LLM 基底模型的指令微调

刘逸飞，522031910023
盛熙然，522031910087
冯海桐，522031910557

1 计算平台

我们小组的大作业，即 LLM 基底模型的指令微调，包括模型训练的相关测评，全都在 Kaggle 平台上进行。

2 实验设置

2.1 实验平台的设置

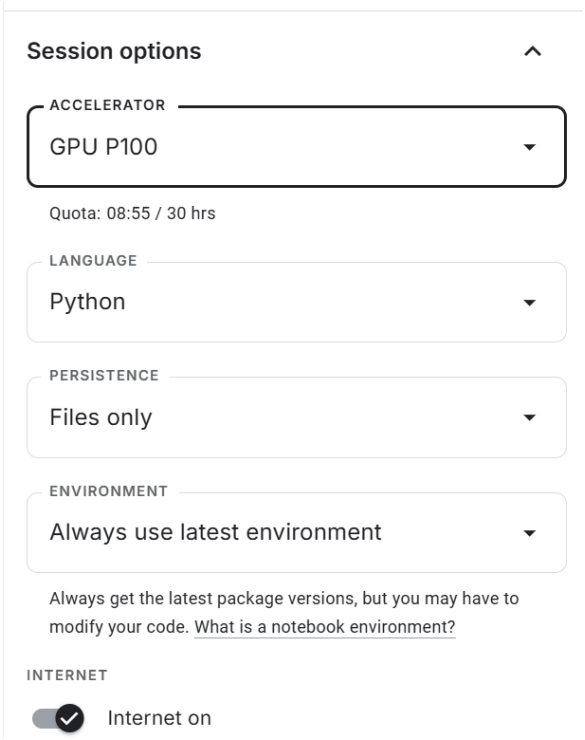

图 1：实验平台的设置

如图所示，在实验平台的设置中，我们选用了平台提供的 GPU P100 进行实验，并以 Files only 的方式保存输出文件，同时将环境设置为永远是最新的环境。

2.2 微调参数的设置

在我们最终提交的代码文件中，对许多的训练参数进行了微调。

```
# Pass your training arguments.
# NOTE [IMPORTANT!!!] DO NOT FORGET TO PASS PROPER ARGUMENTS TO SAVE YOUR CHECKPOINTS!!!
sys.argv = [
    "notebook",
    "--model_name_or_path", "/kaggle/input/qwen2.5/transformers/0.5b/1",
    "--dataset_path", "/kaggle/input/alpaca-language-instruction-training/train.csv",
    "--output_dir", "/kaggle/working/",
    "--logging_dir","/kaggle/working/logs",
    "--logging_steps", "40",   # 每10步输出一次训练损失
    "--logging_strategy", "steps",
    "--save_steps","4000",  # 每4000步保存一次模型检查点
    "--save_total_limit","1",  # 最多保留1个检查点，删除较旧的
    "--report_to","none",  # 禁用 W&B 集成
    "--torch_dtype" , "bfloat16",
    "--remove_unused_columns","False",#保留未使用的列
    "--per_device_train_batch_size" ,"4", # batch_size 4
    "--per_device_eval_batch_size", "4", # batch_size 4
    "--gradient_accumulation_steps", "8", # 累加向后传播次数
    "--lr_scheduler_type", "linear",  # 使用线性学习率调度器
    "--learning_rate", "5e-5", # 学习率
    "--num_train_epochs", "3", # epochs的数目
    "--adam_epsilon", "1e-8",  # 小 epsilon 提高稳定性
    "--evaluation_strategy","no",   # 每隔一定步数进行验证
    # "--load_best_model_at_end","True",   # 训练结束后加载最佳模型
    # "--eval_steps","4000",                # 每 4000 步验证一次
    # "--greater_is_better","False",  # 损失越小越好
    # "--dataloader_drop_last", "True", # 丢弃最后一个不完全的batch
    # "--metric_for_best_model","eval_loss",  # 使用验证集上的损失作为最佳模型标准
]
finetune()
```

图 2：微调参数的设置

如图所示，展示了一些微调过后的参数，包括学习率，批次大小，以及向后传播的次数等。除此之外，我们还调整了模型能够接受的最大输入长度等。

3 评测结果

在最终提交的代码文件所生成的模型中，训练时的 loss 大小最终为 1.0557，结果接近于 1，较好。这里仅截取 loss 表格中的一部分进行展示。

Step	Training Loss
40	1.6723
80	1.6219
120	1.6265
160	1.5961
200	1.6019
240	1.6036

.....	.....
4480	1.0227
4520	1.0058
4560	1.0289
4600	1.0222
4640	1.0149
4680	1.0526
4720	1.0333
4760	1.0668
4800	1.0516
4840	1.0557

表 1：记录 loss 变化的表格

同时在进行测评时，我们微调过后的训练模型也表现良好，在与原模型的测评结果进行比较时，在大部分的测试数据集中与原模型评分相近，在某些数据集中相较于元模型的测评分数更高。

例如在 MMLU，即中学及大学的、各领域的考试题目数据集中，我们微调过后的模型在化学和物理领域的评分更高，在 HellaSwag，即选择最合适的文本续写内容，和 ARC，即常识与推理问题的数据集中，我们的模型的测评得分也要优于原模型。

该部分的评测得分等保留在提交的代码文件中，在此处不进行展示。

4 结果讨论

4.1 微调前后的模型

在本次实验中，我们专注于指令微调这一部分，强调在明确的任务指令上对原有的模型进行训练，来让模型理解和执行特定任务。

通过指令微调，模型的输出发生了改变，可以更好的执行相关指令，在我们的模型的测试结果中体现出了这一点，微调后的模型在某些领域表现更好。

4.2 对于 loss 的计算

在构造输入序列时，我们采用了 instruction + input + output 拼接在一起的方式，在这种情况下，因为是对整个序列进行的自回归训练，我们在查看对应的 loss 计算源码后，得出结论：只算 output 部分的 loss 和整个序列的 loss 的差别不大。因为自回归的训练是通过前面来预测后面，也相当于包含了对于 output 的预测。

5 总结

5.1 简要说明

在代码实现的过程中，包括设置 parser，tokenizer，model 等训练参数的过程都比较简单，这里不讲解细节。

而在实现 data_collator 函数时我们遇到了问题，最初我们打算依次设置好模型输入的 input_id，labels 和 attention_mask，并在计算 loss 时忽略掉 instruction 和 input，仅保留 output 进行计算。但在模型训练时的 loss 过高，难以收敛，进一步导致最后的测评得分下降。

```python
# Step 4: Define the data collator function
def data_collator(batch: List[Dict]):
    # Extract the 'instruction', 'input', and 'output' fields from the batch
    inputs = [
        (example['instruction'] or "") + (example['input'] or "")  # Ensure 'None' is replaced by empty string
        for example in batch
    ]
    targets = [(example['output'] or "") for example in batch]

    # Tokenize inputs and outputs (note that this is still in list form)
    model_inputs = tokenizer(inputs, max_length=data_args.max_input_length, truncation=True, padding="do_not_pad")
    labels = tokenizer(targets, max_length=data_args.max_output_length, truncation=True, padding="do_not_pad")

    # Add the labels to model_inputs (still using list operations)
    model_inputs['labels'] = [
        [-100] * len(input_seq) + label_seq
        for input_seq, label_seq in zip(model_inputs['input_ids'], labels['input_ids'])
    ]
    model_inputs['input_ids'] = [
        input_seq + label_seq
        for input_seq, label_seq in zip(model_inputs['input_ids'], labels['input_ids'])
    ]
    model_inputs['attention_mask'] = [
        input_mask + label_mask
        for input_mask, label_mask in zip(model_inputs['attention_mask'], labels['attention_mask'])
    ]

    # Pad or truncate the model_inputs to the target length
    model_inputs = pad_or_truncate_to_target_length(model_inputs, data_args.max_output_length, tokenizer)

     print("Model Input:", model_inputs)  # This will print the entire model input dictionary

    # Convert everything to tensor at the end
    model_inputs['input_ids'] = torch.tensor(model_inputs['input_ids'], dtype=torch.long)
    model_inputs['labels'] = torch.tensor(model_inputs['labels'], dtype=torch.long)
    model_inputs['attention_mask'] = torch.tensor(model_inputs['attention_mask'], dtype=torch.long)

    return model_inputs
```

图 3：最初的代码实现

为此，我们修改了相关实现，简化了模型输入的设置，不考虑用 output 进行 loss 计算，而是用整个序列进行 loss 计算。也就是最终的代码实现，如图：

```python
def data_collator(batch: List[Dict]):
    """
    batch: list of dict, each dict of the list is a sample in the dataset.
    """
    inputs = [
        (example['instruction'] or "[NA]") + (example['input'] or "[NA]") + (example['output'] or "[NA]")  # Ensure 'None' is replaced by empty string
        for example in batch
    ]
    # targets = [example['output'] if example['output'] is not None else "[NA]" for example in batch]

    # Tokenize inputs and outputs
    model_inputs = tokenizer(inputs, max_length=256, truncation=True, padding="max_length", return_tensors="pt")
    # labels = tokenizer(targets, max_length=128, truncation=True, padding="max_length", return_tensors="pt")

    # Add the labels to model_inputs
    # model_inputs['labels'] = labels['input_ids']
    model_inputs['labels'] = model_inputs['input_ids']
    # model_inputs['attention_mask'] = model_inputs['attention_mask'] + 1

    padding_value = tokenizer.pad_token_id  # or use a specific padding token value, e.g., 0
    model_inputs['labels'] = torch.where(
        model_inputs['labels'] == padding_value,
        torch.full_like(model_inputs['labels'], fill_value=-100),  # Replace padding token with ignore_index
        model_inputs['labels']
    )

    # print("Model Input:", model_inputs)  # This will print the entire model input dictionary

    # Return the processed batch, automatically padded to the same length
    return model_inputs
```

图 4：最终的代码实现

在这次的模型训练中，得到的 loss 结果接近于 1，但是测评的结果并不理想，大部分的数据集的测评结果都发生了下降。

但在查询了相关数据集的输入长度中，我们发现原来设置的最大输入长度无法满足很大一部分输入序列，导致其被截断，影响训练效果。因此，我们更改了最大输入长度，虽然增加了训练时长，但是获得了更好的训练模型。

而在其余的参数设置中，由于本次实验的要求中不希望过分关注参数选择，因此大多数的参数仅为综合了几次实验下的较优值。

5.2 结论

在本次实验中，我们小组成功的实现了对于基底模型 Qwen-2.5-0.5B 在指令数据 alpaca-cleaned 上的全量微调，微调后的模型在训练 loss 以及测评得分中均有着较好的表现。

References

None