ADL21-HW2

B08303028 Hsiang-An Chuang

November 2022

1 Model

1. The model we use in this chinese summarization task is `google/mt5-small`. `mt5-small` is a encoder-decoder Transformer which is a multilingual seq2seq pre-trained model. For every input tokens, they go through encoder, which is stacks of blocks. In each block, there is a self-attention layer followed by a feed-forward neural network, and there is a layer normalization at the end of each block. As for decoder, the only difference is that there is a standard attention mechanism after each self-attention layer and it feed the output of the final block with a linear layer to get the outputs. Thus, the generation strategies can operated on these outputs

2. As for preprocessing, mostly I apply `preprcess_function` in `run_summarization.py` provided by `huggingface`.

```python
def preprocess_function(examples):
        # remove pairs where at least one record is None
        prefix = ""
        padding = 'max_length'
        max_target_length = args.max_target_length

        inputs, targets = examples['maintext'], examples['title']

        inputs = [prefix + inp for inp in inputs]
        model_inputs = tokenizer(inputs, max_length=args.max_source_length, padding=padding, truncation=True)

        # Tokenize targets with the `text_target` keyword argument
        labels = tokenizer(text_target=targets, max_length=max_target_length, padding=padding, truncation=True)

        # If we are padding here, replace all tokenizer.pad_token_id in the labels by -100 when we want to ignore
        # padding in the loss.
        if padding == "max_length":
            labels["input_ids"] = [
                [(l if l != tokenizer.pad_token_id else -100) for l in label] for label in labels["input_ids"]
```

```
            ]

        model_inputs["labels"] = labels["input_ids"]
        return model_inputs

```

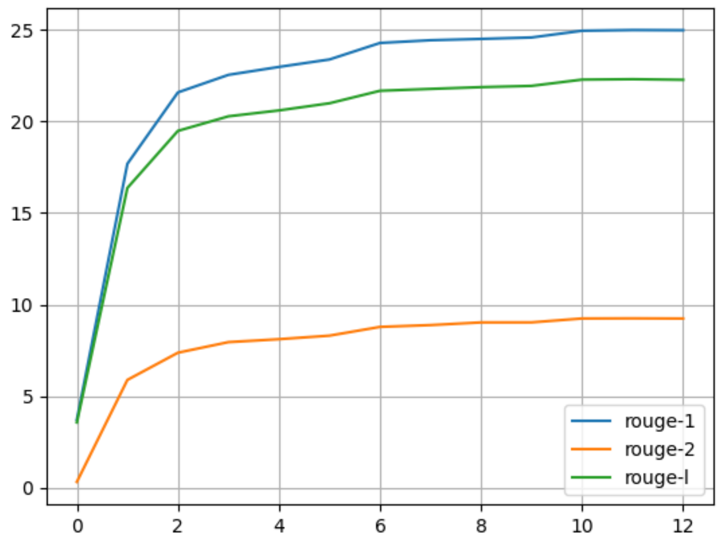
The algorithm simply put maintext, summary(title in our task) together and add a prefix in each text. In term of maintext, it pad all maintext to `max_input_length`, which is 512 by default. As for summary, they are padded to `max_target_length`, which i had tried 64/256 in differenct experiments.

2 Training

1. The hyperparameters I chose mostly consider the problem of 'cuda out of memory'.

```
{
    "num_train_epochs": 30
    "per_device_train_batch_size": 2
    "per_device_eval_batch_size": 2
    "max_source_length": 512
    "max_target_length": 64
    "gradient_accumulation_steps": 16
    "fp16": True
}
```

2. Plot the learning curves (ROUGE versus training steps)



(rouge-1, rouge-2, rouge-l): (24.98, 9.23, 22.28)

3 Generation Strategies

1. Stratgies

   - **Greedy**: Choose the most probable word for each output, and not consider the global maximum.
   - **Beam Search**: Keep track of the k-most probable sequences and finding a better one.
   - **Top-k Sampling**: Kind of sampling, sample the word via distribution but restricted to the top-k probable words.
   - **Top-p Sampling**: Kind of sampling, sample from a subset of vocabulary with the most probability mass.
   - **Temperature**: Apply hyperparameter τ in softmax function. Set a higher value to smooth the distribution.

2. Hyperparameters

Strategy	rouge-1	rouge-2	rouge-l
Greedy	0.244	0.091	0.232
Beam Search(beam=3)	**0.267**	0.104	0.238
Beam Search(beam=5)	0.266	**0.106**	**0.239**
Top-k Sampling(k=10)	0.231	0.068	0.193
Top-k Sampling(k=20)	0.219	0.064	0.184
Top-p Sampling(p=0.8)	0.187	0.053	0.162
Top-p Sampling(p=0.9)	0.175	0.051	0.151
Temperature(τ=0.5)	0.249	0.081	0.210
Temperature(τ=0.8)	0.249	0.081	0.210

My final generation strategy is **Beam Search(beam=5)**.