大语言模型微调

罗从良 (23 级秋)

2023 年 12 月 13 日

本文针对大语言模型 llama2 微调进行操作讲述。llama2 是 meta 最新开源的语言大模型，训练数据集达 2 万亿 token，上下文长度是由 llama 的 2048 扩展到 4096，可以理解和生成更长的文本，包括 7B、13B 和 70B 三个模型，在各种基准集的测试上表现突出。而且该模型可用于研究和商业用途。这对于我们使用 llama2 进行微调是非常友好的。我们这里所说的微调实际上是在特定的任务上进行精细化训练, 从而提高模型的性能。当然也有一种预训练微调，这个相对对资源和数据量有一定的要求。

本次微调使用了 4 张 V100 进行训练。训练过程中发现大约 20 多 G 便可完成训练。所以一种 32G 的卡是可以完成微调训练的。

本次微调是使用 Llama-2-7b-hf 模型作为基座模型，如果用来做对话的话，可以使用 LLama-2-7b-chat 作为基座模型。

在论文”Data Quality Is All You Need。”，MetaAI 进行实验时发现，少量高质量数据集训练模型的效果，要好于大量低质量数据集的训练效果。因此进行 SFT 时候，可以不要一味地追求量，有时质量更为重要。

微调时初始学习率为 2e-4，并采用余弦学习率下降，权重衰减为 0.1，训练批次大小为 64，最大长度为 4096。为了提高模型训练效率，将多组数据进行拼接，尽量填满 4096，每条数据直接用停止符隔开，计算 loss 时仅计算每条样本 target 内容的 loss。

总的来说,Llama2 微调技术让大型语言模型可以高效地迁移到各种下游任务, 同时保持通用语言理解的能力, 是当前自然语言处理领域的热点技术之一。随着计算能力的增强和参数量的扩大, 这类技术还有很大的改进空间。

以下我使用使用 vscode 利用 jupyter 插件进行微调训练操作。

它包括了 python 环境、模块或包引入，数据预处理、bnb 机制处理、LoRa 方法操作等等。

[ ]:
```
!which python
```

```
/data/lcl/anaconda3/envs/chinese_llama2/bin/python
```

[1]:
```
!nvidia-smi
```

```
huggingface/tokenizers: The current process just got forked, after parallelism
has already been used. Disabling parallelism to avoid deadlocks…
To disable this warning, you can either:
        - Avoid using `tokenizers` before the fork if possible
        - Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true |
false)
Mon Dec 11 08:27:49 2023
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 460.67       Driver Version: 460.67       CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla V100-PCIE…  Off  | 00000000:3B:00.0 Off |                  0 |
| N/A   37C    P0    35W / 250W |   2299MiB / 32510MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   1  Tesla V100-PCIE…  Off  | 00000000:86:00.0 Off |                  0 |
| N/A   38C    P0    34W / 250W |   2469MiB / 32510MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   2  Tesla V100-PCIE…  Off  | 00000000:AF:00.0 Off |                  0 |
| N/A   36C    P0    37W / 250W |   2521MiB / 32510MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   3  Tesla V100-PCIE…  Off  | 00000000:D8:00.0 Off |                  0 |
| N/A   38C    P0    33W / 250W |   2307MiB / 32510MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0   N/A  N/A     66429      C   …chinese_llama2/bin/python     2167MiB |
|    1   N/A  N/A     66429      C   …chinese_llama2/bin/python     2337MiB |
|    2   N/A  N/A     66429      C   …chinese_llama2/bin/python     2389MiB |
|    3   N/A  N/A     66429      C   …chinese_llama2/bin/python     2175MiB |
+-----------------------------------------------------------------------------+
```

[2]:
```
import argparse
import bitsandbytes as bnb
from datasets import load_dataset
from functools import partial
import os
import torch
from peft import LoraConfig, get_peft_model, prepare_model_for_kbit_training,␣
↪AutoPeftModelForSeq2SeqLM
from transformers import AutoModelForCausalLM, AutoTokenizer, set_seed,␣
↪Trainer, TrainingArguments, BitsAndBytesConfig, \
    DataCollatorForLanguageModeling, Trainer, TrainingArguments
```

[3]:
```
def load_model(model_name, bnb_config):
    n_gpus = torch.cuda.device_count()
    max_memory = f'{40960}MB'

    # 正常情况下, 我们直接加载 model_name, 可以查找一下资料关于
quantization,device_map,max_memory 等参数
    model = AutoModelForCausalLM.from_pretrained(
        model_name,
        quantization_config=bnb_config,
        device_map='auto',# 高效地将模型分配给可用资源。
        max_memory={i: max_memory for i in range(n_gpus)},
    )
    tokenizer = AutoTokenizer.from_pretrained(model_name, use_auth_token=True)

    # Needed for LLaMA tokenizer
    tokenizer.pad_token = tokenizer.eos_token

    return model, tokenizer
```

[16]:
```python
from datasets import load_dataset

dataset = load_dataset("/data/lcl/LLM/llama/datasets", split="train")
```

```
HF google storage unreachable. Downloading and preparing it from source

Downloading data files:   0%|          | 0/1 [00:00<?, ?it/s]

Extracting data files:   0%|          | 0/1 [00:00<?, ?it/s]

Generating train split: 0 examples [00:00, ? examples/s]
```

[17]:
```python
# 查看数据集
print(f'Number of prompts: {len(dataset)}')
print(dataset, type(dataset))
```

```
Number of prompts: 15011
Dataset({
    features: ['id', 'category', 'instruction_zh', 'context_zh', 'response',
'instruction', 'context'],
    num_rows: 15011
}) <class 'datasets.arrow_dataset.Dataset'>
```

[18]:
```python
# 探索数据集里具体的内容
for e in dataset:
    print(e)
    print(e['instruction'])
```

```
{'id': 1635, 'category': 'information_extraction', 'instruction_zh':
'根据以下关于瑞典经济的文章，哪个经济部门占据了最大的产出？', 'context_zh': '瑞典是一个出口导向的混合经济体，拥有现代化的分销系统、优秀的内外
部通讯和熟练的劳动力。木材、水力和铁矿石构成了一个经济体的资源基础，这个经济体严重依赖对外贸易。瑞典的工程部门占产出和出口的 50%。电信、汽车工业和制药工业也非
常重要。农业占 GDP 和就业的 2%。军火工业具有高度先进的技术声誉。', 'response':␣
↪'According to this passage, the
engineering sector accounts for the largest output, generating 50% of output and
exports.', 'instruction': 'Based on the following passage regarding the economy
of Sweden, what is the economic sector that accounts for the largest output?',
'context': "Sweden is an export-oriented mixed economy featuring a modern
distribution system, excellent internal and external communications, and a
skilled labor force. Timber, hydropower and iron ore constitute the resource
```

base of an economy heavily oriented toward foreign trade. Sweden's engineering
sector accounts for 50% of output and exports. Telecommunications, the
automotive industry and the pharmaceutical industries are also of great
importance. Agriculture accounts for 2 percent of GDP and employment. The
armaments industry has a technologically highly advanced reputation."}
Based on the following passage regarding the economy of Sweden, what is the
economic sector that accounts for the largest output?
{'id': 4959, 'category': 'classification', 'instruction_zh': '`这些国家是共产主义
国家吗：阿富汗、
阿尔巴尼亚、阿尔及利亚、安道尔、安哥拉、安提瓜和巴布达、阿根廷、亚美尼亚、澳大利亚、奥
地利、阿塞拜疆、巴哈马、巴林、孟加拉国、巴巴多斯、白俄罗斯、比利时、伯利兹
、贝宁、不丹、玻利维亚、波斯尼亚和黑塞哥维那、博茨瓦纳、巴西、文莱、保加利亚、布基纳法
索、布隆迪、科特迪瓦、佛得角、柬埔寨、喀麦隆、加拿大、中非共和国、乍得、智
利、中国、哥伦比亚、科摩罗、刚果（布）、哥斯达黎加、克罗地亚、古巴、塞浦路斯、捷克共和国、
刚果（金）、丹麦、吉布提、多米尼克、多米尼加共和国、厄瓜多尔、埃及、萨
尔瓦多、赤道几内亚、厄立特里亚、爱沙尼亚、斯威士兰、埃塞俄比亚、斐济、芬兰、法国、加蓬、
冈比亚、格鲁吉亚、德国、加纳、希腊、格林纳达、危地马拉、几内亚、几内亚比
绍、圭亚那、海地、梵蒂冈、洪都拉斯、匈牙利、冰岛、印度、印度尼西亚、伊朗、伊拉克、爱尔
兰、以色列、意大利、牙买加、日本、约旦、哈萨克斯坦、肯尼亚、基里巴斯、科威
特、吉尔吉斯斯坦、老挝、拉脱维亚、黎巴嫩、莱索托、利比里亚、利比亚、列支敦士登、立陶宛、
卢森堡、马达加斯加、马拉维、马来西亚、马尔代夫、马里、马耳他、马绍尔群岛
、毛里塔尼亚、毛里求斯、墨西哥、密克罗尼西亚、摩尔多瓦、摩纳哥、蒙古、黑山、摩洛哥、莫
桑比克、缅甸（前缅甸）、纳米比亚、瑙鲁、尼泊尔、荷兰、新西兰、尼加拉瓜、尼
日尔、尼日利亚、朝鲜、北马其顿、挪威、阿曼、巴基斯坦、帕劳、巴勒斯坦国、巴拿马、巴布亚
新几内亚、巴拉圭、秘鲁、菲律宾、波兰、葡萄牙、卡塔尔、罗马尼亚、俄罗斯、卢
旺达、圣基茨和尼维斯、圣卢西亚、圣文森特和格林纳丁斯、萨摩亚、圣马力诺、圣多美和普林西
比、沙特阿拉伯、塞内加尔、塞尔维亚、塞舌尔、塞拉利昂、新加坡、斯洛伐克、斯
洛文尼亚、所罗门群岛、索马里、南非、韩国、南苏丹、西班牙、斯里兰卡、苏丹、苏里南、瑞典、
瑞士、叙利亚、塔吉克斯坦、坦桑尼亚、泰国、东帝汶、多哥、汤加、特立尼达和
多巴哥、突尼斯、土耳其、土库曼斯坦、', 'context_zh': '', 'response': 'China, Cuba,␣
↪Laos, Vietnam,
North Korea', 'instruction': "Are they communist countries: Afghanistan,
Albania, Algeria, Andorra, Angola, Antigua and Barbuda, Argentina, Armenia,
Australia, Austria, Azerbaijan, Bahamas, Bahrain, Bangladesh, Barbados, Belarus,
Belgium, Belize, Benin, Bhutan, Bolivia, Bosnia and Herzegovina, Botswana,
Brazil, Brunei, Bulgaria, Burkina Faso, Burundi, Côte d'Ivoire, Cabo Verde,

Cambodia, Cameroon, Canada, Central African Republic, Chad, Chile, China,
Colombia, Comoros, Congo (Congo-Brazzaville), Costa Rica, Croatia, Cuba, Cyprus,
Czechia (Czech Republic), Democratic Republic of the Congo, Denmark, Djibouti,
Dominica, Dominican Republic, Ecuador, Egypt, El Salvador, Equatorial Guinea,
Eritrea, Estonia, Eswatini , Ethiopia, Fiji, Finland, France, Gabon, Gambia,
Georgia, Germany, Ghana, Greece, Grenada, Guatemala, Guinea, Guinea-Bissau,
Guyana, Haiti, Holy See, Honduras, Hungary, Iceland, India, Indonesia, Iran,
Iraq, Ireland, Israel, Italy, Jamaica, Japan, Jordan, Kazakhstan, Kenya,
Kiribati, Kuwait, Kyrgyzstan, Laos, Latvia, Lebanon, Lesotho, Liberia, Libya,
Liechtenstein, Lithuania, Luxembourg, Madagascar, Malawi, Malaysia, Maldives,
Mali, Malta, Marshall Islands, Mauritania, Mauritius, Mexico, Micronesia,
Moldova, Monaco, Mongolia, Montenegro, Morocco, Mozambique, Myanmar (formerly
Burma), Namibia, Nauru, Nepal, Netherlands, New Zealand, Nicaragua, Niger,
Nigeria, North Korea, North Macedonia, Norway, Oman, Pakistan, Palau, Palestine
State, Panama, Papua New Guinea, Paraguay, Peru, Philippines, Poland, Portugal,
Qatar, Romania, Russia, Rwanda, Saint Kitts and Nevis, Saint Lucia, Saint
Vincent and the Grenadines, Samoa, San Marino, Sao Tome and Principe, Saudi
Arabia, Senegal, Serbia, Seychelles, Sierra Leone, Singapore, Slovakia,
Slovenia, Solomon Islands, Somalia, South Africa, South Korea, South Sudan,
Spain, Sri Lanka, Sudan, Suriname, Sweden, Switzerland, Syria, Tajikistan,
Tanzania, Thailand, Timor-Leste, Togo, Tonga, Trinidad and Tobago, Tunisia,
Turkey, Turkmenistan, Tuvalu, Uganda, Ukraine, United Arab Emirates, United
Kingdom, United States of America, Uruguay, Uzbekistan, Vanuatu, Venezuela,
Vietnam, Yemen, Zambia, Zimbabwe", 'context': ''}
Are they communist countries: Afghanistan, Albania, Algeria, Andorra, Angola,
Antigua and Barbuda, Argentina, Armenia, Australia, Austria, Azerbaijan,
Bahamas, Bahrain, Bangladesh, Barbados, Belarus, Belgium, Belize, Benin, Bhutan,
Bolivia, Bosnia and Herzegovina, Botswana, Brazil, Brunei, Bulgaria, Burkina
Faso, Burundi, Côte d'Ivoire, Cabo Verde, Cambodia, Cameroon, Canada, Central
African Republic, Chad, Chile, China, Colombia, Comoros, Congo (Congo-
Brazzaville), Costa Rica, Croatia, Cuba, Cyprus, Czechia (Czech Republic),
Democratic Republic of the Congo, Denmark, Djibouti, Dominica, Dominican
Republic, Ecuador, Egypt, El Salvador, Equatorial Guinea, Eritrea, Estonia,
Eswatini , Ethiopia, Fiji, Finland, France, Gabon, Gambia, Georgia, Germany,
Ghana, Greece, Grenada, Guatemala, Guinea, Guinea-Bissau, Guyana, Haiti, Holy
See, Honduras, Hungary, Iceland, India, Indonesia, Iran, Iraq, Ireland, Israel,
Italy, Jamaica, Japan, Jordan, Kazakhstan, Kenya, Kiribati, Kuwait, Kyrgyzstan,

```
Laos, Latvia, Lebanon, Lesotho, Liberia, Libya, Liechtenstein, Lithuania,
Luxembourg, Madagascar, Malawi, Malaysia, Maldives, Mali, Malta, Marshall
Islands, Mauritania, Mauritius, Mexico, Micronesia, Moldova, Monaco, Mongolia,
Montenegro, Morocco, Mozambique, Myanmar (formerly Burma), Namibia, Nauru,
Nepal, Netherlands, New Zealand, Nicaragua, Niger, Nigeria, North Korea, North
Macedonia, Norway, Oman, Pakistan, Palau, Palestine State, Panama, Papua New
Guinea, Paraguay, Peru, Philippines, Poland, Portugal, Qatar, Romania, Russia,
Rwanda, Saint Kitts and Nevis, Saint Lucia, Saint Vincent and the Grenadines,
Samoa, San Marino, Sao Tome and Principe, Saudi Arabia, Senegal, Serbia,
Seychelles, Sierra Leone, Singapore, Slovakia, Slovenia, Solomon Islands,
Somalia, South Africa, South Korea, South Sudan, Spain, Sri Lanka, Sudan,
Suriname, Sweden, Switzerland, Syria, Tajikistan, Tanzania, Thailand, Timor-
Leste, Togo, Tonga, Trinidad and Tobago, Tunisia, Turkey, Turkmenistan, Tuvalu,
Uganda, Ukraine, United Arab Emirates, United Kingdom, United States of America,
Uruguay, Uzbekistan, Vanuatu, Venezuela, Vietnam, Yemen, Zambia, Zimbabwe
{'id': 3858, 'category': 'classification', 'instruction_zh': '我正在为我的婚礼注册
礼物，需要包括一
些在我新家有用的物品。哪些物品是可以作为婚礼礼物的家庭用品：搅拌机、咖啡机、公交车票、
毛巾、床单、滑板、手机、汽车、餐具、银器、健身会员卡、玻璃器皿、礼服、刹车
片、自行车、相框。', 'context_zh': '', 'response': 'A blender, coffee maker,␣
↪towels,
sheets, dishes, silverware, glassware, picture frames are examples of household
items that can be given as a wedding gift.', 'instruction': 'I am registering
for gifts for my wedding and need to include items that would be useful in my
new home. Which items are household items that can be given as a wedding gift:
blender, coffee maker, bus fare, towels, sheets, skateboard, cell phone, car,
dishes, silverware, gym membership, glassware, tuxedo, brake pads, bicycle,
picture frames', 'context': ''}
```

[31]:
```
# 预处理数据集


# instrcuction fine-tuning(指令微调）是一种常用的技术，用于针对特定的下游用例微调基
础 LLM。


# 按如下方式设置提示的格式:
```

```
Below is an instruction that describes a task. Write a response that appropriately completes t
```

```
### Instruction:
Sea or Mountain

### Response:
I believe Mountain are more attractive but Ocean has it's own beauty and this tropical weather

### End
```

[20]:
```
# 通过主题标签分隔每个提示部分
def create_prompt_formats(sample):
    """
        格式化为 ('instruction', 'context', 'response')
        然后组成为新的特征行。
    """
    INTRO_BLURB = "Below is an instruction that describes a task. Write a␣
↪response that appropriately completes the request."
    INSTRUCTION_KEY = "### Instruction:"
    INPUT_KEY = "Input:"
    RESPONSE_KEY = "### Response:"
    END_KEY = "### End"

    blurb = f"{INTRO_BLURB}"
    instruction = f"{INSTRUCTION_KEY}\n{sample['instruction']}"
    input_context = f"{INPUT_KEY}\n{sample['context']}" if sample['context']␣
↪else None
    response = f"{RESPONSE_KEY}\n{sample['response']}"
    end = f"{END_KEY}"

    parts = [part for part in [blurb, instruction, input_context, response,␣
↪end] if part]

    formatted_prompt = "\n\n".join(parts)

    sample["text"] = formatted_prompt

    return sample
```

```
[21]: # 使用 model tokenizer 将这些 prompts into tokenized ones
# 目标是创建长度均匀的输入序列（适用于微调语言模型，因为它可以最大限度地提高效率并最小化计算开销），并且不得超过模型的最大标记限制。

def get_max_length(model):
    conf = model.config
    max_length = None
    for length_setting in ["n_positions", "max_position_embeddings",␣
 ↪"seq_length"]:
        max_length = getattr(model.config, length_setting, None)
        if max_length:
            print(f"找到最大长度: {max_length}")
            break
    if not max_length:
        max_length = 1024
        print(f"使用默认最大长度: {max_length}")
    return max_length

# 准备 batch
# 对一个批次的数据进行预处理
def preprocess_batch(batch, tokenizer, max_length):
    """
        Tokenizer a batch
    """
    # batch["text"] 表示批次中的文本数据，max_length 表示标记化后每个文本序列的最大长度，trucation=True 表示文本长度超过了 max_length 则进行截取
    # 返回标记后的文本
    return tokenizer(
        batch["text"],
        max_length = max_length,
        truncation=True,
    )

# 准备数据集
def preprocess_dataset(tokenizer: AutoTokenizer, max_length: int, seed, dataset:
 ↪ str):
    """
        准备数据集
    """
    print("准备数据集")
    dataset = dataset.map(create_prompt_formats)#,batched=True)

    # 对数据集中的每个 batch 进行预处理, 移除'instruction', 'context',␣
↪'response', 'category'
    # from functools import partial
    """
        def add(x, y):
            return x + y

        add_5 = partial(add, 5)
        print(add_5(3)) # 输出 8
        print(add_5(7)) # 输出 12
    """
    _preprocessing_function = partial(preprocess_batch, max_length=max_length,␣
↪tokenizer=tokenizer)
    dataset = dataset.map(
        _preprocessing_function,
        batched=True,
        remove_columns=["instruction", "context", "response", "text",␣
↪"category"],
    )

    # Filter out samples that have input_ids exceeding max_length
    dataset = dataset.filter(lambda sample: len(sample["input_ids"]) <␣
↪max_length)

    # Shuffle dataset
    dataset = dataset.shuffle(seed=seed)

    return dataset
```

创建 bitsandbytes 配置，这将允许我们以 4 位加载我们的 LLM。这样，我们可以将使用的内存除以 4，并在较小的设备上导入模型。为了节省内存，我们选择应用 bfloat16 计算数据类型和嵌套量化。

[22]:
```python
def create_bnb_config():
    bnb_config = BitsAndBytesConfig(
        load_in_4bit=True,
        bnb_4bit_use_double_quant=True,
        bnb_4bit_quant_type='nf4',
        bnb_4bit_compute_dtype=torch.bfloat16,
    )
    return bnb_config
```

[23]:
```python
# 为了使用 LoRa 方法，我们需要将模型包装为 PeftModel
# 因此，我们需要实现 LoRa 配置
# 函数需要目标模块来更新必要的矩阵。

def create_peft_config(modules):
    """
        为您的模型创建参数高效的微调配置
    :param modules: 要应用 Lora 的模块的名称
    """
    config = LoraConfig(
        r=16, # 更新矩阵维度 16
        lora_alpha=64, # 表示缩放参数为 64
        target_modules=modules, # 表示要应用 Lora 的模块名称
        lora_dropout=0.1, #  layers 的 dropout 概率为 0.1
        bias="none", # 表示偏差为 none
        task_type="CAUSAL_LM",
    )
    return config
```

[24]:
```python
# 函数用来获取 lora 操作系列模块
def find_all_linear_names(model):
    cls = bnb.nn.Linear4bit #if args.bits == 4 else (bnb.nn.Linear8bitLt if␣
 ↪args.bits == 8 else torch.nn.Linear)
    lora_module_names = set()
    for name, module in model.named_modules():
        if isinstance(module, cls):
            names = name.split('.')
            lora_module_names.add(names[0] if len(names) == 1 else names[-1])

    if 'lm_head' in lora_module_names:  # needed for 16-bit
        lora_module_names.remove('lm_head')
    return list(lora_module_names)
```

[25]:
```
# 创建一个辅助函数用来查看模型中的可训练参数
def print_trainable_parameters(model,use_4bit=False):
    """
        打印模型中的训练参数
    """
    trainable_params = 0
    all_param = 0
    for _, param in model.named_parameters():
        num_params = param.numel()
        # 如果使用 DS Zero 3 并且权重初始化为空。
        if num_params == 0 and hasattr(param, "ds_numel"):
            num_params = param.ds_numel

        all_param += num_params
        if param.requires_grad:
            trainable_params += num_params
    if use_4bit:
        trainable_params /= 2
    # print(f"all params: {all_param:, d} || trainable params:␣
  ↪{trainable_params:, d} || trainable%: {100 * trainable_params / all_param}")
```

[26]:
```
# 加载预训练模型和配置
model_name = "meta-llama/Llama-2-7b-hf"

bnb_config = create_bnb_config()

model, tokenizer = load_model(model_name, bnb_config)
```

```
Loading checkpoint shards:   0%|          | 0/2 [00:00<?, ?it/s]
```

[27]:
```
# 准备数据集
max_length = get_max_length(model)
dataset = preprocess_dataset(tokenizer, max_length, 515, dataset)
```

```
找到最大长度: 4096
准备数据集

Map:   0%|          | 0/15011 [00:00<?, ? examples/s]

Map:   0%|          | 0/15011 [00:00<?, ? examples/s]

Filter:   0%|          | 0/15011 [00:00<?, ? examples/s]
```

[28]:
```
def train(model, tokenizer, dataset, output_dir):
    # 预处理启动梯度检查点技术, 以在微调过程中减少内存使用
    model.gradient_checkpointing_enable()

    # 2 - Using the prepare_model_for_kbit_training method from PEFT
    model = prepare_model_for_kbit_training(model)

    # Get lora module names
    modules = find_all_linear_names(model)

    # Create PEFT config for these modules and wrap the model to PEFT
    peft_config = create_peft_config(modules)
    model = get_peft_model(model, peft_config)

    # Print information about the percentage of trainable parameters
    print_trainable_parameters(model)

    # Training parameters
    trainer = Trainer(
        model=model,
        train_dataset=dataset,
        args=TrainingArguments(
            per_device_train_batch_size=1,
            gradient_accumulation_steps=4,
            warmup_steps=2,
            max_steps=20,
            learning_rate=2e-4,
            fp16=True,
            logging_steps=1,
            output_dir="outputs",
            optim="paged_adamw_8bit",
        ),
        data_collator=DataCollatorForLanguageModeling(tokenizer, mlm=False)
    )

    model.config.use_cache = False  # re-enable for inference to speed up␣
    ↪predictions for similar inputs

    ### SOURCE https://github.com/artidoro/qlora/blob/main/qlora.py
    # Verifying the datatypes before training

    dtypes = {}
    for _, p in model.named_parameters():
        dtype = p.dtype
        if dtype not in dtypes: dtypes[dtype] = 0
        dtypes[dtype] += p.numel()
    total = 0
    for k, v in dtypes.items(): total+= v
    for k, v in dtypes.items():
        print(k, v, v/total)

    do_train = True

    # Launch training
    print("Training...")

    if do_train:
        train_result = trainer.train()
        metrics = train_result.metrics
        trainer.log_metrics("train", metrics)
        trainer.save_metrics("train", metrics)
        trainer.save_state()
```

```
        print(metrics)

    ###

    # Saving model
    print("Saving last checkpoint of the model...")
    os.makedirs(output_dir, exist_ok=True)
    # 为了稍后加载和使用模型进行推理，我们使用了该 trainer.model.
↪save_pretrained(output_dir) 函数，该函数保存了微调模型的权重、配置和分词器文件。
    trainer.model.save_pretrained(output_dir)

    # Free memory for merging weights
    del model
    del trainer
    torch.cuda.empty_cache()
```

[29]:
```
output_dir = "results/llama2/final_checkpoint"
train(model, tokenizer, dataset, output_dir)
```

```
Detected kernel version 4.15.0, which is below the recommended minimum of 5.5.0;
this can cause the process to hang. It is recommended to upgrade the kernel to
the minimum version or higher.
```

```
torch.float32 302387200 0.08541070604255438
torch.uint8 3238002688 0.9145892939574456
Training…
```

```
You're using a LlamaTokenizerFast tokenizer. Please note that with a fast
tokenizer, using the `__call__` method is faster than using a method to encode
the text followed by a call to the `pad` method to get a padded encoding.
```

```
<IPython.core.display.HTML object>
```

```
***** train metrics *****
```

```
  epoch                    =       0.01
  total_flos               =   439355GF
  train_loss               =     1.4391
  train_runtime            = 0:01:14.31
  train_samples_per_second =      1.076
  train_steps_per_second   =      0.269
{'train_runtime': 74.3165, 'train_samples_per_second': 1.076,
'train_steps_per_second': 0.269, 'total_flos': 471754134798336.0, 'train_loss':
1.4391322791576386, 'epoch': 0.01}
Saving last checkpoint of the model…
```

[30]:
```
# 一旦我们有了微调的权重，我们就可以构建微调的模型，并将其保存到一个新目录，以及其关联的分词器。通过执行这些步骤，我们可以为推理提供内存高效的微调模型和分词器！
from peft import AutoPeftModelForCausalLM
model = AutoPeftModelForCausalLM.from_pretrained(output_dir, device_map="auto",␣
 ↪torch_dtype=torch.bfloat16)
model = model.merge_and_unload()

output_merged_dir = "results/llama2/final_merged_checkpoint"
os.makedirs(output_merged_dir, exist_ok=True)
model.save_pretrained(output_merged_dir, safe_serialization=True)

# save tokenizer for easy inference
tokenizer = AutoTokenizer.from_pretrained(model_name)
tokenizer.save_pretrained(output_merged_dir)
```

```
Loading checkpoint shards:   0%|          | 0/2 [00:00<?, ?it/s]
```

[30]:
```
('results/llama2/final_merged_checkpoint/tokenizer_config.json',
 'results/llama2/final_merged_checkpoint/special_tokens_map.json',
 'results/llama2/final_merged_checkpoint/tokenizer.model',
 'results/llama2/final_merged_checkpoint/added_tokens.json',
 'results/llama2/final_merged_checkpoint/tokenizer.json')
```

完成模型训练后，可以进行推理

[32]:
```
tokenizer = AutoTokenizer.from_pretrained(model_name)
print(tokenizer)
```

```
LlamaTokenizerFast(name_or_path='meta-llama/Llama-2-7b-hf', vocab_size=32000,
model_max_length=1000000000000000019884624838656, is_fast=True,
padding_side='right', truncation_side='right', special_tokens={'bos_token':
AddedToken("<s>", rstrip=False, lstrip=False, single_word=False,
normalized=False), 'eos_token': AddedToken("</s>", rstrip=False, lstrip=False,
single_word=False, normalized=False), 'unk_token': AddedToken("<unk>",
rstrip=False, lstrip=False, single_word=False, normalized=False)},
clean_up_tokenization_spaces=False)
```

[35]:
```python
from transformers import AutoModelForCausalLM
output_merged_dir = "results/llama2/final_merged_checkpoint"

model = AutoModelForCausalLM.from_pretrained(output_merged_dir)
print(model)
```

```
Loading checkpoint shards:   0%|          | 0/2 [00:00<?, ?it/s]
```

```
LlamaForCausalLM(
  (model): LlamaModel(
    (embed_tokens): Embedding(32000, 4096, padding_idx=0)
    (layers): ModuleList(
      (0-31): 32 x LlamaDecoderLayer(
        (self_attn): LlamaAttention(
          (q_proj): Linear(in_features=4096, out_features=4096, bias=False)
          (k_proj): Linear(in_features=4096, out_features=4096, bias=False)
          (v_proj): Linear(in_features=4096, out_features=4096, bias=False)
          (o_proj): Linear(in_features=4096, out_features=4096, bias=False)
          (rotary_emb): LlamaRotaryEmbedding()
        )
        (mlp): LlamaMLP(
          (gate_proj): Linear(in_features=4096, out_features=11008, bias=False)
          (up_proj): Linear(in_features=4096, out_features=11008, bias=False)
          (down_proj): Linear(in_features=11008, out_features=4096, bias=False)
          (act_fn): SiLUActivation()
        )
        (input_layernorm): LlamaRMSNorm()
        (post_attention_layernorm): LlamaRMSNorm()
      )
    )
    (norm): LlamaRMSNorm()
  )
  (lm_head): Linear(in_features=4096, out_features=32000, bias=False)
)
```

[37]:
```
model, tokenizer = load_model(output_merged_dir , bnb_config)
```

```
Loading checkpoint shards:   0%|          | 0/2 [00:00<?, ?it/s]
```

```
/data/lcl/anaconda3/envs/chinese_llama2/lib/python3.10/site-
packages/transformers/tokenization_utils_base.py:1714: FutureWarning: The
`use_auth_token` argument is deprecated and will be removed in v5 of
Transformers.
  warnings.warn(
```

[54]:
```
from transformers import GenerationConfig

model.eval()
eval_prompt = """
Below is an instruction that describes a task. Write a response that␣
 ↪appropriately completes the request.

### Instruction:
给我介绍一下北京
"""
generation_config = GenerationConfig(
    temperature=0.1,
    top_p=0.75,
    top_k=40,
    num_beams=4,
)


# generating reply
with torch.no_grad():
    inputs = tokenizer(eval_prompt, return_tensors="pt")
    generation_output = model.generate(
        input_ids=inputs.input_ids,
        generation_config=generation_config,
        return_dict_in_generate=True,
        output_scores=True,
        max_new_tokens=4096,
    )
    print('\n回复: ', tokenizer.decode(generation_output.sequences[0]))
```

```
回复: <s>
Below is an instruction that describes a task. Write a response that
appropriately completes the request.

### Instruction:
给我介绍一下北京

### Response:
北京是中国最大的城市，也是中国政治、经济、科技、文化中心，是中国最大的城市，也是中国政
治、经济、科技、文化中心。

### End:
```

总的来说，当前高效指令微调优化的技术主要涉及的点有减少参数量、压缩梯度、量化等方式来来降低计算和减少内存消耗。这些方法在降低资源占用方面非常有成效，但也存在一定的缺点，比如精度损失、收敛稳定性等问题。本文设计的微调技术采用 LoRA，它的原理就是将模型权重分解为低秩分量进行更新, 使调优局限在相关任务子空间。这样做的优势可以减少调优的参数量，降低计算内存。缺点采用低秩分解操作可能会削弱模型表征能力。