

pretrain_sft

[pretrain 理论](#)

[sft 理论](#)

[pretrain 代码](#)

[sft 代码](#)

[chat template](#)

[sft 数据](#)

[数据任务](#)

[数据形式](#)

[数据生产](#)

[sft 参数](#)

[sft 技巧](#)

[拟合性](#)

[欠拟合](#)

[过拟合](#)

[评估](#)

[Lora](#)

[原理](#)

[初始化](#)

[超参](#)

pretrain 理论

初始将 `input_ids` 复制一份作为 `labels`

input_ids	bos	t1	t2	t3	t4	t5
-----------	-----	----	----	----	----	----

labels	bos	t1	t2	t3	t4	t5
--------	-----	----	----	----	----	----

取 `logits[:-1]` 和 `labels[1:]` 计算 loss

logits	bos	t1	t2	t3	t4
labels	t2	t3	t4	t5	eos

预测关系如图所示

input_ids	bos	t1	t2	t3	t4	t5
labels	bos	t1	t2	t3	t4	t5

sft 理论

初始将 `input_ids` 复制一份作为 `labels`，`p1-p2` 代表 prompt，`r1-r3` 代表 response，`labels` 只保留 response 以后的部分，其他使用 `-100` 进行 mask，期望模型从 prompt 之后开始学习

为什么：因为 sft 的 prompt 同质化严重

input_ids	bos	p1	p2	r1	r2	r3	eos
labels	-100	-100	-100	r1	r2	r3	eos

input_ids	bos	p1	p2	r1	r2	r3	eos
labels	-100	-100	-100	r1	r2	r3	eos

因此，真实计算 loss 时的对应关系为

logits	p2	r1	r2	r3
labels	r1	r2	r3	eos

pretrain 代码

```
1  from tqdm import tqdm
2  import copy
3  import logging
4  from dataclasses import dataclass, field
5  from typing import Dict, Optional, Sequence
6
7  import torch
8  import transformers
9  from torch.utils.data import Dataset
10 from transformers import Trainer
11 from datasets import load_dataset
12 from typing import List
13 import os
14 import logging
15 from transformers import DataCollatorForSeq2Seq, default_data_collator, DataCollatorForLanguageModeling
16 from functools import partial
17 import os
18 os.environ['CUDA_VISIBLE_DEVICES'] = '0'
19 logger = logging.getLogger(__name__)
20
21
22 def get_all_datapath(dir_name: str) -> List[str]:
23     all_file_list = []
24     # all_file_size = []
25
26     for (root, dir, file_name) in os.walk(dir_name):
27         for temp_file in file_name:
28             standard_path = f"{root}/{temp_file}"
29
30             all_file_list.append(standard_path)
31
32     return all_file_list
33
34
35 def load_dataset_from_path(data_path: Optional[str] = None,
36                             cache_dir: Optional[str] = "cache_data",
37                             data_file_number: Optional[int] = 2,
38                             use_streaming: bool = False) -> Dataset:
39     all_file_list = get_all_datapath(data_path)[:data_file_number]
40     data_files = {'train': all_file_list}
41     extension = all_file_list[0].split(".")[1]
42
43     logger.info("load files %d number", len(all_file_list))
```

```

44
45
46
47     raw_datasets = load_dataset(
48         extension,
49         data_files=data_files,
50         cache_dir=cache_dir,
51         streaming=use_streaming
52     )['train']
53     return raw_datasets
54
55
56 IGNORE_INDEX = -100
57
58
59 @dataclass
60 class ModelArguments:
61     model_name_or_path: Optional[str] = field(default="facebook/opt-125
62         m")
63
64 @dataclass
65 class DataArguments:
66     data_path: str = field(default=None, metadata={
67         "help": "Path to the training data."})
68     data_num_limit: int = field(default=None, metadata={
69         "help": "the numbers of data file"
70     })
71     data_proc_num: int = field(default=None, metadata={
72         "help": "the numbers of process"
73     })
74     use_streaming: bool = field(default=False, metadata={
75         "help": "use stream mode to process big dat
76         a"})
77
78 @dataclass
79 class TrainingArguments(transformers.TrainingArguments):
80     cache_dir: Optional[str] = field(default=None)
81     optim: str = field(default="adamw_torch")
82     model_max_length: int = field(
83         default=512,
84         metadata={
85             "help": "Maximum sequence length. Sequences will be right pad
86             ded (and possibly truncated)."},
87         )

```

```

88
89 def make_train_dataset(tokenizer: transformers.PreTrainedTokenizer, data_
    path: str, data_file_number: int, data_proc_num: int, use_streaming: boo
    l) -> Dataset:
90     logging.warning("Loading data...")
91
92     dataset = load_dataset_from_path(
93         data_path=data_path,
94         data_file_number=data_file_number,
95         use_streaming=use_streaming
96     )
97     logging.warning("Formatting inputs...")
98
99     def generate_sources_targets(examples: Dict, tokenizer: transformers.
    PreTrainedTokenizer):
100         ins_data = examples['content']
101
102         input_output = tokenizer(ins_data,
103                                 return_tensors="pt",
104                                 padding="longest",
105                                 max_length=tokenizer.model_max_length-1,
106                                 truncation=True)
107         examples['input_ids'] = input_output['input_ids']
108         return examples
109
110     generate_sources_targets_p = partial(
111         generate_sources_targets, tokenizer=tokenizer)
112
113     if use_streaming:
114         dataset = dataset.map(
115             function=generate_sources_targets_p,
116             batched=True
117         ).shuffle(42, buffer_size=50000)
118     else:
119         dataset = dataset.map(
120             function=generate_sources_targets_p,
121             batched=True,
122             desc="Running tokenizer on train dataset",
123             num_proc=data_proc_num
124         ).shuffle()
125
126     return dataset
127
128
129 def train():
130     parser = transformers.HfArgumentParser(
131         (ModelArguments, DataArguments, TrainingArguments))

```

```

132     model_args, data_args, training_args = parser.parse_args_into_dataclasses()
133
134     model = transformers.AutoModelForCausalLM.from_pretrained(
135         model_args.model_name_or_path,
136         cache_dir=training_args.cache_dir,
137         device_map='auto',
138         torch_dtype=torch.bfloat16
139
140     )
141
142     # model.is_parallelizable = True
143     # model.model_parallel = True
144     torch.cuda.empty_cache()
145
146     tokenizer = transformers.LlamaTokenizer.from_pretrained(
147         model_args.model_name_or_path,
148         cache_dir=training_args.cache_dir,
149         model_max_length=training_args.model_max_length,
150         padding_side="right",
151         use_fast=True,
152     )
153
154     train_dataset = make_train_dataset(
155         tokenizer=tokenizer,
156         data_path=data_args.data_path,
157         data_file_number=data_args.data_num_limit,
158         data_proc_num=data_args.data_proc_num,
159         use_streaming=data_args.use_streaming)
160     train_dataset = train_dataset.remove_columns(
161         ['uniqueKey', 'title', 'titleUkey', 'dataType', 'id', 'content'])
162
163     data_collator = DataCollatorForLanguageModeling(
164         tokenizer=tokenizer, mlm=False, pad_to_multiple_of=8
165     )
166     if not data_args.use_streaming:
167         training_args.max_steps = -1
168
169
170
171     trainer = Trainer(model=model,
172                       tokenizer=tokenizer,
173                       args=training_args,
174                       train_dataset=train_dataset,
175                       eval_dataset=None,
176                       data_collator=data_collator,
177

```

```

178     trainer.train()
179     trainer.save_state()
180     trainer.save_model(output_dir=training_args.output_dir)
181
182
183     if __name__ == "__main__":
184         logging.basicConfig(
185             format="%(asctime)s %(levelname)s [%(name)s] %(message)s", level=
logging.INFO, datefmt="%Y-%m-%d %H:%M:%S"
186         )

```

Python |

```

1  python train.py \
2      --model_name_or_path /data/yh/bigscience_bloom-1b1 \
3      --data_path /data/yh/WuDaoCorpus2.0_base_200G \
4      --data_num_limit 60 \
5      --data_proc_num 40 \
6      --bf16 False \
7      --output_dir output_dir \
8      --num_train_epochs 3 \
9      --per_device_train_batch_size 2 \
10     --per_device_eval_batch_size 1 \
11     --gradient_accumulation_steps 8 \
12     --evaluation_strategy "no" \
13     --save_strategy "steps" \
14     --save_steps 1000 \
15     --save_total_limit 10 \
16     --learning_rate 2e-5 \
17     --weight_decay 0. \
18     --warmup_ratio 0.03 \
19     --logging_steps 10 \
20     --tf32 False \
21     --model_max_length 1024 \
22     --use_streaming True\
23     --max_steps 10000

```

sft 代码


```

1  from peft.tuners.lora import LoraLayer
2  import copy
3  import logging
4  import logging
5  import os
6  import torch
7  import transformers
8  from dataclasses import dataclass, field
9  from datasets import load_dataset
10 from functools import partial
11 from torch.utils.data import Dataset
12 from tqdm import tqdm
13 from transformers import DataCollatorForSeq2Seq, Trainer
14 from typing import Dict, Optional, Sequence, List
15
16 logger = logging.getLogger(__name__)
17
18
19 @dataclass
20 class ModelArguments:
21     model_name_or_path: Optional[str] = field(default="facebook/opt-125
m")
22     use_lora: Optional[bool] = field(default=False)
23
24
25 @dataclass
26 class DataArguments:
27     data_path: str = field(default=None, metadata={
28         "help": "Path to the training data."})
29     source_length: int = field(default=512)
30     target_length: int = field(default=512)
31
32
33 @dataclass
34 class TrainingArguments(transformers.TrainingArguments):
35     cache_dir: Optional[str] = field(default=None)
36     optim: str = field(default="adamw_torch")
37     model_max_length: int = field(
38         default=512,
39         metadata={
40             "help": "Maximum sequence length. Sequences will be right pad
ded (and possibly truncated)."},
41     )
42     use_deepspeed: bool = field(default=False)

```

```

43
44
45 def get_all_datapath(dir_name: str) -> List[str]:
46     all_file_list = []
47     # all_file_size = []
48
49     for (root, dir, file_name) in os.walk(dir_name):
50         for temp_file in file_name:
51             standard_path = f"{root}/{temp_file}"
52
53             all_file_list.append(standard_path)
54
55     return all_file_list
56
57
58 def load_dataset_from_path(data_path: Optional[str] = None,
59                             cache_dir: Optional[str] = "cache_data") -> Da
60 taset:
61     all_file_list = get_all_datapath(data_path)
62     data_files = {'train': all_file_list}
63     extension = all_file_list[0].split(".")[1]
64
65     logger.info("load files %d number", len(all_file_list))
66
67     raw_datasets = load_dataset(
68         extension,
69         data_files=data_files,
70         cache_dir=cache_dir,
71     )['train']
72     return raw_datasets
73
74 IGNORE_INDEX = -100
75 PROMPT_DICT = {
76     "prompt_input": (
77         "Below is an instruction that describes a task, paired with an in
78 put that provides further context. "
79         "Write a response that appropriately completes the request.\n\n"
80         "### Instruction:\n{instruction}\n\n### Input:\n{input}\n\n### Re
81 sponse:"
82     ),
83     "prompt_no_input": (
84         "Below is an instruction that describes a task. "
85         "Write a response that appropriately completes the request.\n\n"
86         "### Instruction:\n{instruction}\n\n### Response:"
87     ),
88 }

```

```

87
88
89 def _tokenize_fn(strings: Sequence[str], tokenizer: transformers.PreTrain
edTokenizer) -> Dict:
90     """Tokenize a list of strings."""
91     tokenized_list = [
92         tokenizer(
93             text,
94             return_tensors="pt",
95             padding="longest",
96             max_length=tokenizer.model_max_length,
97             truncation=True,
98         )
99         for text in strings
100     ]
101     input_ids = labels = [tokenized.input_ids[0]
102                           for tokenized in tokenized_list]
103     ne_pad_token_id = IGNORE_INDEX if tokenizer.pad_token_id is None els
e tokenizer.pad_token_id
104     input_ids_lens = labels_lens = [
105         tokenized.input_ids.ne(ne_pad_token_id).sum().item() for tokenize
d in tokenized_list
106     ]
107     return dict(
108         input_ids=input_ids,
109         labels=labels,
110         input_ids_lens=input_ids_lens,
111         labels_lens=labels_lens,
112     )
113
114
115 def preprocess(
116     sources: Sequence[str],
117     targets: Sequence[str],
118     tokenizer: transformers.PreTrainedTokenizer,
119 ) -> Dict:
120     """Preprocess the data by tokenizing."""
121     examples = [s + t for s, t in zip(sources, targets)]
122     examples_tokenized, sources_tokenized = [_tokenize_fn(
123         strings, tokenizer) for strings in (examples, sources)]
124     input_ids = examples_tokenized["input_ids"]
125     labels = copy.deepcopy(input_ids)
126     for label, source_len in zip(labels, sources_tokenized["input_ids_len
s"]):
127         label[:source_len] = IGNORE_INDEX
128     return dict(input_ids=input_ids, labels=labels)
129

```

```

130
131 def make_train_dataset(tokenizer: transformers.PreTrainedTokenizer, data_
path: str, data_args: DataArguments) -> Dataset:
132     logging.warning("Loading data...")
133
134     dataset = load_dataset_from_path(
135         data_path=data_path,
136     )
137     logging.warning("Formatting inputs...")
138     prompt_input, prompt_no_input = PROMPT_DICT["prompt_input"], PROMPT_D
ICT["prompt_no_input"]
139
140     def generate_sources_targets(examples: Dict, tokenizer: transformers.
PreTrainedTokenizer):
141         ins_data = examples['instruction']
142         if 'input' not in examples.keys():
143             input_data = [""] * len(ins_data)
144         else:
145             input_data = examples['input']
146         output = examples['output']
147
148         len_ = len(ins_data)
149
150         # sources = []
151         # targets = []
152
153         # for i in range(len_):
154         #     s_t = prompt_input.format_map({'instruction':ins_data[i],
155         #                                     'input':input_data[i]}) if i
nput_data[i] != "" else prompt_input.format_map({'instruction':ins_data
[i]})
156         #     sources.append(s_t)
157
158         sources = [prompt_input.format_map({'instruction': ins_data[i],
'input': input_data[i]}) if input_data[
159             i] != "" else prompt_no_input.format_map(
160                 {'instruction': ins_data[i]})
161                 for i in range(len_)]
162         sources = [i[:data_args.source_length] for i in sources]
163         targets = [
164             f"{example[:data_args.target_length-1]}{tokenizer.eos_toke
n}" for example in output]
165
166         # sources = [prompt_input.format_map(example) if example.get("inp
ut", "") != "" else prompt_no_input.format_map(example)
167         #             for example in examples]
168         # targets = [

```

```

169         #         f"{example['output']}{tokenizer.eos_token}" for example in
examples]
170
171         input_output = preprocess(
172             sources=sources, targets=targets, tokenizer=tokenizer)
173         examples['input_ids'] = input_output['input_ids']
174         examples['labels'] = input_output['labels']
175         return examples
176
177     generate_sources_targets_p = partial(
178         generate_sources_targets, tokenizer=tokenizer)
179
180     dataset = dataset.map(
181         function=generate_sources_targets_p,
182         batched=True,
183         desc="Running tokenizer on train dataset",
184         num_proc=20
185     ).shuffle()
186     return dataset
187
188
189
190 def load_model_and_tokenizer(model_args: ModelArguments, training_args: T
rainingArguments, data_args: DataArguments) -> tuple:
191
192     if training_args.use_deepspeed:
193
194         model = transformers.AutoModelForCausalLM.from_pretrained(
195             model_args.model_name_or_path,
196             cache_dir=training_args.cache_dir,
197             torch_dtype='auto',
198             # if model_args.model_name_or_path.find("falcon") != -1 else
False
199             trust_remote_code=True
200
201         )
202     else:
203         model = transformers.AutoModelForCausalLM.from_pretrained(
204             model_args.model_name_or_path,
205             cache_dir=training_args.cache_dir,
206             device_map='auto',
207             torch_dtype='auto',
208             # if model_args.model_name_or_path.find("falcon") != -1 else
False
209             trust_remote_code=True
210
211         )

```

```

212
213     if model_args.use_lora:
214
215         logging.warning("Loading model to Lora")
216
217         from peft import LoraConfig, get_peft_model
218         LORA_R = 32
219         # LORA_ALPHA = 16
220         LORA_DROPOUT = 0.05
221         TARGET_MODULES = [
222             "o_proj", "gate_proj", "down_proj", "up_proj"
223         ]
224
225         config = LoraConfig(
226             r=LORA_R,
227             # lora_alpha=LORA_ALPHA,
228             target_modules=TARGET_MODULES,
229             lora_dropout=LORA_DROPOUT,
230             bias="none",
231             task_type="CAUSAL_LM",
232         )
233         # model = model.to(torch.bfloat16)
234         model = get_peft_model(model, config)
235         # peft_module_casting_to_bf16(model)
236         model.print_trainable_parameters()
237
238         # model.is_parallelizable = True
239         # model.model_parallel = True
240         # torch.cuda.empty_cache()
241
242         tokenizer = transformers.AutoTokenizer.from_pretrained(
243             model_args.model_name_or_path, trust_remote_code=True)
244
245         return model, tokenizer
246
247
248     def train():
249         parser = transformers.HfArgumentParser(
250             (ModelArguments, DataArguments, TrainingArguments))
251         model_args, data_args, training_args = parser.parse_args_into_dataclasses()
252
253         model, tokenizer = load_model_and_tokenizer(
254             model_args, training_args, data_args)
255
256         with training_args.main_process_first(desc="loading and tokenization"):

```

```

257
258     train_dataset = make_train_dataset(
259         tokenizer=tokenizer, data_path=data_args.data_path, data_args
260         =data_args)
261     data_collator = DataCollatorForSeq2Seq(tokenizer=tokenizer, model=model,
262                                           label_pad_token_id=IGNORE_INDEX,
263                                           X=X)
264
265     trainer = Trainer(model=model,
266                       tokenizer=tokenizer,
267                       args=training_args,
268                       train_dataset=train_dataset,
269                       eval_dataset=None,
270                       data_collator=data_collator)
271     trainer.train()
272     trainer.save_state()
273     trainer.save_model(output_dir=training_args.output_dir)
274
275
276 if __name__ == "__main__":
277     logging.basicConfig(
278         format="%(asctime)s %(levelname)s [%(name)s] %(message)s", level=
279         logging.INFO, datefmt="%Y-%m-%d %H:%M:%S"

```

```
1  # --nnodes 1 --nproc_per_node 4 --master_port 25641
2
3  deepspeed --include localhost:0,1,2,3 train_sft.py \
4      --deepspeed ds_zero2_no_offload.json \
5      --model_name_or_path internlm-7b \
6      --use_lora true \
7      --use_deepspeed true \
8      --data_path hz_sft_datav2 \
9      --bf16 true \
10     --fp16 false \
11     --output_dir output_refusev2 \
12     --num_train_epochs 5 \
13     --per_device_train_batch_size 3 \
14     --per_device_eval_batch_size 1 \
15     --gradient_accumulation_steps 8 \
16     --evaluation_strategy "no" \
17     --save_strategy "epoch" \
18     --save_total_limit 3 \
19     --learning_rate 4e-4 \
20     --logging_steps 10 \
21     --tf32 False \
22     --model_max_length 2048
23
24  # --save_steps 1000 \
```

chat template


```

1  from transformers import AutoTokenizer
2
3  tokenizer = AutoTokenizer.from_pretrained("beomi/Llama-3-Open-Ko-8B")
4
5  messages = [
6      {"role": "user", "content": "Hi there!"},
7      {"role": "assistant", "content": "Nice to meet you!"},
8      {"role": "user", "content": "Can I ask a question?"}
9  ]
10
11 print(tokenizer.apply_chat_template(messages, tokenize=False))
12 =====
13 <|begin_of_text|><|start_header_id|>user<|end_header_id|>
14
15 Hi there!<|eot_id|><|start_header_id|>assistant<|end_header_id|>
16
17 Nice to meet you!<|eot_id|><|start_header_id|>user<|end_header_id|>
18
19 Can I ask a question?<|eot_id|>

```

```

1  # 添加模型开始回复
2  print(tokenizer.apply_chat_template(messages, tokenize=False, add_generati
on_prompt=True))
3  =====
4  =====
5
6  <|begin_of_text|><|start_header_id|>user<|end_header_id|>
7
8  Hi there!<|eot_id|><|start_header_id|>assistant<|end_header_id|>
9
10 Nice to meet you!<|eot_id|><|start_header_id|>user<|end_header_id|>
11
12 Can I ask a question?<|eot_id|><|start_header_id|>assistant<|end_header_id
|>

```

sft 数据

数据任务

1.OpenAI 官网列出了 ChatGPT 擅长的所有任务项，诸如翻译、emoji 聊天.....之类的。

2.NER、机器阅读理解、意图识别等传统的 NLP 任务。

3.参考业务需求。

数据量配比：难 task_type 数据多点，简单 task_type 数据少点

数据形式

1.prompt 表达方式需要多样性，不要千篇一律的“把中文句子 A 翻译成英文”，也要适当有一些“我在英国旅游，我现在需要向路人问路，我想表达 A 的意思，该怎么说”，“我是一个英文老师，我需要向我的学生讲解句子 A 用英文怎么写，请你用最正宗的表达方式帮我完成。”这么做的目的是防止模型只认识 prompt 中的几个关键 token，进而导致训练过拟合或者泛化性变差

2.prompt 长度均衡，既要有短数据，也要有长数据，避免模型的 attention 退化到无法聚焦长 prompt

3.answer 长度均衡，不能让模型没输出几个 token 就停止，适当的有一些语料让它学会输出尽量长的 answer，否则模型会很难 follow “不少于2000字”这种指令

4.多轮聊天的切换 topic 能力，有的数据当前 query 是和 session 有关系的，有的数据则是当前 query 和 session 毫无关系，要让模型自己学会判断 query 是否和 session 有关。类似的

数据还要有 system 是否生效，有些数据 system 是个摆设，有些数据的 answer 则和 system 直接相关

5.answer 分布的多样性，例如总共一万条训练数据，一千条数据的 answer 都说同一句话，太单一的话会严重让模型过拟合

数据生产

prompt: <https://github.com/yizhongw/self-instruct>

answer: GPT/Claude, 部署选择 qwen 或者 deepseek-moe

筛选用户日志，例如有用/没用可以辅助 DPO/RLHF

sft 参数

注意的参数：

```
epoch, gradient_accumulation_steps, global_batch_size, learning_rate, lr_scheduler_type, dropout
```

影响速度的参数

```
zero_stage, max_seq_len, offload, gradient_checkpointing, seq_parallel_size
```

其他

`weight_decay, per_device_train_batch_size, num_warmup_steps`

sft 技巧

- 1.小模型大学习率，大模型小学习率，epoch 基本 1~3
- 2.模型的初始 loss，7B / 13B 可能在 2 左右，数据难了也有可能到 3，72B 则大多在 1 ~ 2 之间这个水平状态；最终 loss 则大概在 0.5 左右

拟合性

欠拟合

判断模型是真的连训练数据都没学会，还是学会了训练数据但无法进行泛化

测试方法：直接让模型回答训练集

1.没学会训练集

解决方法：多训 1 epoch，调整学习率（观察 loss 曲线和梯度，如果 loss 下降缓慢就增大学习率跳出局部最小值，如果 loss 比较震荡学习很困难就减小学习率提高训练稳定性）

2.学会了训练集

判断任务难度与模型 size 是否相符，判断 pretrain 是否学过相关知识（续写相关内容），抽样 answer 查看质量，重写 prompt

过拟合

暴露模型对什么任务类型过拟合，调整数据配比

评估

原则：Helpfulness、Honesty、Harmlessness

方法：GPT（prompt 参考Alignbench）、人工

Lora

原理

将 ΔW 分解为 $A*B$

初始化

一个初始化为 0，一个使用某种随机初始化（保证训练的开始旁路矩阵依然是0矩阵）

超参

$$\mathbf{h} = (\mathbf{W}_0 + \frac{\alpha}{r} \Delta \mathbf{W}) \mathbf{x}$$

r: 秩 α : 缩放因子

系数 $\frac{\alpha}{r}$ 越大, LoRA微调权重的影响就越大, 在下游任务上越容易过拟合

系数 $\frac{\alpha}{r}$ 越小, LoRA微调权重的影响就越小 (微调的效果不明显, 原始模型参数受到的影响也较少)

注:

1. 简单任务所需的 r 不大, 任务越难/多任务混合的情况, 需要更大的 r
2. 越强的基座, 所需的 r 应该更小
3. 数据规模越大, 需要更大的 r