NTU ADL hw2 Report

Q1: Model

Model

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

> I use `google/mt5-small` pretrained model.
> It’s a multilingual version `t5` model. It basically has the same model architecture as `t5` but trained with multilingual corpus.
> It’s a classic encoder-decoder architecture. Encoder fisrt encodes `maintext` into embedding. Then, decoder will takes `encoder_embedding` and `BOS` token, and then predict the token probability of next sequence. The step by step generate the tokens based on different generation strategies (e.g. beam search, greedy, top_p sampling).
> During training the decoder uses attention mask to simulate the seqeunce. In simple words, the decoder will only takes the seqeunce embedding of all sequences tokens before current position. By using this, the training phase can have the same behavior as inferencing phase.
> It’s seq2seq task. The model takes `maintext` , which is the news content, as input, the news title as expected output.

Preprocessing

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

> Tokenizer has few differences compared to hw1's tasks.
> Since `mt5-small` is larger (encoder-decoder, not just encoder) and the news content may be quite long.
> So we have the limit the token max length. As TA's suggestion, I use `max_source_length` 256.
> In the transformers summarization example codes. It says that adding 'summarizes: ' as context prefix may increases the model performance. So I also add that.
> Data cleaning may be quite effective in this news title summarization task. But I didn't find any simple and usable methods for cleaning data. So I didn't use data cleaning. The model will only sees the first 256 tokens.

Q2: Training

Hyperparameter

1. **Describe your hyperparameter you use and how you decide it.**

> The training hyperparameters I used is listed below.

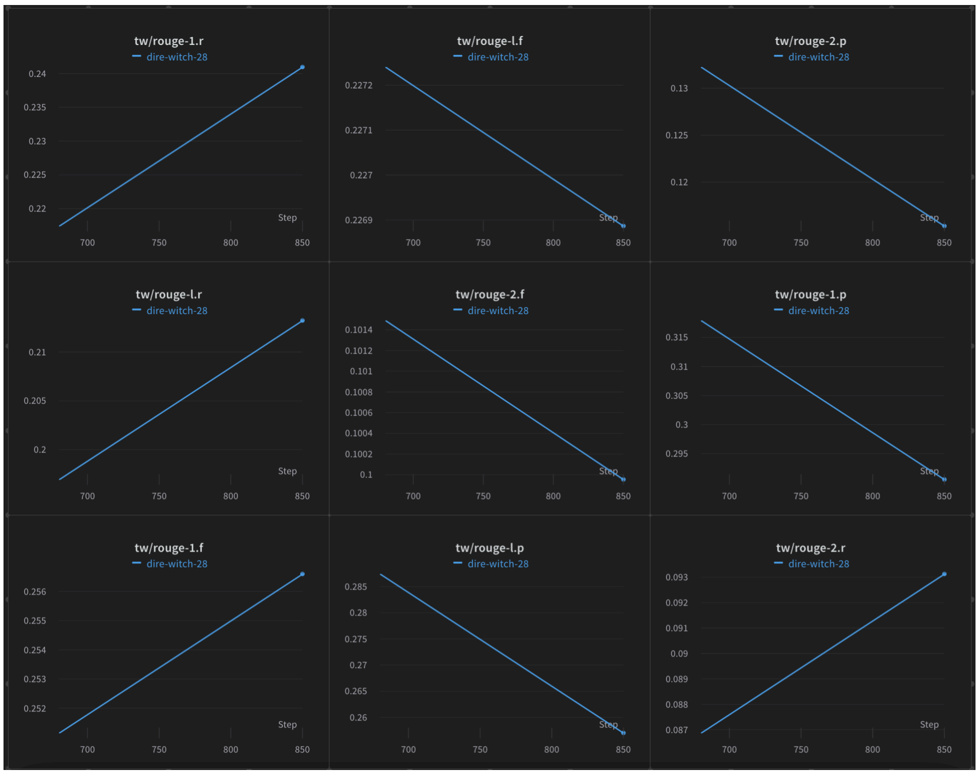
Name	**Value**
num_devices	2
gradient_accumulation_steps	32
per_device_train_batch_size	2
per_device_train_batch_size	2
num_train_epochs	5
learning_rate	2e-5
lr_scheduler_type	cosine
weight_decay	5e-4
max_source_length	256
max_target_length	64
num_beams	30
pad_to_max_length	false

> I’ve tried many hyperparameter combiniation for this model, including learning_rate, lr_scheduler_type, larger batch size, num_beams, etc… Finally I’ve tried out this set of parameter can easily beat the baseline by a large margin.

Learning Curves

1. **Plot the learning curves (ROUGE versus training steps)**

> Below is the learning curves. Step 850 is end of epoch 5. As the chart dipicts that the f1-socre reaches its best performance at epoch 4. And finda overfitted start from epoch 5.
> While rouge-1.f1-score still increases, however, rouge-2.f1-score and rouge-l.f1-score decreases. So I stopped the training right there.



Q3: Generation Strategies

Stratgies

1. **Describe the detail of the following generation strategies**

Hyperparameters

1. **Try at least 2 settings of each strategies and**

compare the result.

Below generation strategies list are all methods I tried.
By comparing the results, I choose the best one for my submission code.
Beam search generally outperforms others.
But top_k and top_p is quite parameter sensitive. I kinda afraid that it’s result not good because of I set the wrong parameter for those methods.

```python
gen_list = {
    'beam_search_5': {
        'max_length': args.val_max_target_length,
        'num_beams': 5
    },
    'beam_search_20_no_repeat_2_gram': {
        'max_length': args.val_max_target_length,
        'num_beams': 20,
        'no_repeat_ngram_size': 2
    },
    'beam_search_30_no_repeat_2_gram_early_stopping': {
        'max_length': args.val_max_target_length,
        'num_beams': 30,
        'no_repeat_ngram_size': 2,
        'early_stopping': True
    },
    'greedy': {
        'max_length': args.val_max_target_length,
        'num_beams': 1
    },
    'greedy_no_repeat_2_gram': {
        'max_length': args.val_max_target_length,
        'num_beams': 1,
        'no_repeat_ngram_size': 2
    },
    'top_k_0': {
        'max_length': args.val_max_target_length,
        'do_sample': True,
        'top_k': 0
    },
    'top_k_0_temp_0.8': {
        'max_length': args.val_max_target_length,
        'do_sample': True,
        'top_k': 0,
        'temperature': 0.8
    },
    'top_k_0_temp_0.9': {
        'max_length': args.val_max_target_length,
        'do_sample': True,
        'top_k': 0,
        'temperature': 0.9
    },
    'top_k_5': {
        'max_length': args.val_max_target_length,
        'do_sample': True,
        'top_k': 5
    },
    'top_k_10': {
        'max_length': args.val_max_target_length,
        'do_sample': True,
        'top_k': 10
    },
    'top_p_0.95': {
        'max_length': args.val_max_target_length,
        'do_sample': True,
        'top_p': 0.95,
        'top_k': 0
    },
    'top_p_0.98': {
        'max_length': args.val_max_target_length,
        'do_sample': True,
        'top_p': 0.98,
        'top_k': 0
    },
}
```

> Below table is the result of the generation strategies I’ve tried.

```
{
  "beam_search_5": {
    "rouge-1": {
      "r": 0.2457426285633042,
      "p": 0.2993114575413157,
      "f": 0.26109382648769974
    },
    "rouge-2": {
      "r": 0.09471835354893139,
      "p": 0.11146204550078927,
      "f": 0.09894193579366511
    },
    "rouge-l": {
      "r": 0.21900578251793523,
      "p": 0.267395197864463,
      "f": 0.23275299426999962
    }
  },
  "beam_search_20_no_repeat_2_gram": {
    "rouge-1": {
      "r": 0.25133119224300543,
      "p": 0.2881018477131765,
      "f": 0.26070050552047863
    },
    "rouge-2": {
      "r": 0.09673789622778171,
      "p": 0.11391090624047692,
      "f": 0.10113962439941689
    },
    "rouge-l": {
      "r": 0.22130831217709543,
      "p": 0.2537757085657312,
      "f": 0.2294128965294994
    }
  },
  "beam_search_30_no_repeat_2_gram_early_stopping": {
    "rouge-1": {
      "r": 0.2409749131614148,
      "p": 0.29054924526570536,
      "f": 0.25660625365317274
    },
    "rouge-2": {
      "r": 0.09313464148591864,
      "p": 0.11533273022882702,
      "f": 0.09995357726941075
    },
    "rouge-l": {
      "r": 0.21324528678833915,
      "p": 0.2570134916377498,
      "f": 0.22688639858646478
    }
  },
  "greedy": {
    "rouge-1": {
      "r": 0.21826234413143197,
      "p": 0.2829041239717245,
      "f": 0.23823516887117246
    },
    "rouge-2": {
      "r": 0.07692125757460207,
      "p": 0.09394166689617224,
      "f": 0.08177733310641652
    },
    "rouge-l": {
      "r": 0.1949033806457341,
      "p": 0.25303635956937015,
      "f": 0.2127063534710711
    }
  },
  "greedy_no_repeat_2_gram": {
    "rouge-1": {
      "r": 0.22322731185146139,
      "p": 0.26374483669899057,
      "f": 0.23492734061649106
    },
    "rouge-2": {
      "r": 0.07508324595789174,
      "p": 0.08617709362881067,
      "f": 0.07765597522951076
    },
    "rouge-l": {
      "r": 0.19490489946362513,
      "p": 0.23031116471428395,
      "f": 0.2049986788660036
    }
  },
  "top_k_0": {
    "rouge-1": {
      "r": 0.1581819686103622,
      "p": 0.16450688071193104,
      "f": 0.15571348334216956
    },
    "rouge-2": {
      "r": 0.042507176012482786,
      "p": 0.042900993241695426,
      "f": 0.04108915287879166
    },
    "rouge-l": {
      "r": 0.13992903424287978,
      "p": 0.1456954402932762,
      "f": 0.13770477844641643
    }
  },
  "top_k_0_temp_0.8": {
    "rouge-1": {
      "r": 0.18406194581707014,
      "p": 0.21019262089521878,
      "f": 0.19003198062718127
    },
    "rouge-2": {
      "r": 0.05581795995905348,
      "p": 0.06178529860394363,
      "f": 0.05658656735359137
    },
    "rouge-l": {
      "r": 0.16328309168488925,
      "p": 0.18714504596099046,
      "f": 0.1687354909405109
    }
  },
  "top_k_0_temp_0.9": {
    "rouge-1": {
      "r": 0.17364786938815732,
      "p": 0.19004383508466027,
      "f": 0.17568828224919847
    },
    "rouge-2": {
      "r": 0.05098226448968667,
      "p": 0.05433078937286188,
      "f": 0.05074012462577128
    },
    "rouge-l": {
      "r": 0.15363831091034685,
      "p": 0.16855326134195056,
      "f": 0.1555413480198916
    }
  },
  "top_k_5": {
    "rouge-1": {
      "r": 0.20709258427176244,
      "p": 0.24678983450878333,
      "f": 0.21804226901686255
    },
    "rouge-2": {
      "r": 0.06626063950161308,
      "p": 0.07472567193875299,
      "f": 0.06765512160758949
    },
    "rouge-l": {
      "r": 0.182725630376137,
      "p": 0.21776257961128412,
      "f": 0.19223445506450226
    }
  },
  "top_k_10": {
    "rouge-1": {
      "r": 0.19998542335970051,
      "p": 0.23171566236598115,
      "f": 0.2077010517265036
    },
    "rouge-2": {
      "r": 0.062384166857273014,
      "p": 0.06786807028223599,
      "f": 0.06269070504646339
    },
    "rouge-l": {
      "r": 0.17564843519501833,
      "p": 0.20397889625245524,
      "f": 0.18246982293796624
    }
  },
  "top_p_0.95": {
      "rouge-1": {
        "r": 0.1654873054309177,
        "p": 0.1760196408422681,
        "f": 0.16510268313352383
      },
      "rouge-2": {
        "r": 0.04738654876818313,
        "p": 0.04906654001617522,
        "f": 0.04648799484499182
      },
      "rouge-l": {
        "r": 0.14598879311285073,
        "p": 0.1556214066126723,
        "f": 0.14568971803607203
      }
    }
  }
```

2. **What is your final generation strategy? (you can combine any of them)**

> My final generation strategy is based on the previous experiement's result. I decided to use beam search with early stopping, no repeat 2 gram and beam size 30.
> As the experiement's result, the chosen generate method can perform best in this news title summarization.

Bonus: Applied RL on Summarization

Algorithm

1. **Describe your RL algorithms, reward function, and hyperparameters.**

I use vanilla policy gradient reinforcement learning algorithm for this title summarization task. The observations are the decoder inputs, which is the layer 0 of `decoder_hidden_states` , and the actions are the decoder outputs, which is the last layer of `decoder_hidden_states` .
And the model is trained using a reward mechanism, which is based on the rouge score, especially tw_rouge. This metric is for evalutating the quality of summarized title and it rewards the generated text if it is similar to the reference text.
The reward is calculates using both rouge-1, rouge-2 and rouge-l scores.
Hyperparameters used include a batch size of 1 (due to hardward limit), a discount factor of 0.99, and number of episodes set to 1500. The learning rate of the model is 3e-5, which is set for the optimizer. The model achieves early stopping if the no-repeat-ngram-size (set to 2) is reached during training, and used a beam search strategy with num_beams set to 15 (due to training speed) during decoding.

Compare to Supervised Learning

1. **Observe the loss, ROUGE score and output texts, what differences can you find?**

I trained using the vanilla policy gradient for 100 episodes. Each episode is a entire train datasets. Observations and actions are stored during the episode. Then the rewards are calculated based on the TA's instruction on the homework slides.
The training loss gradually decays, however, the output text is kind weird, it start outputing the certain token. And of course the rouge-score isn't going to be well, so I think my implementation of reinforcement learning may still exists some problems.
But, by this homework, the implementation of this task let me learned a lot. Including the reinforcement learning techniques, how to implement this algorithm by hand.