arXiv:1808.06226 (2018).