Q1: Model (2%)

Model (1%)

> Describe the model architecture and how it works on text summarization.

`google/mt5-small` consists of an encode and a decoder. The encoder takes the tokenized sentence as input, passes it through transformer model (bi-directional encoder) and outputs the vector representation of the sentence (hidden state). The decoder takes the hidden state and a sentence as input, then it predicts the next word (token) following the given sentence with transformer model. By setting the initial sentence to be a `<BOS>` token and iteratively predicting the next token, the decoder can eventually generate a meaningful sentence, especially with the help of some generation algorithms.

On text summarization task, encoder turns the sentence into a high dimensional vector representing its meaning. Then, decoder can thus generate a summary according to the vector passed from encoder.

Preprocessing (1%)

> Describe your preprocessing (e.g. tokenization, data cleaning and etc.)

The data preprocessing is relatively simple. We only have to tokenize the main text and the title, then clipped them to under 1024 and 64 tokens respectively.

Although only 60% of the main texts are within 1024 tokens, we can't further extend the input size because of the 8GB GPU limit.

Q2: Training (2%)

Hyperparameter (1%)

> Describe your hyperparameter you use and how you decide it.

- ***maximum title length: 64***
  All of the titles from training data are within 64 tokens.
- ***maximum paragraph length: 1024***
  Limited by 8GB GPU memory.
- ***batch size: 1***
  Since `google/mt5-small` is still really large, larger batch size can't fit within 8GB

memory limit.

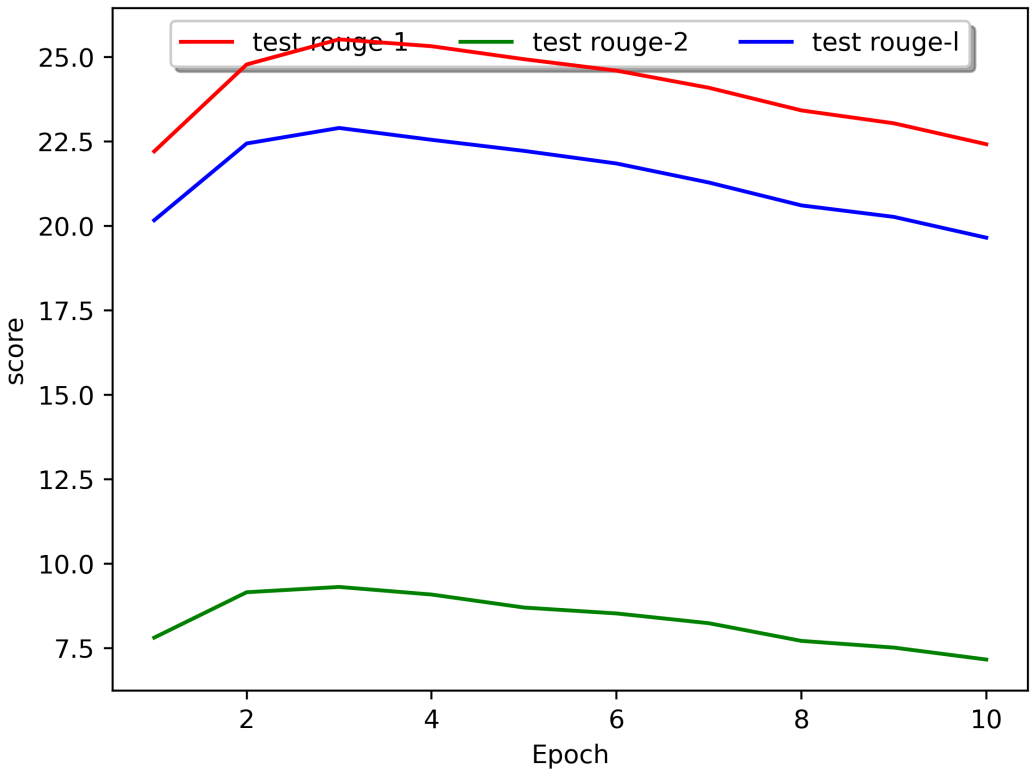
- ***gradient accumulation steps: 2***
  Add gradient accumulation to adjust the randomness.
- ***optimizer: Adafactor***

```python
Adafactor(
  params=model.parameters(),
  lr = 0.0001,
  scale_parameter=False,
  relative_step=False,
  clip_threshold=1.0,
)
```

As suggested by huggingface website, adafactor is specially designed for MT5 and is used to pretrain MT5 as well. Combined with the advantage of lower GPU consumption, I choose Adafactor as the optimizer.

Learning Curves (1%)

> Plot the learning curves (ROUGE versus training steps)



Q3: Generation Strategies(6%)

Stratgies (2%)

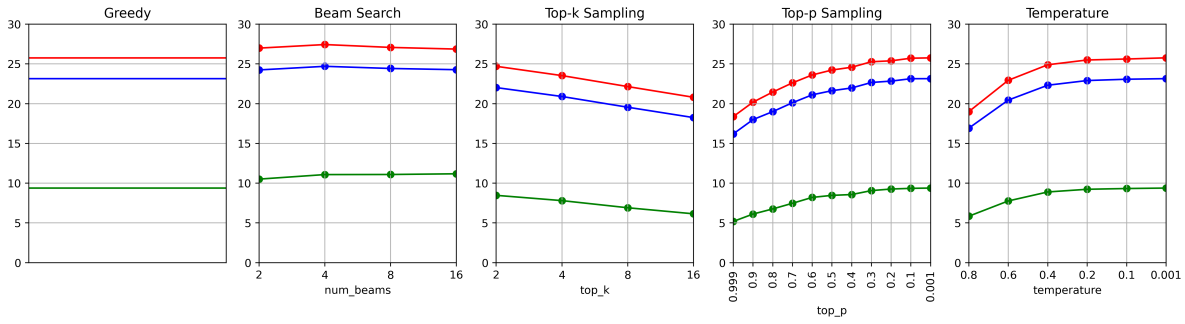
> Describe the detail of the following generation strategies:

- ***Greedy***
  Greedily choose the next token with highest probability.
- ***Beam Search***
  Keep the k most probable sequences, and iteratively find the k most probable successors extended from current k sequences.
- ***Top-k Sampling***
  Choose the next token by sampling from the k hightest probable tokens according to the probability distribution predicted by the model.
- ***Top-p Sampling***
  Similar to Top-p. However, instead of limiting the set of sampled tokens, Top-p limit the total probability of the sampled tokens.
- ***Temperature***
  Controll the sampling diversity, i.e., smoothening the probability distribution, by dividing the model output by a constant before softmax.

Hyperparameters (4%)

> Try at least 2 settings of each strategies and compare the result.
>
> What is your final generation strategy? (you can combine any of them)

Red = rouge-1 Green = rouge-2 Blue = rouge-l



I choose Beam Search with 16 beams as my final generation strategy.

Bonus: Applied RL on Summarization (2%)

Algorithm (1%)

> Describe your RL algorithms, reward function, and hyperparameters.

I use the Python library `TextRL` to build up the RL training process, which utilizes Actor-Critic algorithm.

The reward function first calculates tw_rouge score for the sentence generated so far. Then, the reward is defined as minimum difference between baseline and calculated score among all three rouge scores.

hyperparameters:

```python
update_interval       = 10
minibatch_size        = 10
epochs                = 1000
steps                 = 1000
train_max_episode_len = 50
lr                    = 1e-4
```

Compare to Supervised Learning (1%)

> Observe the loss, ROUGE score and output texts, what differences can you find?

	rouge-1	**rouge-2**	**rouge-l**
w/o RL	25.76	9.38	23.14
w/ RL	25.07	9.00	22.51

The score did not really improved, but after observing the generated text, the model fine-tuned with RL generates more reasonable texts, i.e., the perplexity is lower. In my RL fine-tuning, the model's output becomes more human-friendly but is less similar to the given title.