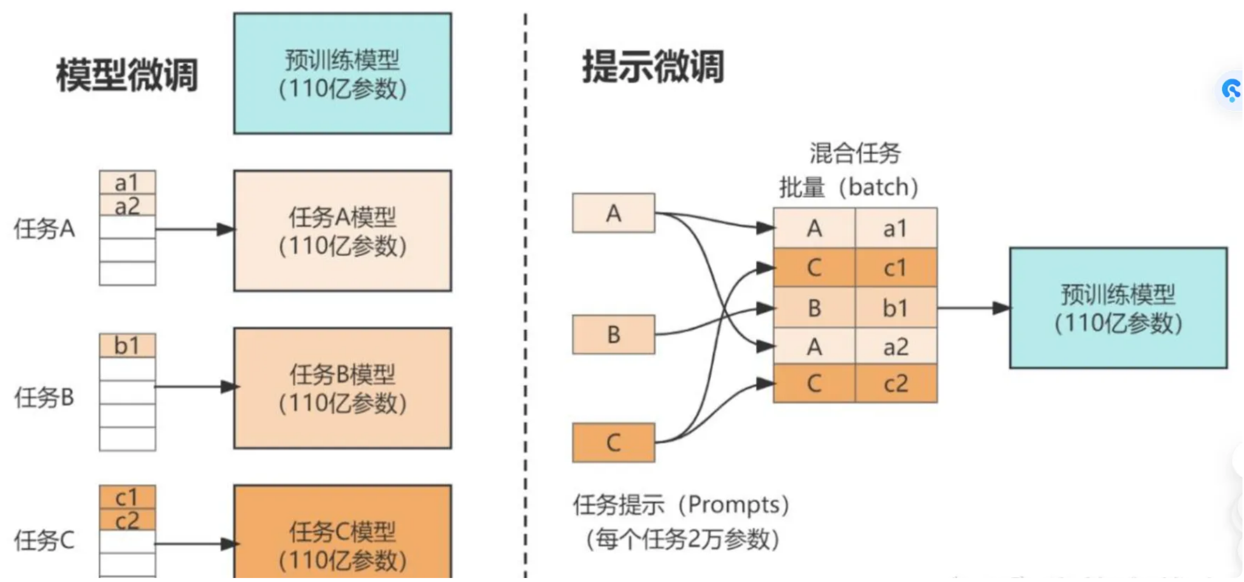
Prompt Tuning详解及代码实战

Prompt Tuning 是2021年谷歌在论文《The Power of Scale for Parameter-Efficient Prompt Tuning》中提出的微调方法。

1. 概念&原理

该方法可以看作是 Prefix Tuning 的简化版本，只在输入层加入 prompt tokens，并不需要加入 MLP 进行调整来解决难训练的问题。主要在 T5 预训练模型上做实验。似乎只要预训练模型足够强大，其他的一切都不是问题。作者也做实验说明随着预训练模型参数量的增加，Prompt Tuning的方法会逼近 Fine-tune 的结果。

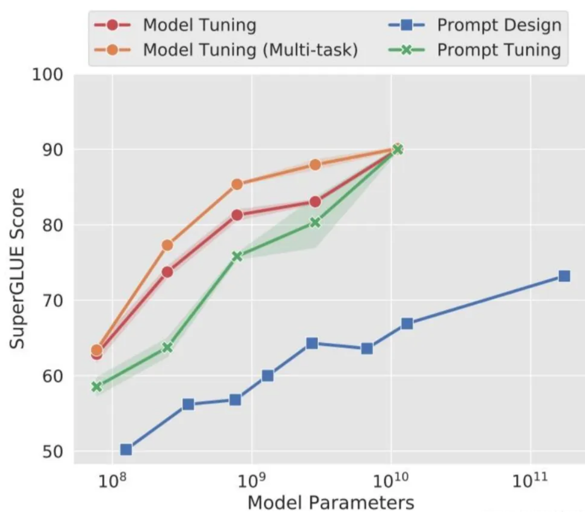
固定预训练参数，为每一个任务额外添加一个或多个 embedding，之后拼接 query 正常输入 LLM，并只训练这些 embedding。左图为单任务全参数微调，右图为 Prompt tuning。



2. 实验论证

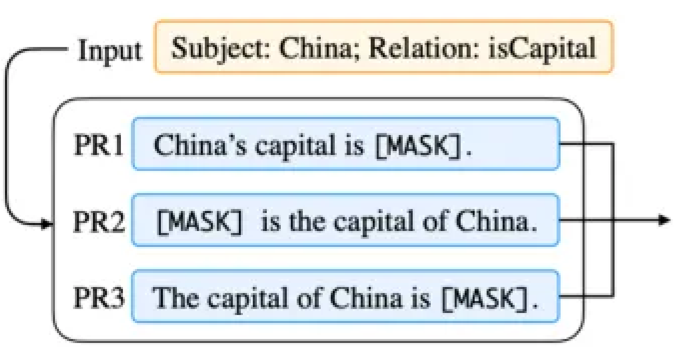
作者做了一系列对比实验，都在说明：随着预训练模型参数的增加，一切的问题都不是问题，最简单的设置也能达到极好的效果。

- **Prompt 长度影响：**模型参数达到一定量级时，Prompt 长度为1也能达到不错的效果，Prompt 长度为20就能达到极好效果。
- **Prompt初始化方式影响：**Random Uniform 方式明显弱于其他两种，但是当模型参数达到一定量级，这种差异也不复存在。
- **预训练的方式：**LM Adaptation 的方式效果好，但是当模型达到一定规模，差异又几乎没有了。
- 微调步数影响：模型参数较小时，步数越多，效果越好。同样随着模型参数达到一定规模，zero shot 也能取得不错效果。
- 当参数达到100亿规模与全参数微调方式效果无异。



3. Prompt Ensembling

同时，Prompt Tuning 还提出了 Prompt Ensembling，**也就是在一个批次（Batch）里同时训练同一个任务的不同 prompt**（即采用多种不同方式询问同一个问题），这样相当于训练了不同模型，比模型集成的成本小多了。



(a) Prompt Ensembling.

4. 代码实践

https://github.com/huggingface/peft/blob/main/src/peft/tuners/prompt_tuning/model.py

训练

```python
from transformers import AutoModelForCausalLM
from peft import get_peft_config, get_peft_model, PromptTuningInit, PromptTuningConfig, TaskType, PeftType
import torch
from datasets import load_dataset
import os
from transformers import AutoTokenizer
from torch.utils.data import DataLoader
from transformers import default_data_collator, get_linear_schedule_with_warmup
from tqdm import tqdm
from datasets import load_dataset


device = "cuda"

model_name_or_path = "/data/nfs/llm/model/bloomz-560m"
tokenizer_name_or_path = "/data/nfs/llm/model/bloomz-560m"

peft_config = PromptTuningConfig(
    task_type=TaskType.CAUSAL_LM,
    prompt_tuning_init=PromptTuningInit.TEXT,
    num_virtual_tokens=8,
    prompt_tuning_init_text="Classify if the tweet is a complaint or not:",
    tokenizer_name_or_path=model_name_or_path,
)

dataset_name = "twitter_complaints"

text_column = "Tweet text"
label_column = "text_label"
max_length = 64
lr = 3e-2
num_epochs = 10
batch_size = 8

from datasets import load_dataset

#dataset = load_dataset("ought/raft", dataset_name)
dataset = load_dataset("/home/guodong.li/data/peft/raft/raft.py", dataset_name, cache_dir="/home/guodong.li/data/peft/data")

classes = [k.replace("_", " ") for k in dataset["train"].features["Label"].names]
```

```
print(classes)

dataset = dataset.map(
    lambda x: {"text_label": [classes[label] for label in x["Label"]]},
    batched=True,
    num_proc=1,
)
print(dataset)

dataset["train"][0]


# data preprocessing
tokenizer = AutoTokenizer.from_pretrained(model_name_or_path)
if tokenizer.pad_token_id is None:
    tokenizer.pad_token_id = tokenizer.eos_token_id

target_max_length = max([len(tokenizer(class_label)["input_ids"]) for class_label in classes])
print("target_max_length:", target_max_length)


# 预处理
def preprocess_function(examples):
    batch_size = len(examples[text_column])
    print("batch_size:", batch_size)

    inputs = [f"{text_column} : {x} Label : " for x in examples[text_column]]
    targets = [str(x) for x in examples[label_column]]

    model_inputs = tokenizer(inputs)
    labels = tokenizer(targets)

    for i in range(batch_size):
        sample_input_ids = model_inputs["input_ids"][i]
        label_input_ids = labels["input_ids"][i] + [tokenizer.pad_token_id]
        if i == 0:
            print(i, sample_input_ids, label_input_ids)
        model_inputs["input_ids"][i] = sample_input_ids + label_input_ids
        labels["input_ids"][i] = [-100] * len(sample_input_ids) + label_input_ids
        model_inputs["attention_mask"][i] = [1] * len(model_inputs["input_ids"][i])
    #print(model_inputs)

    for i in range(batch_size):
```

```python
        sample_input_ids = model_inputs["input_ids"][i]
        label_input_ids = labels["input_ids"][i]

        model_inputs["input_ids"][i] = [tokenizer.pad_token_id] * (max_length - len(sample_input_ids)) + sample_input_ids
        model_inputs["attention_mask"][i] = [0] * (max_length - len(sample_input_ids)) + model_inputs["attention_mask"][i]
        labels["input_ids"][i] = [-100] * (max_length - len(sample_input_ids)) + label_input_ids

        model_inputs["input_ids"][i] = torch.tensor(model_inputs["input_ids"][i][:max_length])
        model_inputs["attention_mask"][i] = torch.tensor(model_inputs["attention_mask"][i][:max_length])
        labels["input_ids"][i] = torch.tensor(labels["input_ids"][i][:max_length])
        if i == 0:
            print("model_inputs input_ids:", model_inputs["input_ids"][i])
            print("model_inputs attention_mask:", model_inputs["attention_mask"][i])
            print("labels input_ids:", labels["input_ids"][i])



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


print("column_names:", dataset["train"].column_names)

# 将原始的训练和测试数据同时预处理，然后作为训练和评估数据集
processed_datasets = dataset.map(
    preprocess_function,
    batched=True,
    num_proc=1,
    remove_columns=dataset["train"].column_names,
    load_from_cache_file=False,
    desc="Running tokenizer on dataset",
)

train_dataset = processed_datasets["train"]
eval_dataset = processed_datasets["train"]

# 训练与评估使用同一份数据，但是训练数据打乱
train_dataloader = DataLoader(train_dataset, shuffle=True, collate_fn=default_data_collator, batch_size=batch_size, pin_memory=True)

```

```python
eval_dataloader = DataLoader(eval_dataset, collate_fn=default_data_collator, batch_size=batch_size, pin_memory=True)
print(len(train_dataloader))
print(len(eval_dataloader))

def test_preprocess_function(examples):
    batch_size = len(examples[text_column])
    inputs = [f"{text_column} : {x} Label : " for x in examples[text_column]]
    model_inputs = tokenizer(inputs)
    # print(model_inputs)
    for i in range(batch_size):
        sample_input_ids = model_inputs["input_ids"][i]

        model_inputs["input_ids"][i] = [tokenizer.pad_token_id] * ( max_length - len(sample_input_ids)) + sample_input_ids
        model_inputs["attention_mask"][i] = [0] * (max_length - len(sample_input_ids)) + model_inputs["attention_mask"][i]

        model_inputs["input_ids"][i] = torch.tensor(model_inputs["input_ids"][i][:max_length])
        model_inputs["attention_mask"][i] = torch.tensor(model_inputs["attention_mask"][i][:max_length])
    return model_inputs

# 将原始的测试数据用于测试
test_dataset = dataset["test"].map(
    test_preprocess_function,
    batched=True,
    num_proc=1,
    remove_columns=dataset["train"].column_names,
    load_from_cache_file=False,
    desc="Running tokenizer on dataset",
)

test_dataloader = DataLoader(test_dataset, collate_fn=default_data_collator, batch_size=batch_size, pin_memory=True)
next(iter(test_dataloader))


# creating model
model = AutoModelForCausalLM.from_pretrained(model_name_or_path)
model = get_peft_model(model, peft_config)
model.print_trainable_parameters()

# model
# optimizer and lr scheduler
optimizer = torch.optim.AdamW(model.parameters(), lr=lr)
```

```python
lr_scheduler = get_linear_schedule_with_warmup(
    optimizer=optimizer,
    num_warmup_steps=0,
    num_training_steps=(len(train_dataloader) * num_epochs),
)

# training and evaluation
model = model.to(device)

for epoch in range(num_epochs):
    model.train()
    total_loss = 0
    for step, batch in enumerate(tqdm(train_dataloader)):
        batch = {k: v.to(device) for k, v in batch.items()}
        #         print(batch)
        #         print(batch["input_ids"].shape)
        outputs = model(**batch)
        loss = outputs.loss
        total_loss += loss.detach().float()
        loss.backward()
        optimizer.step()
        lr_scheduler.step()
        optimizer.zero_grad()

    model.eval()
    eval_loss = 0
    eval_preds = []
    for step, batch in enumerate(tqdm(eval_dataloader)):
        batch = {k: v.to(device) for k, v in batch.items()}
        with torch.no_grad():
            outputs = model(**batch)
        loss = outputs.loss
        eval_loss += loss.detach().float()
        eval_preds.extend(
            tokenizer.batch_decode(torch.argmax(outputs.logits, -1).detach().cpu().numpy(), skip_special_tokens=True)
        )

    eval_epoch_loss = eval_loss / len(eval_dataloader)
    eval_ppl = torch.exp(eval_epoch_loss)
    train_epoch_loss = total_loss / len(train_dataloader)
    train_ppl = torch.exp(train_epoch_loss)
    print(f"{epoch=}: {train_ppl=} {train_epoch_loss=} {eval_ppl=} {eval_epoch_loss=}")
```

推理

Python

```python
from peft import PeftModel, PeftConfig

peft_model_id = f"{model_name_or_path}_{peft_config.peft_type}_{peft_config.task_type}"

# 加载PEFT配置
config = PeftConfig.from_pretrained(peft_model_id)

# 加载基础模型
model = AutoModelForCausalLM.from_pretrained(config.base_model_name_or_path)
# 加载PEFT模型
model = PeftModel.from_pretrained(model, peft_model_id)

# Tokenizer编码
inputs = tokenizer(f'{text_column} : {dataset["test"][i]["Tweet text"]} Label : ', return_tensors="pt")

# 模型推理
outputs = model.generate(
        input_ids=inputs["input_ids"],
        attention_mask=inputs["attention_mask"],
        max_new_tokens=10,
        eos_token_id=3
    )

# Tokenizer 解码
print(tokenizer.batch_decode(outputs.detach().cpu().numpy(), skip_special_tokens=True))
```