0922 xtuner微调模型 作业

本节课老师的讲义地址

https://docs.qq.com/doc/p/c82bd5576c813cae2cb8538303a4a95bd52fb324

Xtuner的使用

这是官方介绍。
https://github.com/InternLM/xtuner/blob/main/README_zh-CN.md

它是一个大语言模型的微调工具。提供了很多开箱即用的配置文件,同时它也是一个命令行工具。

1. 查看配置文件

通过 `xtuner list-cfg` 可以查看所有配置文件

```
root@f1d398e9664c4a169ebd0aa529352eea-task0-0:/# xtuner list-cfg
==========================CONFIGS===========================
baichuan2_13b_base_qlora_alpaca_e3
baichuan2_13b_base_qlora_alpaca_enzh_e3
baichuan2_13b_base_qlora_alpaca_enzh_oasst1_e3
baichuan2_13b_base_qlora_alpaca_zh_e3
baichuan2_13b_base_qlora_arxiv_gentitle_e3
baichuan2_13b_base_qlora_code_alpaca_e3
baichuan2_13b_base_qlora_colorist_e5
baichuan2_13b_base_qlora_lawyer_e3
baichuan2_13b_base_qlora_oasst1_512_e3
baichuan2_13b_base_qlora_oasst1_e3
baichuan2_13b_base_qlora_open_platypus_e3
baichuan2_13b_base_qlora_sql_e3
baichuan2_13b_chat_qlora_alpaca_e3
baichuan2_13b_chat_qlora_alpaca_enzh_e3
baichuan2_13b_chat_qlora_alpaca_enzh_oasst1_e3
baichuan2_13b_chat_qlora_alpaca_zh_e3
baichuan2_13b_chat_qlora_code_alpaca_e3
baichuan2_13b_chat_qlora_lawyer_e3
baichuan2_13b_chat_qlora_oasst1_512_e3
baichuan2_13b_chat_qlora_oasst1_e3
baichuan2_13b_chat_qlora_open_platypus_e3
```

以第一个为例：
`baichuan2_13b_base_qlora_alpaca_e3`

baichuan2_13b_base 指的是大模型的名称，其中 13b指的是参数数量是 130亿，base指的是大模型版本。

qlora指的是训练方法

alpaca指的是数据集。

e3 指的是 训练主要配置epoch数

2.复制配置文件并按照需求去更改

`xtuner copy-cfg baichuan2_13b_base_qlora_alpaca_e3 /code`

这个文件的内容如下：

```
# Copyright (c) OpenMMLab. All rights reserved.
import torch
from datasets import load_dataset
from mmengine.dataset import DefaultSampler
from mmengine.hooks import (CheckpointHook, DistSamplerSeedHook, IterTimerHook,
                            LoggerHook, ParamSchedulerHook)
from mmengine.optim import AmpOptimWrapper, CosineAnnealingLR, LinearLR
from peft import LoraConfig
from torch.optim import AdamW
from transformers import (AutoModelForCausalLM, AutoTokenizer,
                          BitsAndBytesConfig)

from xtuner.dataset import process_hf_dataset
from xtuner.dataset.collate_fns import default_collate_fn
from xtuner.dataset.map_fns import alpaca_map_fn, template_map_fn_factory
from xtuner.engine.hooks import (DatasetInfoHook, EvaluateChatHook,
                                 VarlenAttnArgsToMessageHubHook)
from xtuner.engine.runner import TrainLoop
from xtuner.model import SupervisedFinetune
from xtuner.utils import PROMPT_TEMPLATE, SYSTEM_TEMPLATE

#######################################################################
#                          PART 1  Settings                           #
#######################################################################
# Model
pretrained_model_name_or_path = 'baichuan-inc/Baichuan2-13B-Base'
use_varlen_attn = False

# Data
alpaca_en_path = 'tatsu-lab/alpaca'
prompt_template = PROMPT_TEMPLATE.default
max_length = 2048
pack_to_max_length = True
```

```
# Scheduler & Optimizer
batch_size = 1  # per_device
accumulative_counts = 16
dataloader_num_workers = 0
max_epochs = 3
optim_type = AdamW
lr = 2e-4
betas = (0.9, 0.999)
weight_decay = 0
max_norm = 1  # grad clip
warmup_ratio = 0.03

# Save
save_steps = 500
save_total_limit = 2  # Maximum checkpoints to keep (-1 means unlimited)

# Evaluate the generation performance during the training
evaluation_freq = 500
SYSTEM = SYSTEM_TEMPLATE.alpaca
evaluation_inputs = [
    '请给我介绍五个上海的景点', 'Please tell me five scenic spots in Shanghai'
]

#######################################################################
#                      PART 2  Model & Tokenizer                      #
#######################################################################
tokenizer = dict(
    type=AutoTokenizer.from_pretrained,
    pretrained_model_name_or_path=pretrained_model_name_or_path,
    trust_remote_code=True,
    padding_side='right')

model = dict(
    type=SupervisedFinetune,
    use_varlen_attn=use_varlen_attn,
    llm=dict(
        type=AutoModelForCausalLM.from_pretrained,
        pretrained_model_name_or_path=pretrained_model_name_or_path,
        trust_remote_code=True,
        torch_dtype=torch.float16,
        quantization_config=dict(
            type=BitsAndBytesConfig,
            load_in_4bit=True,
            load_in_8bit=False,
            llm_int8_threshold=6.0,
            llm_int8_has_fp16_weight=False,
            bnb_4bit_compute_dtype=torch.float16,
            bnb_4bit_use_double_quant=True,
            bnb_4bit_quant_type='nf4')),
    lora=dict(
        type=LoraConfig,
        r=64,
```

```
        lora_alpha=16,
        lora_dropout=0.1,
        bias='none',
        task_type='CAUSAL_LM'))

#######################################################################
#                      PART 3  Dataset & Dataloader                   #
#######################################################################
alpaca_en = dict(
    type=process_hf_dataset,
    dataset=dict(type=load_dataset, path=alpaca_en_path),
    tokenizer=tokenizer,
    max_length=max_length,
    dataset_map_fn=alpaca_map_fn,
    template_map_fn=dict(
        type=template_map_fn_factory, template=prompt_template),
    remove_unused_columns=True,
    shuffle_before_pack=True,
    pack_to_max_length=pack_to_max_length,
    use_varlen_attn=use_varlen_attn)

train_dataloader = dict(
    batch_size=batch_size,
    num_workers=dataloader_num_workers,
    dataset=alpaca_en,
    sampler=dict(type=DefaultSampler, shuffle=True),
    collate_fn=dict(type=default_collate_fn, use_varlen_attn=use_varlen_attn))

#######################################################################
#                    PART 4  Scheduler & Optimizer                    #
#######################################################################
# optimizer
optim_wrapper = dict(
    type=AmpOptimWrapper,
    optimizer=dict(
        type=optim_type, lr=lr, betas=betas, weight_decay=weight_decay),
    clip_grad=dict(max_norm=max_norm, error_if_nonfinite=False),
    accumulative_counts=accumulative_counts,
    loss_scale='dynamic',
    dtype='float16')

# learning policy
# More information: https://github.com/open-
mmlab/mmengine/blob/main/docs/en/tutorials/param_scheduler.md  # noqa: E501
param_scheduler = [
    dict(
        type=LinearLR,
        start_factor=1e-5,
        by_epoch=True,
        begin=0,
        end=warmup_ratio * max_epochs,
        convert_to_iter_based=True),
```

```
    dict(
        type=CosineAnnealingLR,
        eta_min=0.0,
        by_epoch=True,
        begin=warmup_ratio * max_epochs,
        end=max_epochs,
        convert_to_iter_based=True)
]

# train, val, test setting
train_cfg = dict(type=TrainLoop, max_epochs=max_epochs)

#######################################################################
#                           PART 5  Runtime                           #
#######################################################################
# Log the dialogue periodically during the training process, optional
custom_hooks = [
    dict(type=DatasetInfoHook, tokenizer=tokenizer),
    dict(
        type=EvaluateChatHook,
        tokenizer=tokenizer,
        every_n_iters=evaluation_freq,
        evaluation_inputs=evaluation_inputs,
        system=SYSTEM,
        prompt_template=prompt_template)
]

if use_varlen_attn:
    custom_hooks += [dict(type=VarlenAttnArgsToMessageHubHook)]

# configure default hooks
default_hooks = dict(
    # record the time of every iteration.
    timer=dict(type=IterTimerHook),
    # print log every 10 iterations.
    logger=dict(type=LoggerHook, log_metric_by_epoch=False, interval=10),
    # enable the parameter scheduler.
    param_scheduler=dict(type=ParamSchedulerHook),
    # save checkpoint per `save_steps`.
    checkpoint=dict(
        type=CheckpointHook,
        by_epoch=False,
        interval=save_steps,
        max_keep_ckpts=save_total_limit),
    # set sampler seed in distributed evrionment.
    sampler_seed=dict(type=DistSamplerSeedHook),
)

# configure environment
env_cfg = dict(
    # whether to enable cudnn benchmark
    cudnn_benchmark=False,
```

```
    # set multi process parameters
    mp_cfg=dict(mp_start_method='fork', opencv_num_threads=0),
    # set distributed parameters
    dist_cfg=dict(backend='nccl'),
)

# set visualizer
visualizer = None

# set log level
log_level = 'INFO'

# load from which checkpoint
load_from = None

# whether to resume training from the loaded checkpoint
resume = False

# Defaults to use random seed and disable `deterministic`
randomness = dict(seed=None, deterministic=False)

# set log processor
log_processor = dict(by_epoch=False)
```

其中大部分配置我们无须关心，下面这些我们可以稍作修改：

参数	描述
pretrained_model_name_or_path	我们使用的基础模型，填写本地文件的路径即可
alpaca_en_path	数据集路径
max_length	数据最大长度，越长会越消耗显存
batch_size	每个GPU上一个batch数据的大小，越大越消耗显存
accumulative_counts	梯度累积的次数，总的batch为batch_size * accumulative_counts
max_epochs	训练最大的epoch，整体数据迭代次数
save_steps	训练到多少步的时候保存模型，需要根据总步数进行调整
save_total_limit	总共保存的checkpoint数量，太多会导致磁盘满了
evaluation_freq	训练过程中进行到多少步的时候会做一次评估
SYSTEM	评估时需要用的系统提示词，xtuner内置了一些数据集的系统提示词，比如SYSTEM_TEMPLATE.alpaca

evaluation_inputs	用户的输入
quantization_config	量化配置，需要查看自己的硬件是否支持量化参数

其中
`pretrained_model_name_or_path` 是我们要微调的基础模型路径，必须修改。
`alpaca_en_path` 是我们用来微调模型的数据集的路径，也必须修改。
`quantization_config` 汇视威平台暂时不支持量化配置，所以要将上面的quantization_config配置注释掉

3. 开始微调

微调分为单机微调和多机微调。

单机，就是用一台机器上的显卡来对模型进行训练。多机微调，则是需要多台机器使用IB网卡进行通信，最终在master机器上生成训练后的产物。

接下来我直接从作业的角度去试试微调。

网络环境准备

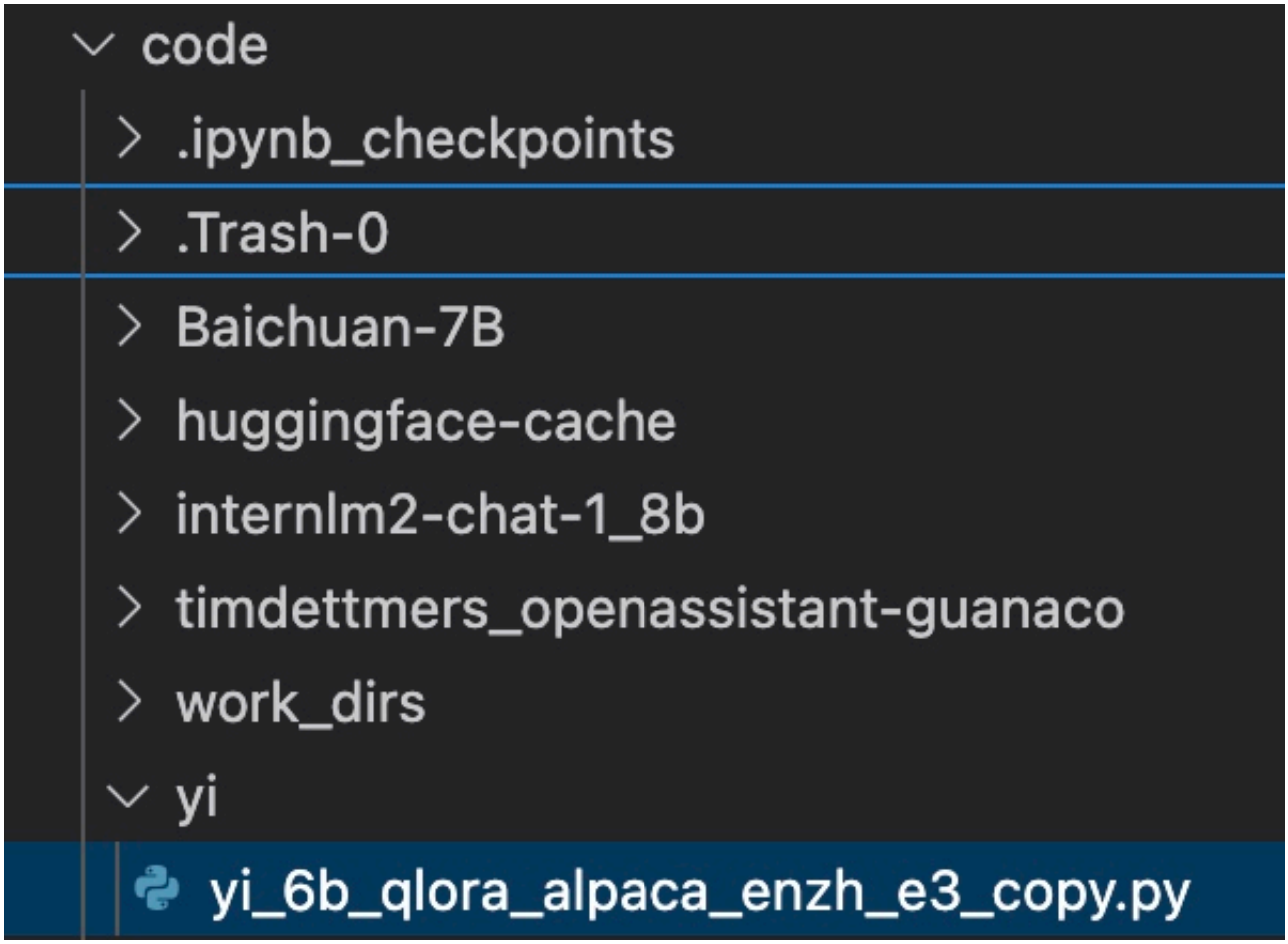
```
export http_proxy=http://10.10.9.50:3000
export https_proxy=http://10.10.9.50:3000
export no_proxy=localhost,127.0.0.1
export HF_HOME=/code/huggingface-cache
export HF_ENDPOINT=https://hf-mirror.com
```

单机微调作业

从 xtuner的配置文件中取出这个复制到本地。

`yi_6b_qlora_alpaca_enzh_e3`

`xtuner copy-cfg yi_6b_qlora_alpaca_enzh_e3 /code/yi` 来将微调文件复制到本地。



下一步，下载需要微调的模型。

```
huggingface-cli download 01-ai/Yi-6B --revision main --local-dir-use-symlinks False --local-dir /code/01-ai/Yi-6B
```

再下一步，下载数据集：

从上面的配置文件中发现了两个数据集，所以我需要下载两个数据集，如下：`silk-road/alpaca-data-gpt4-chinese` 和 `tatsu-lab/alpaca`

所以我要分别执行两条指令：

```
huggingface-cli download silk-road/alpaca-data-gpt4-chinese --repo-type dataset --revision main --local-dir-use-symlinks False --local-dir /code/silk-road/alpaca-data-gpt4-chinese
```

和

```
huggingface-cli download tatsu-lab/alpaca --repo-type dataset --revision main --local-dir-use-symlinks False --local-dir /code/tatsu-lab/alpaca
```

等模型和数据集都下载完成之后，他们应该都会存在于我的code目录下。

现在可以开始微调了。

如果想单机单卡，就只需要使用：

`xtuner train code/yi/yi_6b_qlora_alpaca_enzh_e3_copy.py`

如果是要单机多卡，那就在前面加一个 `NPROC_PER_NODE=4`：

```
NPROC_PER_NODE=4 xtuner train code/yi/yi_6b_qlora_alpaca_enzh_e3_copy.py
```

这个4表示你要使用的显卡数量。

另外，使用deepspeed参数可以显著降低显卡消耗，它是一种优化算法 `--deepspeed deepspeed_zero3`：

```
NPROC_PER_NODE=4 xtuner train code/yi/yi_6b_qlora_alpaca_enzh_e3_copy.py --
deepspeed deepspeed_zero3
```

deepspeed的参数的值，可以从 `xtuner list-cfg -p deepspeed` 中去查看。

```
root@f1d398e9664c4a169ebd0aa529352eea-task0-0:/# xtuner list-cfg -p deepspeed
==========================CONFIGS===========================
PATTERN: deepspeed
------------------------------
deepspeed_zero1
deepspeed_zero2
deepspeed_zero2_offload
deepspeed_zero3
deepspeed_zero3_offload
```

多机微调

首先要启用ib网卡：

```
export NCCL_DEBUG=INFO
export NCCL_IB_DISABLE=0
export NCCL_IB_HCA==mlx5_0:1,mlx5_1:1,mlx5_2:1,mlx5_3:1
export NCCL_SOCKET_IFNAME=eth0
export GLOO_SOCKET_IFNAME=eth0
export HF_HOME=/code/huggingface-cache/
```

如果是用4台机器进行训练：

```
NNODES=4 NPROC_PER_NODE=4 PORT=12345 ADDR=10.244.199.211 NODE_RANK=0 xtuner
traincode/yi/yi_6b_qlora_alpaca_enzh_e3_copy.py --work-dir /userhome/xtuner-
workdir --deepspeed deepspeed_zero3
```

两个关键因素 ADDR ， 这是master机器的ip地址，NODE_RANK，这是机器序号，一般master机器为0，其他机器就递增就行。

剩余3台机器，就是：

```
NNODES=4 NPROC_PER_NODE=4 PORT=12345 ADDR=10.244.199.211 NODE_RANK=1 xtuner
train code/yi/yi_6b_qlora_alpaca_enzh_e3_copy.py --work-dir /userhome/xtuner-
workdir --deepspeed deepspeed_zero3
```

```
NNODES=4 NPROC_PER_NODE=4 PORT=12345 ADDR=10.244.199.211 NODE_RANK=2 xtuner
train code/yi/yi_6b_qlora_alpaca_enzh_e3_copy.py --work-dir /userhome/xtuner-
workdir --deepspeed deepspeed_zero3
```

```
NNODES=4 NPROC_PER_NODE=4 PORT=12345 ADDR=10.244.199.211 NODE_RANK=3 xtuner
traincode/yi/yi_6b_qlora_alpaca_enzh_e3_copy.py --work-dir /userhome/xtuner-
workdir --deepspeed deepspeed_zero3
```

作业2 自备一个数据集，并且用它来微调大模型

todo 难度较大，上了后面的课再继续。