ADL Homework II

Wei-Hsiang Huang
B09202011

2023 年 11 月 4 日

Q1: Model

Model

The main structures of "mt5-small" are:

1. Input Embedding System:
   We will give the model our tokens as input. Then, the model will first do the subwords tokenizing. And the model will convert the words into vectors word by Word Embedding and Position Embedding (represent the posistion of each subtokens in the sentence).

2. Encoder Structure: After embedding, the input vectors will go through an encoder, which contains self-attention first and several feed-forward neuron nets.

3. Pretraining.

And for my model, I fine-tuned the pre-trained "mt5-small" in order to suit my tasks. What I do is to make the model predict to the output "title" for the input "maintext". Therefore, the "input_ids" is the "maintext", "label" is the title. After training, I decoded the output "label" and abandoned "extra_id" to be my final predicted "title".

Preprocessing

For the preprocessing, I first did the padding and masking, trying to make the length of tokens in each step remain the same and let the model ignore the padding tokens. Then, I

tried to add starting token and end token in front of and behind the whole sentences respectively. Finally, I organized data into batches. (For the training dataset, I also did the random shuffling.)

Q2: Training

Hyperparameters

For the hyperparameters setting, I refered to hw1, setting the learning rate to be 3×10^{-5} and used a linear learning rate scheduler (decay per steps, not per epochs). The total batch size is 8. (I don't use any gradient accumulated batch.) In order to have a taste on how well an epoch may have. I tried to train 1 epoch two times from the pre-trained "mtf-small". (i.e: total 2 epochs, but training in different times.) Then the public results were (with beams=10, top_p=0.9, temperature=1.0):

Status	Rouge-1	Rouge-2	Rouge-L
First	0.1595	0.0580	0.1522
Second	0.1721	0.0644	0.1625

This result is obviously to be far from the baseline. Therefore, I decided to fine-tuned the model from "Status=Second", with 5 epochs. Here are the public results (with beams=10, top_p=0.9, temperature=1.0):

Rouge-1	Rouge-2	Rouge-L
0.2272	0.0873	0.2068

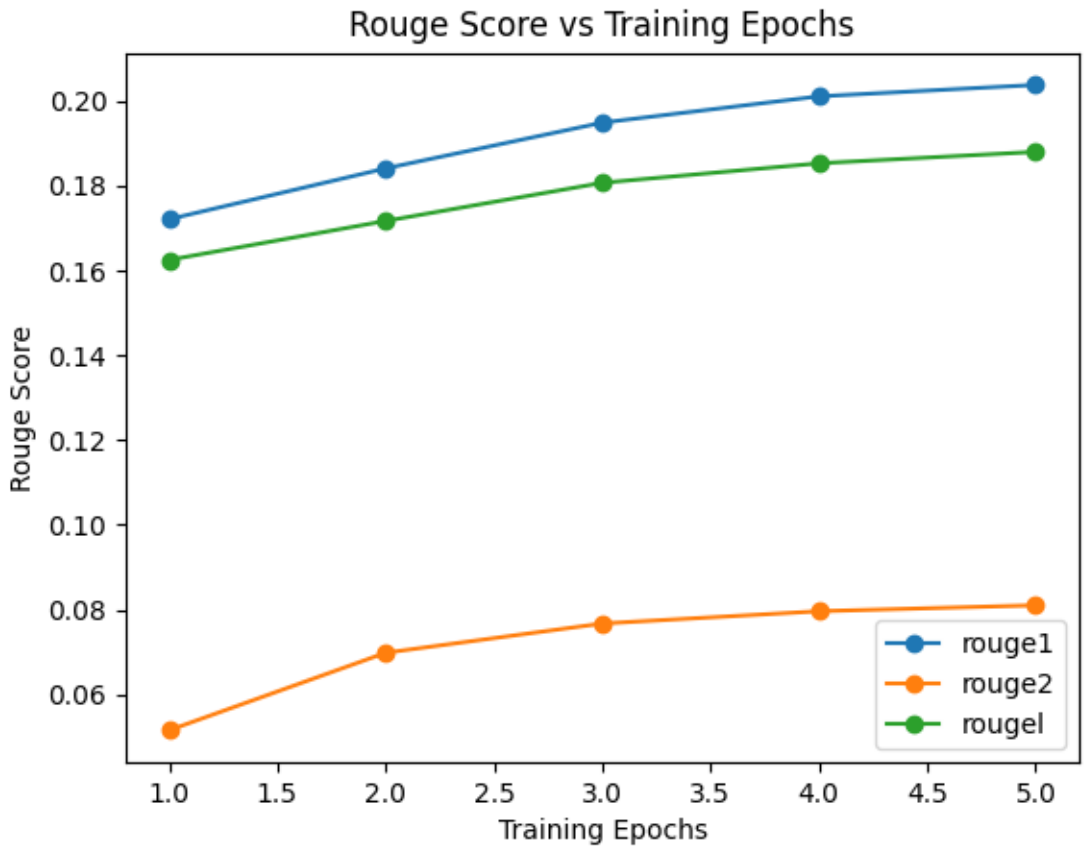
which passed the baseline.
Out of curiosity I trained the model with extra 3 epochs. Here are the public results. (with beams=10, top_p=0.9, temperature=1.0)

Rouge-1	Rouge-2	Rouge-L
0.2367	0.0927	0.2144

which is better than previous results. Therefore, I then decided to use this model for my generations.

Learning Curves

For the learning curves, I retrained mymodel from the pretrained "mtf-small" with 5 epochs and a linear learning rate scheduler to get a more consistent and meaningful curves. Here is the result:



Rouge Score v.s Training Epochs

Q3: Generation Strategies

Strategies

Optimization of the title is a Dynamic Programming problem, which we want to maximize the probability from the beginning of the sentence to the end. Since the true solutions are too time-consuming and too deterministic (which we, human, tend to speak with diversity). Therefore, they are several ways to approximate the optimal solutions or to do some sampling to consider the diversity.

- **Greedy:** Only consider the most probable answers in the current path. Blind all the results in previous path.
- **Beam Search:** With a parameter "num_beams", in each step, record the paths of the top-"num_beams" probability. Based on those paths, finding the possible next paths, comparing the probablity in each path and make the ordering to reserve only the paths of the top-"num_beams" probability. Finally, when encountering EOF, choose the best one. This will get a more optimized solution to greedy with the cause of more computational effort.
- **Sampling:** Instead of finding the best solution, doing the sampling for each predicted paths, each process (including inner process) in beam search.
- **Top-k Sampling:** In each sampling, only preserve the words of top-k probability, renormalizing the probability of them. Top-k sampling can prevent from sampling the noise. (those with low probability.)
- **Top-p Sampling:** In each sampling, only consider the word with total accumulated probability to be smaller than "p" (or just crossed "p"). For example, the probability of "cat", "dog", "egg" is 0.6, 0.3, 0.1. We set "p=0.9". Then we will abandon "egg" in the smapling. Since the accumulated probability for "dog" is 0.6+0.3=0.9, which is just crossed "0.9". Combining the top-p sampling and top-k sampling, we can truly avoid sample noise to our text generations.
- **Temperature:** Temperature is a parameter used for adjust the probability of each word. If we set a higher temperature (i.e: $t > 1$), then the probability distribution will become flat, which means the sampling can be more diverse. In contrast, if we dig into a lower

temperature (i.e: $t < 1$), the probability distribution will be more narrow, leading to a more conservative model generations.

Hyperparameters of Generations

I attempted the following 7 ways of generations:

The following results are the one with 1+1+5 epochs (see Training):

Strategies	Rouge-1	Rouge-2	Rouge-L
Only Greedy	0.2104	0.0756	0.1924
Only Beam Search (n=5)	0.2261	0.0888	0.2058
Beam Search (n=5) and Sampling (k=10, p=0.9, t=1)	0.2247	0.0860	0.2048
Beam Search (n=5) and Sampling (k=10, p=0.9, t=1.2)	0.2252	0.0862	0.2048
Only Beam Search (n=10)	0.2263	0.0897	0.2063
Beam Search (n=10) and Sampling (k=10, p=0.9, t=1)	0.2272	0.0873	0.2068
Beam Search (n=10) and Sampling (k=10, p=0.9, t=1.2)	0.2277	0.0887	0.2069
Public Baseline	0.22	0.085	0.205

The following results are the one with 1+1+5+3 epochs, the final model (see Training):

Strategies	Rouge-1	Rouge-2	Rouge-L
Only Greedy	0.2216	0.0814	0.2014
Only Beam Search (n=5)	0.2380	0.0951	0.2160
Beam Search (n=5) and Sampling (k=10, p=0.9, t=1)	0.2355	0.0920	0.2127
Beam Search (n=5) and Sampling (k=10, p=0.9, t=1.2)	0.2350	0.0914	0.2127
Only Beam Search (n=10)	0.2370	0.0951	0.2147
Beam Search (n=10) and Sampling (k=10, p=0.9, t=1)	0.2367	0.0927	0.2144
Beam Search (n=10) and Sampling (k=10, p=0.9, t=1.2)	0.2386	0.940	0.2165
Public Baseline	0.22	0.085	0.205

As you can see in the table, only applied Greedy Search is definitely not a good choice. And if I tried to increase my beam search number, the overall performances increase. In my attempt, the most interesting parts are the following two:

1. For beams=5, if doing sampling, the performances decrease for both 1+1+5 and 1+1+5+3. This indicates that in "beams=5", paths with most highest scores are more reliable than random sampling.

2. When doing sampling, the Rouge-2 score decreases. This mean that when doing sampling, the model will probably give us more words which should be contained inside the "title". (beams=10) But somehow, it sabotaged the word-word continuity.

Besides for the above, I have found that when I set temperature to be "1.2", the overall performances increase. (except for beams=5, 1+1+5+3 epochs) This may mean that we need a more diverse sampling when doing the task. (which, I think, is reasonable since the "title" generations are more creative rather than normal text generations in human's world.)
Finally, I chose to use the one with "Beam Search (n=10) and Sampling (k=10, p=0.9, t=1.2)". This is because:

- It passes public baseline.
- Doing sampling can be more robust to the unknown data. Especially, we are told that the private data are the newest one, which may be a little bit different with our public dataset. Sampling can reduce the overfitting effect on our public dataset.
- From my attempt, The performances of model generations can gain by using a temperature "1.2" for beams=10.