

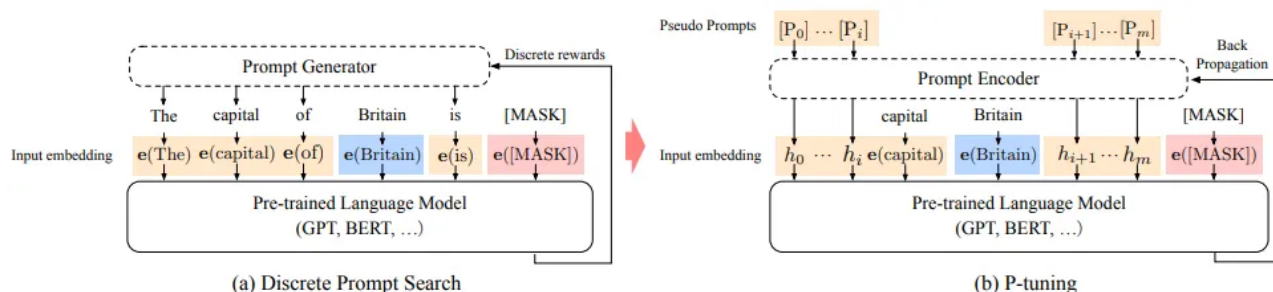
P-Tuning详解及代码实战

1. 概述
2. 方法
3. 代码

1. 概述

P-Tuning (论文: GPT Understands, Too) , 该方法将Prompt转换为可以学习的Embedding层, 并用MLP+LSTM的方式来对Prompt Embedding进行一层处理。

2. 方法



相比Prefix Tuning, P-Tuning加入的可微的virtual token, 但仅限于输入层, 没有在每一层都加; 另外, virtual token的位置也不一定是前缀, 插入的位置是可选的。

如果随机初始化virtual token, 容易优化到局部最优值。因此, 作者通过实验发现用一个prompt encoder来编码会收敛更快, 效果更好。即用一个LSTM+MLP去编码这些virtual token以后, 再输入到模型。

encoder采用LSTM或者MLP:

$$\begin{aligned} h_i &= \text{MLP} \left(\left[\vec{h_i} : \overleftarrow{h_i} \right] \right) \\ &= \text{MLP} ([\text{LSTM}(h_{0:i}) : \text{LSTM}(h_{i:m})]) \end{aligned}$$

3. 代码

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

```
1  from transformers import AutoModelForCausalLM
2  from peft import (
3      get_peft_config,
4      get_peft_model,
5      get_peft_model_state_dict,
6      set_peft_model_state_dict,
7      PeftType,
8      TaskType,
9      PromptEncoderConfig,
10 )
11
12 import torch
13 from datasets import load_dataset
14 import os
15 from transformers import AutoTokenizer
16 from torch.utils.data import DataLoader
17 from transformers import default_data_collator, get_linear_schedule_with_
    warmup
18 from tqdm import tqdm
19 from datasets import load_dataset
20
21
22 device = "cuda"
23
24 model_name_or_path = "/data/nfs/llm/model/bloomz-560m"
25 tokenizer_name_or_path = "/data/nfs/llm/model/bloomz-560m"
26
27 peft_config = PromptEncoderConfig(task_type=TaskType.CAUSAL_LM,
28                                   num_virtual_tokens=20,
29                                   encoder_hidden_size=128)
30
31 dataset_name = "twitter_complaints"
32 checkpoint_name = f"{dataset_name}_{model_name_or_path}_{peft_config.peft_
    _type}_{peft_config.task_type}_v1.pt".replace("/", "_")
33 text_column = "Tweet text"
34 label_column = "text_label"
35 max_length = 64
36 lr = 3e-2
37 num_epochs = 10
38 batch_size = 8
39
40 from datasets import load_dataset
41
42 # dataset = load_dataset("ought/raft", dataset_name)
43
```

```

44 dataset = load_dataset("/home/guodong.li/data/peft/raft/raft.py", dataset
45 _name, cache_dir="/home/guodong.li/data/peft/data")

46 classes = [k.replace("_", " ") for k in dataset["train"].features["Label"]
47 ].names]
48 print(classes)
49 dataset = dataset.map(
50     lambda x: {"text_label": [classes[label] for label in x["Label"]]},
51     batched=True,
52     num_proc=1,
53 )
54 print(dataset)
55 dataset["train"][0]

56 # data preprocessing
57 # padding_side = "left"
58 # tokenizer = AutoTokenizer.from_pretrained(model_name_or_path, padding_s
59 ide=padding_side)
60 tokenizer = AutoTokenizer.from_pretrained(model_name_or_path)
61 if tokenizer.pad_token_id is None:
62     tokenizer.pad_token_id = tokenizer.eos_token_id
63 target_max_length = max([len(tokenizer(class_label)["input_ids"]) for cla
64 ss_label in classes])
65 print("target_max_length:", target_max_length)

66 def preprocess_function(examples):
67     batch_size = len(examples[text_column])
68     inputs = [f"{text_column} : {x} Label : " for x in examples[text_colu
69 mn]]
70     targets = [str(x) for x in examples[label_column]]
71     model_inputs = tokenizer(inputs)
72     labels = tokenizer(targets)
73     for i in range(batch_size):
74         sample_input_ids = model_inputs["input_ids"][i]
75         label_input_ids = labels["input_ids"][i] + [tokenizer.pad_token_i
76 d]
77         # print(i, sample_input_ids, label_input_ids)
78         model_inputs["input_ids"][i] = sample_input_ids + label_input_ids
79         labels["input_ids"][i] = [-100] * len(sample_input_ids) + label_i
80 nput_ids
81         model_inputs["attention_mask"][i] = [1] * len(model_inputs["input
82 _ids"][i])
83         # print(model_inputs)
84         for i in range(batch_size):
85             sample_input_ids = model_inputs["input_ids"][i]
86             label_input_ids = labels["input_ids"][i]
87             model_inputs["input_ids"][i] = [tokenizer.pad_token_id] * (

```

```

84         max_length - len(sample_input_ids)
85     ) + sample_input_ids
86     model_inputs["attention_mask"][i] = [0] * (max_length - len(sampl
87 e_input_ids)) + model_inputs[
88         "attention_mask"
89     ][i]
90     labels["input_ids"][i] = [-100] * (max_length - len(sample_input_
91 ids)) + label_input_ids
92     model_inputs["input_ids"][i] = torch.tensor(model_inputs["input_i
93 ds"][i][:max_length])
94     model_inputs["attention_mask"][i] = torch.tensor(model_inputs["at
95 tention_mask"][i][:max_length])
96     labels["input_ids"][i] = torch.tensor(labels["input_ids"][i][:max
97 _length])
98     model_inputs["labels"] = labels["input_ids"]
99     return model_inputs
100
101 processed_datasets = dataset.map(
102     preprocess_function,
103     batched=True,
104     num_proc=1,
105     remove_columns=dataset["train"].column_names,
106     load_from_cache_file=False,
107     desc="Running tokenizer on dataset",
108 )
109
110 train_dataset = processed_datasets["train"]
111 eval_dataset = processed_datasets["train"]
112
113
114 train_dataloader = DataLoader(train_dataset, shuffle=True, collate_fn=def
115 ault_data_collator, batch_size=batch_size, pin_memory=True)
116 eval_dataloader = DataLoader(eval_dataset, collate_fn=default_data_collat
117 or, batch_size=batch_size, pin_memory=True)
118
119
120 def test_preprocess_function(examples):
121     batch_size = len(examples[text_column])
122     inputs = [f"{text_column} : {x} Label : " for x in examples[text_colu
123 mn]]
124     model_inputs = tokenizer(inputs)
125     # print(model_inputs)
126     for i in range(batch_size):
127         sample_input_ids = model_inputs["input_ids"][i]
128         model_inputs["input_ids"][i] = [tokenizer.pad_token_id] * (max_le
129 ngth - len(sample_input_ids)) + sample_input_ids

```

```

122         model_inputs["attention_mask"][i] = [0] * (max_length - len(sampl
123 e_input_ids)) + model_inputs["attention_mask"][i]

124         model_inputs["input_ids"][i] = torch.tensor(model_inputs["input_i
125 ds"][i][:max_length])
126         model_inputs["attention_mask"][i] = torch.tensor(model_inputs["at
127 tention_mask"][i][:max_length])
128         return model_inputs
129
130 test_dataset = dataset["test"].map(
131     test_preprocess_function,
132     batched=True,
133     num_proc=1,
134     remove_columns=dataset["train"].column_names,
135     load_from_cache_file=False,
136     desc="Running tokenizer on dataset",
137 )

138 test_dataloader = DataLoader(test_dataset, collate_fn=default_data_collat
139 or, batch_size=batch_size, pin_memory=True)
140 next(iter(test_dataloader))
141
142 model = AutoModelForCausalLM.from_pretrained(model_name_or_path)
143 model = get_peft_model(model, peft_config)
144 model.print_trainable_parameters()
145
146 # model
147 # optimizer and lr scheduler
148 optimizer = torch.optim.AdamW(model.parameters(), lr=lr)
149 lr_scheduler = get_linear_schedule_with_warmup(
150     optimizer=optimizer,
151     num_warmup_steps=0,
152     num_training_steps=(len(train_dataloader) * num_epochs),
153 )
154
155 # training and evaluation
156 model = model.to(device)
157
158 for epoch in range(num_epochs):
159     model.train()
160     total_loss = 0
161     for step, batch in enumerate(tqdm(train_dataloader)):
162         batch = {k: v.to(device) for k, v in batch.items()}
163         #         print(batch)
164         #         print(batch["input_ids"].shape)
165         outputs = model(**batch)
166         loss = outputs.loss

```

```

167         total_loss += loss.detach().float()
168         loss.backward()
169         optimizer.step()
170         lr_scheduler.step()
171         optimizer.zero_grad()
172
173     model.eval()
174     eval_loss = 0
175     eval_preds = []
176     for step, batch in enumerate(tqdm(eval_dataloader)):
177         batch = {k: v.to(device) for k, v in batch.items()}
178         with torch.no_grad():
179             outputs = model(**batch)
180             loss = outputs.loss
181             eval_loss += loss.detach().float()
182             eval_preds.extend(
183                 tokenizer.batch_decode(torch.argmax(outputs.logits, -1).detach().cpu().numpy(), skip_special_tokens=True)
184             )
185
186     eval_epoch_loss = eval_loss / len(eval_dataloader)
187     eval_ppl = torch.exp(eval_epoch_loss)
188     train_epoch_loss = total_loss / len(train_dataloader)
189     train_ppl = torch.exp(train_epoch_loss)
190     print(f"epoch={epoch}: {train_ppl=} {train_epoch_loss=} {eval_ppl=} {eval_
epoch_loss=}")

```