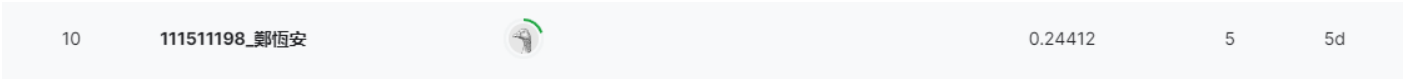
Data Science HW4

Model Compression

Compressed Model Result

Non-zero-ratio = 0.29
Result = 0.24412



In this homework, I use **Pruning** to compress the model.

- To run the model training code, directly exectue **Data_Science_HW4_111511198.ipynb**
- To use trained pruned_model to generate csv result

```
python3 generate_csv.py
```

- To view the score of the generated csv

```
python3 score_result.py
```

Pruning Method

I use **Pytorch built-in package prune** to prune t5 model.

Code(Pruning Part)

```
import torch.nn.utils.prune as prune
from transformers.models.t5.modeling_t5 import T5LayerSelfAttention, T5LayerCrossAttention, T5LayerFF
parameters_to_prune = {
    'encoder_T5LayerSelfAttention':[],
    'encoder_T5LayerFF':[],
    'decoder_T5LayerSelfAttention':[],
    'decoder_T5LayerCrossAttention':[],
    'decoder_T5LayerFF':[],
    'lm_head':[],
}
amounts_to_prune = [
    0.1,
    0.1,
    0.3,
    0.2,
    0.3,
    0.3,
]
for name, module in pruned_model.named_modules():
    part = name.split('.')[0]+'_'
    if isinstance(module, T5LayerSelfAttention):
        for name2, layer in module.named_modules():
            if isinstance(layer, torch.nn.Linear):
                parameters_to_prune[part+'T5LayerSelfAttention'].append((layer, 'weight'))
    elif isinstance(module, T5LayerCrossAttention):
        for name2, layer in module.named_modules():
            if isinstance(layer, torch.nn.Linear):
                parameters_to_prune[part+'T5LayerCrossAttention'].append((layer, 'weight'))
    elif isinstance(module, T5LayerFF):
        for name2, layer in module.named_modules():
            if isinstance(layer, torch.nn.Linear):
                parameters_to_prune[part+'T5LayerFF'].append((layer, 'weight'))
    elif isinstance(module, torch.nn.Linear) and name=="lm_head":
        parameters_to_prune['lm_head'].append((module, 'weight'))
for name,amount in zip(parameters_to_prune,amounts_to_prune):
    prune.global_unstructured(
        parameters_to_prune[name],
        pruning_method=prune.L1Unstructured,
        amount=amount,
    )
show_param_ratio(pruned_model)
```

I compress model little by little and prune seperately by the type of layers.

Based on this repo NASH: A Simple Unified Framework of Structured Pruning for Accelerating Encoder-Decoder Language Models (https://github.com/jongwooko/NASH-Pruning-Official/tree/master), they discover that **moderate sparsity in the pruned encoder network enhances generation quality**.

Thus, I seperate encoder and decoder, and use less amount to prune layer in encoder.

Moreover, based on serveral testing and classmates' advice, I know that we should not prune too much CrossAttetion layers and here is why my amounts are set as above.

In Version 1 to 5, I set the amount_to_prune as below.

```
amounts_to_prune = [
    0.1,
    0.1,
    0.3,
    0.2,
    0.3,
    0.3,
]
```

While Version 6, I set the amount_to_prune as below.

```
amounts_to_prune = [
    0,
    0,
    0.1,
    0.1,
    0.1,
    0.2,
]
```

Version	non zero ratio
1	0.77
2	0.60
3	0.48
4	0.38
5	0.31
6	0.29

max_length with Lsum

After reviewing my result, I find that there is a parameter control the generated answers' length called **max_length**. Different max_length will lead to different Lsum score as you can see as below.

```
results = pruned_trainer.predict(tokenized_billsum_test['train'],max_length=128)
```

```
pruned_trainer.evaluate(tokenized_billsum_test['train'],max_length = 20)
```

```
{'eval_loss': 3.982614517211914,
 'eval_rouge1': 0.1934,
 'eval_rouge2': 0.0898,
 'eval_rougeL': 0.1611,
 'eval_rougeLsum': 0.1608,
 'eval_gen_len': 19.0,
 'eval_runtime': 4.4159,
 'eval_samples_per_second': 7.247,
 'eval_steps_per_second': 3.623,
 'epoch': 20.0}
```

```
pruned_trainer.evaluate(tokenized_billsum_test['train'],max_length = 50)
```

```
{'eval_loss': 4.0735673904418945,
 'eval_rouge1': 0.3314,
 'eval_rouge2': 0.1116,
 'eval_rougeL': 0.2501,
 'eval_rougeLsum': 0.2496,
 'eval_gen_len': 48.875,
 'eval_runtime': 7.9999,
 'eval_samples_per_second': 4.0,
 'eval_steps_per_second': 2.0}
```

```
pruned_trainer.evaluate(tokenized_billsum_test['train'],max_length = 70)
```

```
{'eval_loss': 4.0735673904418945,
 'eval_rouge1': 0.3312,
 'eval_rouge2': 0.1059,
 'eval_rougeL': 0.2434,
 'eval_rougeLsum': 0.2436,
 'eval_gen_len': 68.25,
 'eval_runtime': 10.1828,
 'eval_samples_per_second': 3.143,
 'eval_steps_per_second': 1.571}
```

```
pruned_trainer.evaluate(tokenized_billsum_test['train'],max_length=100)
```

```
[16/16 17:06]

{'eval_loss': 4.0735673904418945,
 'eval_rouge1': 0.2946,
 'eval_rouge2': 0.0913,
 'eval_rougeL': 0.2288,
 'eval_rougeLsum': 0.2294,
 'eval_gen_len': 97.3125,
 'eval_runtime': 10.4055,
 'eval_samples_per_second': 3.075,
 'eval_steps_per_second': 1.538}
```

However, because of the limit of my machine and I discover that in preprocess_function that tokenize dataset, we set tokenizer=128, I use max_length=128 to generate my final result.

```
prefix = "summarize: "


def preprocess_function(examples):
    inputs = [prefix + doc for doc in examples["text"]]
    model_inputs = tokenizer(inputs, max_length=1024, truncation=True)

    labels = tokenizer(text_target=examples["summary"], max_length=128, truncation=True)

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