InternLM 模型微调

一. xTuner

XTuner 是由上海人工智能实验室开发的**大模型训练、微调工具箱**，通过 XTuner，仅需 24GB 显存，就可以微调 InternLM-20B 系列模型。

目前，XTuner 已支持 InternLM-20B 模型的 **LoRA、QLoRA、全参数微调**，集成 **DeepSpeed ZeRO** 训练优化技巧，并支持诸如 **Alpaca、OpenAssistant、MSAgent** 等热门开源数据集，用户可以**“开箱即用”**!
XTuner GitHub 链接：

https://github.com/InternLM/xtunergithub.com/InternLM/xtuner

InternLM-20B GitHub 链接：

[https://github.com/InternLM/InternLMgithub.com/InternLM/InternLM]

XTuner 大语言模型微调优化策略

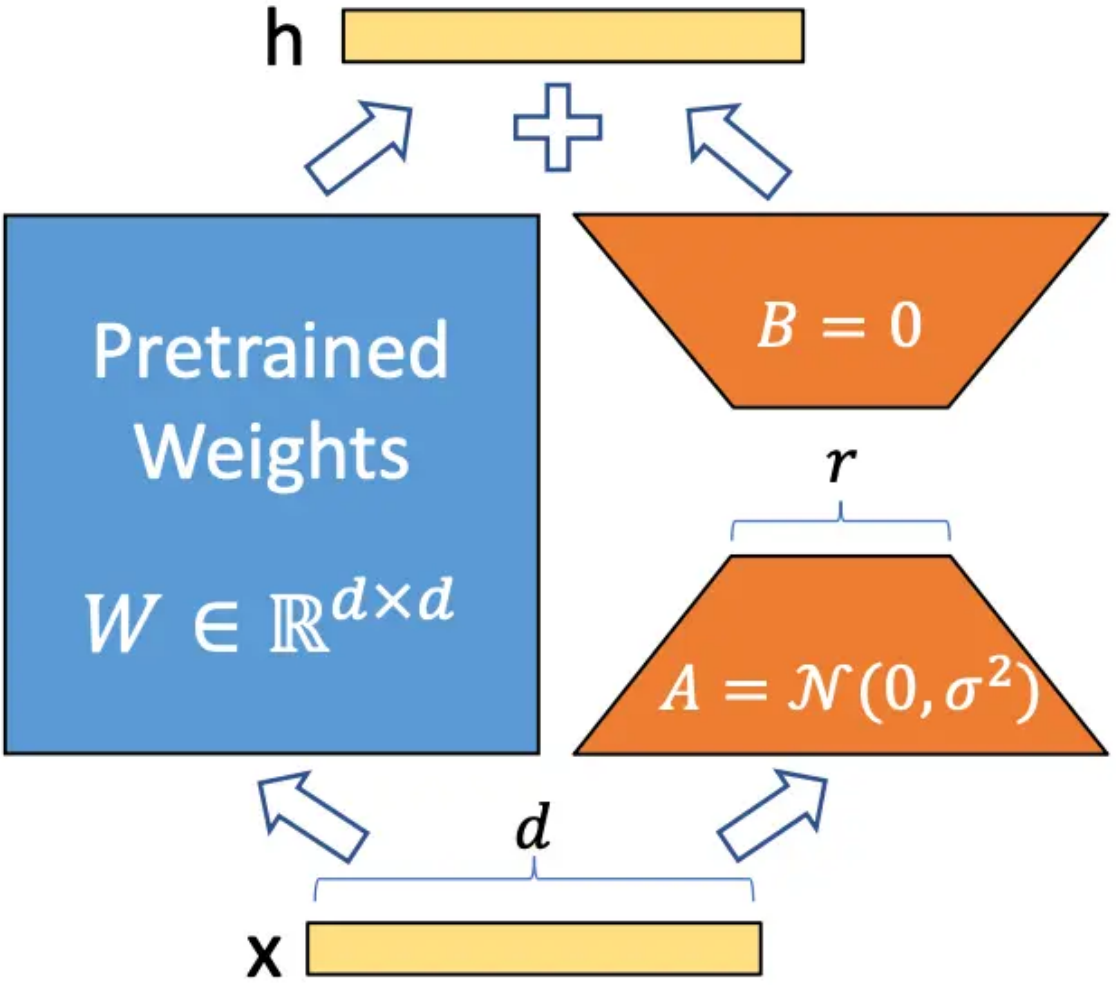
大语言模型微调时显存占用主要包括三部分：权重参数、权重梯度、优化器状态。其中，权重参数一般为静态占用，并不随训练变化，而权重梯度、优化器状态则是训练时占用。LoRA 用于优化**权重梯度**、**优化器状态**的显存开销，而引入量化的 QLoRA 可以在此基础上进一步优化**权重参数**的显存开销。

DeepSpeed ZeRO

DeepSpeed 的 ZeRO 优化，通过将训练过程中的权重参数、权重梯度和优化器状态**切片保存**，每个 GPU 仅保存部分内容，以此在多 GPU 训练时大幅度降低显存开销。

DeepSpeed 训练时使用 FP16/BF16 的类型转换，相较于 PyTorch 的自动混合精度训练（AMP），在单 GPU 上也能有效降低训练时间，节省显存。

XTuner 目前已经集成了 DeepSpeed 的各项 ZeRO 优化策略，以 InternLM-20B QLoRA 在 Alpaca 数据集微调为例，引入 ZeRO 策略后，可以**加速训练 25%，降低显存开销约 5 GB（29GB --> 24GB）。**



LoRA 旁路示意图

结合 LoRA 技巧（如上图），冻结大语言模型中的 Linear 层并新增一条包含两个连续“低维度” 的可训练 Linear 旁路（即 Adapter），可以有效降低**权重梯度**和**优化器状态**的显存占用。进一步，结合 QLoRA 技巧，将权重参数的存储由 FP16/BF16 转换为 INT4，可以有效降低**权重参数**的显存占用。

Flash Attention

Flash Attention 将大语言模型中的 Attention 计算并行化，可以有效降低 Attention Score NxN 的显存占用（N 通常表示 token 个数，与文本长度强相关）。

XTuner 目前已经默认开启 InternLM-20B 的 Flash Attention 训练，在训练时对原始朴素 Attention 模块进行自动替换。

大型语言模型的监督微调 （SFT） 旨在通过监督微调来提高预训练模型在特定任务上的性能。为了支持尽可能多的下游任务，XTuner 支持三种数据集格式：增量预训练、单轮对话和多轮对话。

- 增量预训练数据集用于增强模型在特定领域或任务中的能力。
- 指令调优阶段常使用单轮和多轮对话数据集，以增强模型对特定指令的响应能力。

二.数据集构建

单轮对话数据集格式

单轮对话数据集通常由单个指令（或问题）及其相应的 GroundTruth 答案组成。由于只有答案部分应该用于梯度反向传播，因此数据集的“system”和“input”字段是输入指令，而“output”字段是对应的答案。单轮对话数据集的格式如下图所示：

```
[{
    "conversation":[
        {
            "system": "You are an AI asssistant."
            "input": "Give three tips for staying healthy.",
            "output": "1.Eat a balanced diet. 2. Exercise regularly. 3. Get
enough sleep."
        }
    ]
},
{
    "conversation":[
        {
            "system": "You are an AI asssistant."
            "input": "How to study English?",
            "output": "1. Set clear goals. 2. Create a study plan. 3. Build
vocabulary. 4. Practice speaking."
        }
    ]
}]
```

[](

多轮对话数据集格式

多轮对话数据集通常由多轮指令（或问题）及其相应的 GroundTruth 答案组成。假设我们有一个多回合对话数据对于第 n 轮对话

```
System: You are an AI asssistant.
User1：Hello?
Assistant1：Hello! How can I help you?
User2：What's the date today?
Assistant2：Today is Monday, August 14, 2023.
User3：Thank you!
Assistant3：You are welcome.
```

增量预训练数据集格式

由于增量预训练旨在帮助模型学习为特定下游任务量身定制的语言知识和表达能力，因此应使用数据集整个内容对应的损失进行梯度反向传播。因此，数据集的“system”和“input”留空，而“output”由整个语料库数据组成。增量预训练任务对应的数据集格式如下图所示：

```
[{
```

```
    "conversation":[
        {
            "system": "",
            "input": "",
            "output": "I am an artificial intelligence (AI) assistant named Puyu.
I was created by the Shanghai AI Laboratory and my purpose is to assist users
with various tasks through natural language processing technology."
        }
    ]
},
{
    "conversation":[
        {
            "system": "",
            "input": "",
            "output": "I am an artificial intelligence programmed to assist with
various types of tasks, including answering questions, providing information, and
performing automated processes."
        }
    ]
}]
```

三. 指令调优

指令调优阶段常使用单轮和多轮对话数据集，以增强模型对特定指令的响应能力。

在指令调优阶段，我们的目标是训练语言模型根据人类指令进行回答。**因此，通常只有响应部分（Output）的损失用于梯度反向传播，而指令部分（System，Input）的损失不用于权重更新。**

基于此，我们在预处理数据集时引入“system”、“input”和“output”字段。

1. “system”、“input”:字段用于保存不需要计算损失的字段，例如系统和用户指令，
2. “output”:字段用于保存需要计算损失的字段，例如输入指令对应的 GroundTruth 答案。

第1步，构建数据集

我们将数据集格式设置为以下形式：

```
[{
    "conversation":[
        {
            "system": "xxx",
            "input": "xxx",
            "output": "xxx"
        }
    ]
},
{
    "conversation":[
        {
            "system": "xxx",
            "input": "xxx",
            "output": "xxx"
```

```
        },
        {
            "input": "xxx",
            "output": "xxx"
        }
    ]
}]
```

在整个训练阶段，我们将来自单个数据实例的多个“system”、“input”和“output”对合并在一起，然后将其输入到模型中。在每个位置同时计算损耗，但只有与“输出”分量相关的损耗参与梯度反向传播过程。下图阐明了这一过程。

请注意，标记和标记用于指示句子或文本的开头和结尾。

第2步，列出候选模型名称

XTuner 提供了几个现成的配置文件。用户可以使用以下命令查看它们：

```
xtuner list-cfg -p internlm
```

第 3 步，导出配置文件

`-p`用于模糊搜索。如果要训练其他模型，可以将 internlm 替换为 XTuner 支持的其他模型名称。

如果提供的配置文件不符合您的需求，请导出提供的配置文件并进行相应的修改：

```
xtuner copy-cfg ${CONFIG_NAME} ${SAVE_DIR}
```

例如，您可以使用以下命令将名为“internlm_7b_qlora_oasst1_e3”的配置导出到当前目录：

```
xtuner copy-cfg internlm_7b_qlora_oasst1_e3 .
```

第 4 步，修改配置文件

需要对步骤 3 中复制的配置文件进行以下修改：

1. 导入步骤 1 中实现的映射函数。`oasst1_incremental_map_fn`
2. 将 in 替换为 。`dataset_map_fn``train_dataset``custom_map_fn`
3. 将 in 设置为 'None''（因为无需将对话模板添加到增量预训练数据集中）。`template_map_fn``train_dataset`
4. 调整原始数据集的路径。有关 的操作，请参阅用户文档。`load_dataset`
5. 关闭 ，因为模型在增量预训练期间只有延续函数，没有对话函数。`EvaluateChatHook`

```
from xtuner.dataset import process_hf_dataset
from datasets import load_dataset
- from xtuner.dataset.map_fns import oasst1_map_fn, template_map_fn_factory
+ from mmengine.config import read_base
+ with read_base():
```

```
+     from .map_fn import custom_map_fn
...
#######################################################################
#                          PART 1  Settings                           #
#######################################################################
- data_path = 'timdettmers/openassistant-guanaco'
- prompt_template = PROMPT_TEMPLATE.internlm_chat
+ data_path = 'path/to/your/data'
#######################################################################
#                      STEP 3  Dataset & Dataloader                   #
#######################################################################
train_dataset = dict(
    type=process_hf_dataset,
    dataset=dict(type=load_dataset, path=data_path),
    tokenizer=tokenizer,
    max_length=max_length,
-   dataset_map_fn=oasst1_map_fn,
+   dataset_map_fn=custom_map_fn,
-   template_map_fn=dict(
-       type=template_map_fn_factory, template=prompt_template),
+   template_map_fn=None,
    remove_unused_columns=True,
    shuffle_before_pack=True,
    pack_to_max_length=pack_to_max_length)
...
#######################################################################
#                           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,
-       instruction=prompt_template.INSTRUCTION)
]
...
```

第 5 步，检查自定义数据集（可选）

修改配置文件后，可以执行 xtuner/tools/check_custom_dataset.py 脚本来验证数据集的构造是否正确。

```
xtuner check-custom-dataset $CONFIG
```

第 6步，开始训练

修改配置文件后，可以执行 xtuner/tools/check_custom_dataset.py 脚本来验证数据集的构造是否正确。

```
xtuner train $CONFIG
```

`$CONFIG`表示配置文件的文件路径。

第7步，将得到的 PTH 模型转换为 HuggingFace 模型：

```
xtuner convert pth_to_hf ${CONFIG_NAME_OR_PATH} ${PTH_file_dir} ${SAVE_PATH}

#示例
mkdir hf
xtuner convert pth_to_hf \
    ./internlm_chat_7b_qlora_oasst1_e3_copy.py \
    ./work_dirs/internlm_chat_7b_qlora_oasst1_e3_copy/epoch_3.pth \
    ./hf
```

此时，路径中应该长这样：

```
|-- internlm-chat-7b-v1_1
|-- internlm_chat_7b_qlora_oasst1_e3_copy.py
|-- openassistant-guanaco
|   |-- openassistant_best_replies_eval.jsonl
|   `-- openassistant_best_replies_train.jsonl
|-- hf
|   |-- README.md
|   |-- adapter_config.json
|   |-- adapter_model.bin
|   `-- xtuner_config.py
`-- work_dirs
    `-- internlm_chat_7b_qlora_oasst1_e3_copy
        |-- 20231101_152923
        |   |-- 20231101_152923.log
        |   `-- vis_data
        |       |-- 20231101_152923.json
        |       |-- config.py
        |       `-- scalars.json
        |-- epoch_1.pth
        |-- epoch_2.pth
        |-- epoch_3.pth
        |-- internlm_chat_7b_qlora_oasst1_e3_copy.py
        `-- last_checkpoint
```

此时，hf文件夹即为我们平时所理解的所谓 “Lora模型文件”

> 可以简单理解：Lora模型文件 = Adapter文件夹

第8步，部署与测试

```
# 加载 Adapter 模型对话
# xtuner chat internlm/internlm-chat-20b  --adapter  $ADAPTER_DIR --prompt-template internlm_chat
# 例：
xtuner chat ./internlm-chat-7b-v1_1 --adapter ./hf --prompt-template internlm_chat

# Float 16 模型对话
# xtuner chat internlm/internlm-chat-20b --prompt-template internlm_chat

# 4 bit 模型对话
# xtuner chat internlm/internlm-chat-20b  --bits 4 --prompt-template internlm_chat
```

`xtuner chat` 的启动参数

启动参数	干哈滴
--prompt-template	指定对话模板
--system	指定SYSTEM文本
--system-template	指定SYSTEM模板
--bits	LLM位数
--bot-name	bot名称
--with-plugins	指定要使用的插件
--no-streamer	是否启用流式传输
--lagent	是否使用lagent
--command-stop-word	命令停止词
--answer-stop-word	回答停止词
--offload-folder	存放模型权重的文件夹（或者已经卸载模型权重的文件夹）
--max-new-tokens	生成文本中允许的最大token数量
--temperature	温度值
--top-k	保留用于顶k筛选的最高概率词汇标记数
--top-p	如果设置为小于1的浮点数，仅保留概率相加高于top_p的最小一组最有可能的标记
--seed	用于可重现文本生成的随机种子

pth 转 Hugging Face 格式

```
export CONFIG_NAME_OR_PATH=configs/internlm_7b_qlora_alpaca_internlm_assistant.py
export PTH=work_dirs/internlm_7b_qlora_alpaca_internlm_assistant/epoch_2.pth
export SAVE_PATH=./internlm_7b_qlora_alpaca_internlm_assistant/hf

xtuner convert pth_to_hf $CONFIG_NAME_OR_PATH $PTH $SAVE_PATH
```

Merge 权重

```
export NAME_OR_PATH_TO_LLM=./internlm/internlm-7b
export NAME_OR_PATH_TO_ADAPTER=./internlm_7b_qlora_alpaca_internlm_assistant/hf
export SAVE_PATH=./internlm_7b_qlora_alpaca_internlm_assistant/hf_merge

xtuner convert merge $NAME_OR_PATH_TO_LLM $NAME_OR_PATH_TO_ADAPTER $SAVE_PATH --
max-shard-size 2GB
```

测试微调后的模型效果

```
xtuner chat ./internlm/internlm-7b --adapter
./internlm_7b_qlora_alpaca_internlm_assistant/hf --prompt-template internlm_chat
--system-template alpaca
```

四. 增量预训练

增量预训练旨在增强模型在特定领域或任务中的能力。

XTuner 支持使用 HuggingFace Hub 数据集或自定义数据集进行 SFT（监督 FineTune）。它们之间的主要区别在于，在使用 HuggingFace Hub 数据集时，需要将原始数据映射到 XTuner 定义的增量预训练数据格式。对于自定义数据集，建议用户按照增量预训练数据格式构建数据集。

asst1 训练数据集有 9846 行，1 列，列名为 'text'。此“文本”列是增量预训练所需的文本数据。增量预训练数据格式描述在增量预训练过程中，数据格式应为：

```
[{
    "conversation":[
        {
            "input": "",
            "output": "xxx"
        },
    ]
}]
```

因此，您可以使用以下地图函数将原始数据映射到标准格式：

```
# Suppose the function is stored in ./map_fn.py
def custom_map_fn(example):
    """
    >>> train_ds = ds['train'].map(oasst1_map_fn)
    >>> train_ds
    Dataset({
        features: ['text', 'conversation'],
        num_rows: 9846
    })
    >>> train_ds[0]['conversation']
    [{'input': '', 'output': 'xxx'}]
    """
    return {'conversation': [{'input': '', 'output': example['text']}]}
```

训练部分与指令微调相同

五. 用MS-Agent数据集 赋予LLM以Agent能力

概述

MSAgent 数据集每条样本包含一个对话列表（conversations），其里面包含了 system、user、assistant 三种字段。其中：

- system: 表示给模型前置的人设输入，其中有告诉模型如何调用插件以及生成请求
- user: 表示用户的输入prompt，分为两种，通用生成的prompt和调用插件需求的 prompt
- assistant: 为模型的回复。其中会包括插件调用代码和执行代码，调用代码是要 LLM 生成的，而执行代码是调用服务来生成结果的

一条调用网页搜索插件查询“上海明天天气”的数据样本示例如下图所示：

```
{
  "conversations":[
    {
      "from":"system",
      "value":"你有多种能力，可以通过插件集成GoogleSearch..."
    },
    {
      "from":"user",
      "value":"上海明天天气怎么样？"
    },

      "from":"assistant",
      "value":"
        <|startofthink|>
        {
          "api_name": "GoogleSearch",
          "parameters": {"query": "上海 明天 天气"}
        }
        <|endofthink|>
        <|startofexec|>
        {
           "results": ['预报-上海-10日(明天).多云转晴. 23/12°C.', ...]
        }
        <|endofexec|>
        上海明天多云转晴，最低气温12度，最高气温23度。
      "
    }
  ]
}
```

详细步骤如下：

第1步，数据集

提前下载MS-Agent数据集，也可以自己构建

第2步:，查看并复制配置文件

```
# 准备工作
cd ~
mkdir ft-interlm7bchatv11-msagent
cd ft-interlm7bchatv11-msagent
cp -r ~/ft-internlm-7b-chat-v1_1-ft-oasst1/internlm-chat-7b-v1_1 .

# 查看配置文件
xtuner list-cfg | grep msagent

# 复制配置文件当当前目录
xtuner copy-cfg internlm_7b_qlora_msagent_react_e3_gpu8 .
```

第3步，修改配置文件

后续步骤与指令微调相同

六. 远程服务器文件操作

SCP远程拷贝

> 注意下方指令需要在您**本地**的机器上执行，向实例中拷贝数据，而不是在实例中执行该命令;
>
> 如果您是Windows用户，默认Windows系统未安装SSH客户端，推荐下载使用Cmder工具，免安装解压即用。

复制您的ssh登录指令，指令格式为：`ssh -p 35394 root@region-1.autodl.com` （注意35394为端口号，region-1.autodl.com为远程地址，请更换为您的实例端口和地址）

那么scp远程拷贝文件的指令为：`scp -rP 35394 <本地文件/文件夹> root@region-1.autodl.com:/root/autodl-tmp` （注意需要在您**本地**的机器上执行）

如果是将实例中的数据拷贝到本地，那么scp远程拷贝指令为：`scp -rP 35394 root@120.92.100.9:<实例中的文件/文件夹> <本地文件/文件夹>`

高级使用方法

上面直接scp文件夹的方法，如果小文件多，那么scp速度会非常缓慢。可以使用以下方法拷贝tar流（前提需要本地电脑有tar命令）

```
cd <要拷贝的文件夹目录下>
tar cf - * | ssh -p 35394 root@region-1.autodl.com "cd /root/autodl-tmp && tar xf -"
```