Lab4 报告

PB21061361

模型准备

模型选择

模型使用`qwen/Qwen1.5-1.8B`模型。使用`modelscope`模块下载

```
from modelscope.hub.snapshot_download import snapshot_download

model_name = "qwen/Qwen1.5-1.8B"
model_dir = snapshot_download(model_name, cache_dir='./FineTunning/Model', revision='ma
```

数据集准备

使用`/simpleai/HC3-Chinese`数据集，其中包含有四个纬度的数据，分别是`question`，`human_answer`，`chatgpt_answer`，`source`. 现在我们想让其选择出一些回答，让模型判断这段文本是人类回答还是gpt回答。
为此，可以随机地选取一些数据，并做上标签，然后训练模型。

```
dataset = dataset["train"].shuffle(seed=42).select(range(5000))
# 取出其中的两列。
dataset = dataset.select_columns(['question', 'human_answers', 'chatgpt_answers'])
print(dataset)

def process_dataset(examples):
    inputs = []
    labels = []
    for human, gpt in zip(examples["human_answers"], examples["chatgpt_answers"]):
        if random.random() < 0.5:
            inputs.append(human)
            labels.append(1)
        else:
            inputs.append(gpt)
            labels.append(0)

    return {"inputs": inputs, "labels": labels}
```

做好标记后存储，方便后续使用。

数据预处理

使用hugging face的datasets模块，可以很方便地处理数据。在加载数据后添加prompt，然后使用tokenizer进行编码。最后删除与训练无关的列。tokenizer使用hugging face的tansformers中的AutoTokenizer模块，从下载好的模型中加载。

```
train_dataset = load_dataset('csv', data_files='./data/train.csv')
eval_dataset = load_dataset('csv', data_files='./data/eval.csv')
test_dataset = load_dataset('csv', data_files='./data/test.csv')

tokenizer = AutoTokenizer.from_pretrained(PATH, local_files_only=True)

def tokenize_function(examples):
    prompt = '以下是一段文本，判断其是人类写的还是ChatGPT写的，以1代表人类写的，0代表ChatGPT写的。'
    return tokenizer([prompt + ' <文本开始> ' + ex + '<文本结束>' for ex in examples['inpu
                      padding=True, truncation=True, max_length=512)

train_dataset = train_dataset.map(tokenize_function, batched=True).remove_columns(['inp
eval_dataset = eval_dataset.map(tokenize_function, batched=True).remove_columns(['input
test_dataset = test_dataset.map(tokenize_function, batched=True).remove_columns(['input
```

随后使用set_format函数将数据集转换为torch格式。

```
train_dataset.set_format('torch')
eval_dataset.set_format('torch')
test_dataset.set_format('torch')


train_loader = DataLoader(train_dataset['train'].select(range(500)), batch_size=4, shuf
eval_loader = DataLoader(eval_dataset['train'].select(range(100)), batch_size=4)
test_loader = DataLoader(test_dataset['train'].select(range(100)), batch_size=4)
```

由于算力限制，测试在5000个样本上运行一个epoch时，需要花费近20分钟。因此此处先暂取500个样本进行训练，100个样本进行验证，100个样本进行测试。

模型微调

混合模型

这里使用peft框架进行微调，使用hugging face的transformers模块进行模型的加载，并使用LoRA进行模型的微调。这里加载模型时，使用AutoModelForSequenceClassification并设置`num_labels=2`，表示二分类任务。

```
from transformers import (
    AutoTokenizer, AutoModelForSequenceClassification, DataCollatorWithPadding, get_sch
)
from peft import LoraConfig, TaskType, get_peft_model

model = AutoModelForSequenceClassification.from_pretrained(PATH, local_files_only=True,
lora_config = LoraConfig(
    r=8,
    lora_alpha=16,
    lora_dropout=0.1,
    target_modules=['q_proj', 'v_proj']
)

model = get_peft_model(model, lora_config)

# 优化器
optimizer = torch.optim.AdamW(model.parameters(), lr=5e-5)
num_epochs = 3
num_training_steps = num_epochs * len(train_loader)

lr_scheduler = get_scheduler(
    name="linear",
    optimizer=optimizer,
    num_warmup_steps=0,
    num_training_steps=num_training_steps
)
```

在这里定义了模型的优化器和学习率调度器。其中LoRA的参数为`r=8`，`lora_alpha=16`，`lora_dropout=0.1`。`r=8`表示插入低秩矩阵的秩，`lora_alpha=16`表示插入低秩矩阵在前向传播中的权重。

`lora_dropout=0.1`表示插入低秩矩阵的dropout率，减缓过拟合。

`target_modules`表示插入低秩矩阵的位置。此处在attention模块中的Q和V（query以及value矩阵）中插入低秩矩阵。

最后使用`get_peft_model`函数将模型和LoRA配置整合在一起，然后定义学习率调度器，使学习率线性减少。

模型训练

使用torch自定义训练过程，设置epoch=3.

```
# 训练
device = torch.device("cuda") if torch.cuda.is_available() else torch.device("cpu")
print(device)
model.to(device)
model.config.pad_token_id = tokenizer.pad_token_id

for epoch in range(num_epochs):
    model.train()
    epoch_loss = 0
    progress = tqdm.trange(len(train_loader))
    for batch in train_loader:
        batch = {k: v.to(device) for k, v in batch.items()}
        outputs = model(**batch)
        # print(outputs.loss, outputs.logits.shape)
        loss = outputs.loss
        loss.backward()

        optimizer.step()
        lr_scheduler.step()
        optimizer.zero_grad()

        epoch_loss += loss.item()

        progress.set_description(f"Epoch {epoch}")
        progress.update(1)


    print(f"Epoch {epoch} loss: {epoch_loss / len(train_loader)}")
```

遍历数据集，对模型进行训练，计算损失，然后反向传播，更新参数。最后输出每个epoch的损失。

```
    model.eval()
    metric = evaluate.load("./metrics/accuracy.py")
    eval_loss = 0
    for batch in eval_loader:
        batch = {k: v.to(device) for k, v in batch.items()}
        with torch.no_grad():
            outputs = model(**batch)
            eval_loss += outputs.loss.item()
        predictions = torch.argmax(outputs.logits, dim=-1)
        metric.add_batch(predictions=predictions, references=batch["labels"])

    acc = metric.compute()
    print(f"Epoch {epoch} eval accuracy: {acc}")
    print(f"Epoch {epoch} eval loss: {eval_loss / len(eval_loader)}")
```

此处使用hugging face的evaluate模块，计算模型的准确率。在每个epoch结束后输出**验证集**的准确率和损失。
最后，在测试集上测试。

prompt设计

初始时，添加的prompt为（后面为填充的内容，保证每个样本的长度相同，设置`max_length=512`）

```
以下是一段文本，判断其是人类写的还是ChatGPT写的，以1代表人类写的，0代表ChatGPT写的。<文本开始> <文本结
```

在此基础上，未训练前的测试集准确率55%，训练后得到的最终测试集的准确率为98%。
若去除本文开始和结束标记，再进行训练，可以得到未训练前的测试集准确率为45%，训练后得到的最终测试集的准确率为91%。

```
*************Testing untrained model************
We detected that you are passing `past_key_values` as a tuple and this is deprecated an
Test accuracy: {'accuracy': 0.45}
Test loss: 2.372902575060725
.
.
.
************Testing trained model*************
Test accuracy: {'accuracy': 0.91}
Test loss: 0.40372550616506486
```

若不指示人类为1，ChatGPT为0，而是让模型自己判断，可以得到训练前测试集准确率为53%，训练后为94%。

若完全去除prompt，让模型自己判断（被训练的模型事先已知为分类任务）。可以得到训练前测试集准确率为47%，训练后为93%。

可见prompt的影响是比较明显的，需要告知模型任务以及对应文本的起止时，模型能够更好地学习到任务。

模型评估

最后选取2000个样本训练，500个样本验证，500个样本测试，训练epoch=3，其余参数如前所述。得到的微调前测试集的准确率为51%，微调后最终测试集准确率为99.4%，得到的loss曲线为