目录

1 概述

目前，基于 Transformers 架构的大型语言模型 (LLM)，如 GPT、T5 和 BERT，已经在各种自然语言处理 (NLP) 任务中取得了 SOTA 结果。此外，还开始涉足其他领域，例如：计算机视觉 (VIT、Stable Diffusion、LayoutLM) 和音频 (Whisper、XLS-R)。将预训练好的语言模型（LM）在下游任务上进行微调已成为处理 NLP 任务的一种范式。与使用开箱即用的预训练 LLM (例如：零样本推理) 相比，在下游数据集上微调这些预训练 LLM 会带来巨大的性能提升。

但是，随着模型变得越来越大，在消费级硬件上对模型进行全部参数的微调（full fine-tuning）变得不可行。

此外，为每个下游任务独立存储和部署微调模型变得非常昂贵，因为微调模型（调整模型的所有参数）与原始预训练模型的大小相同。因此，近年来研究者们提出了各种各样的参数高效迁移学习方法（Parameter-efficient Transfer Learning），即固定住 Pretrain Language model（PLM）的大部分参数，仅调整模型的一小部分参数来达到与全部参数的微调接近的效果（调整的可以是模型自有的参数，也可以是额外加入的一些参数）。

2 相关技术介绍

2.1 预训练大模型

预训练大模型是指在庞大的语料库上进行大规模训练的深度学习模型。这些模型使用了大量的计算资源和数据，并通过自监督学习或有监督学习的方式进行训练。

预训练大模型的训练过程通常分为两个阶段：预训练和微调。在预训练阶段，模型会在大规模的未标记文本数据上进行自监督学习。它通过尝试预测文本中的下一个词或掩盖的词，来学习捕捉语言的各种特征和关系。预训练过程中使用的语料库通常是互联网上的大量文本，例如维基百科、网页内容和书籍等。

在预训练阶段完成后，模型进入微调阶段。在微调阶段，模型会使用有标签的任务特定数据集进行进一步训练。这些任务可以是文本分类、情感分析、问答系统等。通过在特定任务上进行微调，预训练大模型可以将其学习到的语言特征和上下文理解能力应用于更具体的应用领域。

预训练大模型的一个重要特点是其能够产生高质量的文本生成结果。这些模型可以用于自然语言处理任务，如机器翻译、文本摘要、对话生成等。它们在多个领域展示了强大的性能，但同时也需要大量的计算资源和数据来进行训练和推理。

2.2 指令微调

大模型的指令微调是指在预先训练好的大型模型上，通过微调少量的参数来完成特定任务的技术。该技术的作用是可以在具备大量数据和计算资源的前提下，通过迁移学习来加速特定任务的完成，同时也可以提高模型的准确性。指令微调在不断发展过程中出现了许多不同类型的微调方式，其本质是参数有效性学习（Parameter-Efficient Learning，PEL）。

在一般的计算资源条件下，大规模的模型（例如 GPT-3）很难再进行微调，因为所有的参数都需要计算梯度并进行更新，消耗时间和空间资源。为了解决这个问题，参数有效性学习被提出，其旨在确保模型效果不受太大影响的条件下尽可能地提高训练的时间和空间效率。

在参数有效性学习过程中，大模型中只需要指定或额外添加少量的可训练参数，而其余的参数全部冻结，这样可以大大提高模型的训练效率的同时，确保指标不会受到太大影响。

2.3 PEFT

大型预训练模型的训练成本非常高昂，需要庞大的计算资源和大量的数据，一般人难以承受。这也导致了一些研究人员难以重复和验证先前的研究成果。为了解决这个问题，研究人员开始研究 Parameter-Efficient Fine-Tuning (PEFT)技术。PEFT 技术旨在通过最小化微调参数的数量和计算复杂度，来提高预训练模型在新任务上的性能，从而缓解大型预训练模型的训练成

本。这样一来，即使计算资源受限，也可以利用预训练模型的知识来迅速适应新任务，实现高效的迁移学习。因此，PEFT 技术可以在提高模型效果的同时，大大缩短模型训练时间和计算成本，让更多人能够参与到深度学习研究中来。

2.3.1 Prompt-Tuning

Prompt-Tuning 是一种更近期的精调预训练语言模型的方法，重点是调整输入提示（input prompt）而非修改模型参数。这意味着预训练模型保持不变，只有输入提示被修改以适应下游的任务。通过设计和优化一组提示，可以使预训练模型执行特定任务。

Prompt-Tuning 和传统的 fine-tuning 的主要区别在于预训练模型被修改的程度。fine-tuning 修改模型的权重，而提示调整只修改模型的输入。因此，Prompt-Tuning 调整比精调的计算成本低，需要的资源和训练时间也更少。此外，Prompt-Tuning 比精调更灵活，因为它允许创建特定任务的提示，可以适应各种任务。

Prompt-Tuning 发展的两年来，有诸多工作发现，对于超过 10 亿参数量的模型来说，Prompt-Tuning 所带来的增益远远高于标准的 fine-tuning，小样本甚至是零样本的性能也能够极大地被激发出来，得益于这些模型的参数量足够大，训练过程中使用了足够多的语料，同时设计的预训练任务足够有效。最为经典的大规模语言模型则是 2020 年提出的 GPT-3，其拥有大约 1750 亿的参数，整体精调可能需要大量计算资源，而用 Prompt-Tuning 发现只需要设计合适的模板或指令甚至可以实现免参数训练的零样本学习。

2.3.2 LoRA: Low-Rank Adaptation of Large Language Models

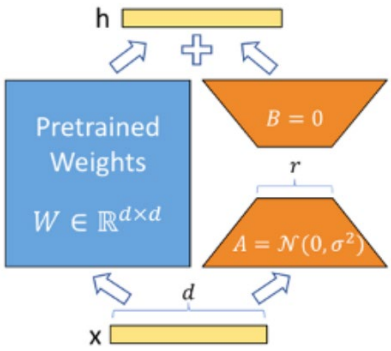


Figure 1: Our reparametrization. We only train A and B.

低秩自适应，即 LoRA，它冻结预训练的模型权重，并将可训练的秩分解矩阵注入 Transformer 架构的每一层与之并行，从而大大减少了下游任务的可训练参数数量。

如下图所示，在训练期间，预训练模型的参数 W 被冻结并且不接收梯度更新，而 LoRA 引入的参数矩阵 A 和 B 包含可训练的参数，且与参数 W 并行。这里，W 和ΔW=BA 都与相同

的输入相乘，并且它们各自的输出矢量按坐标进行求和。

我们可以将 LoRA 适用于神经网络中权重矩阵的任何子集，以减少可训练参数的数量。在 Transformer 架构中，自注意模块中有四个权重矩阵（Wq、Wk、Wv、Wo），MLP 模块中有两个。我们将 Wq（或 Wk，Wv）视为维度 dmodel×dmodel 的单个矩阵。

为了简单和参数高效，我们将 LoRA 限制为在下游任务训练时，仅通过 LoRA 引入额外矩阵ΔW=BA 去调整注意力的权重，并冻结 MLP 模块（它们不在下游任务中训练）。

实验表明，在大多数任务中仅对注意力机制中的 Wq 和 Wv 矩阵引入 LoRA 可以获得较好结果和更少的参数数量。在 RoBERTa、DeBERTa、GPT-2、GPT-3 与我们所选择的模型 GPTNeo 上，LoRA 在模型质量上的性能与全参数微调的表现不相上下或更好，而且具有较少的可训练参数、较高的训练吞吐量。并且与 Adapter 等方案不同，它没有额外的推理延迟。

3 实验过程

3.1 模型与任务选择

本实验所选择的预训练大模型为 GPT-Neo 1.3B。GPT-Neo 1.3B 是 EleutherAI 复制 GPT-3 的架构设计的 Transfomer 类模型。GPT-Neo 指的是模型的类别，1.3B 表示这个特定预训练模型的参数数量为 13 亿。

GPT-Neo 1.3B 在 Pile 上进行了训练，Pile 是 EleutherAI 为训练该模型而创建的大规模精选数据集。该模型在 Pile 上进行了 362000 步、3800 亿个 token 的训练。使用交叉熵损失将其训练为掩码自回归语言模型。通过这种方式，该模型学习了英语的内部表示，从而可用于各种各样的自然语言处理任务。

然而该模型仅经过大语料的自监督学习，虽然学习到了文本的概率分布，但没有经过微调（Fine-Tuning）。如果将该模型直接运用到特定的 NLP 任务上，效果并不理想。

本实验选择了命名实体识别、情感分类和语义相似三个 NLP 任务来对该模型进行微调。由于 GPT-Neo 1.3B 参数量较大，直接对整个模型进行微调需要较高的实验条件，因此本实验选择使用指令微调的方式进行。

3.2 命名实体识别实验

3.2.1 实验环境

- 硬件环境：
  - CPU：Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz(12 CPUs)
  - GPU：Tesla v100 (32GB) * 1
- 软件环境：
  - Python：3.8
  - PyTorch：2.0.0
  - Cuda：11.7
  - Transformers：4.30.0

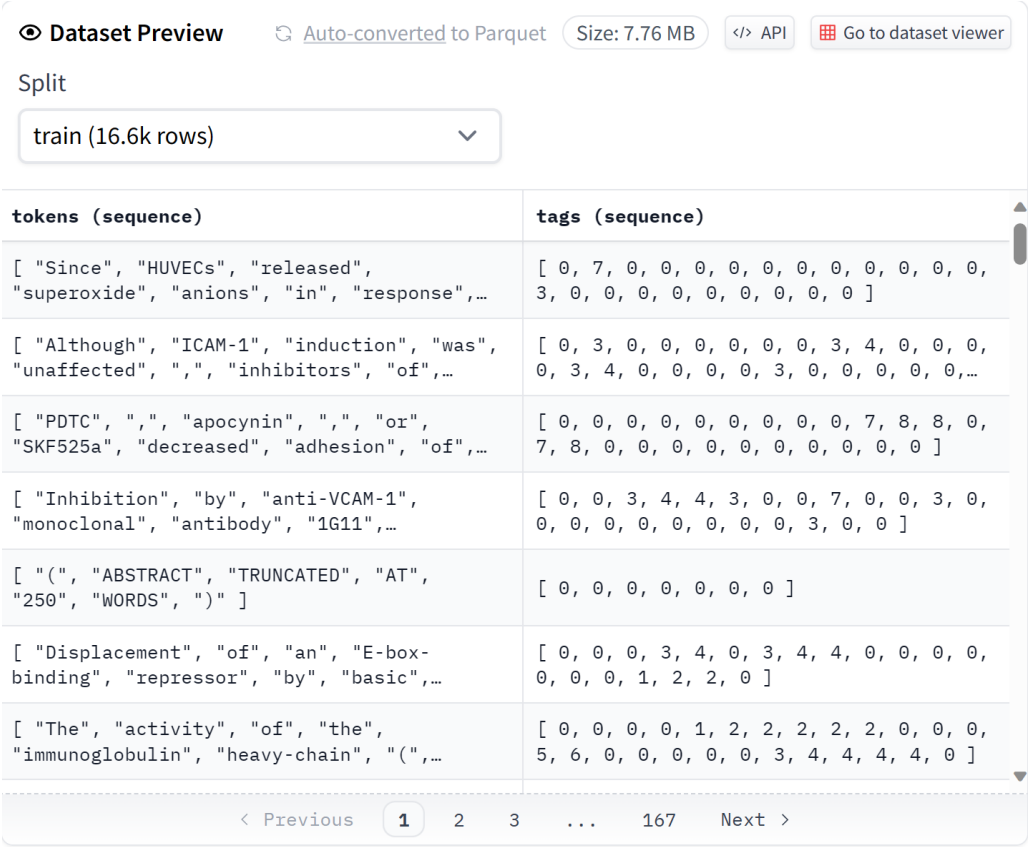
3.2.2 数据集介绍

BioNLP Shared Task(BioNLP-ST)是一个生物文本挖掘领域的国际比赛，截止到 2016 年，已经举办了 10 年，每年都吸引了来自剑桥大学、麻省理工学院等国际一流大学的科研人员参加。比赛的任务就是看哪个团队研发的算法模型能够精准智能地从文本中自动提取出复杂的生化反应网络。

BioNLP2004 数据集包含了 212 篇关于生物医学领域的文献，这些文献主要涉及基因、蛋白质、化合物以及它们之间的关系等内容。这些文献都经过了人工注释，标注了每篇文献中所有实体的位置、类型和关系，并且提供了数据集的官方评估脚本和评估结果。

该数据集的实体和关系类型包括：

实体类型：基因、蛋白质、DNA、RNA、细胞、细胞组分、组织、化合物、酶、代谢通路；关系类型：基因-基因关系、基因-蛋白质关系、蛋白质-化合物关系、酶-代谢通路关系等。具体的标号为："O": 0, "B-DNA": 1, "I-DNA": 2, "B-protein": 3, "I-protein": 4, "B-cell_type": 5, "I-cell_type": 6, "B-cell_line": 7, "I-cell_line": 8, "B-RNA": 9, "I-RNA": 10。其中标签值在标签 ID 字典中定义。每个标签前面的字母表示令牌位置: B 表示实体的第一个令牌，I 表示实体内的令牌，0 表示不属于实体的令牌。



BioNLP2004 数据集的重要性在于它提供了大量高质量的生物医学文献注释数据，并且这些数据都是专业人士经过仔细标注得出的。这使得该数据集成为了自然语言处理和生物信息学领域实体识别、关系提取等任务的重要评测基准数据集。

该数据集包含 2 列，分别是 tokens 的分词和它的命名实体识别结果的 tags。

3.2.3 实验过程

该实验采用的微调方法为 LORA（Low-Rank Adaptation of Large Language Models）微调方法，模型为 GPTNeo，实验过程大致分为：实验数据集的处理、训练微调以及模型效果评估三

大部分。

数据集处理：初始化一个标记器，并确保将其设置为 is_split_into_words = True，因为文本序列已经被分割成单词。但是，这并不意味着它已经被标记化了(即使它看起来像)，并且需要进一步将单词标记为子单词。同时，我们要写一个函数达到以下效果:

1）使用 word_ids 方法将每个标记映射到它们各自的单词；

2）通过将特殊标记设置为 -100 忽略它们；

3）标记给定实体的第一个标记。

该部分核心代码如下：

```
def tokenize_and_align_labels(examples):
    tokenized_inputs = tokenizer(examples["tokens"], truncation=True, is_split_into_words=True)

    labels = []
    for i, label in enumerate(examples[f"tags"]):
        word_ids = tokenized_inputs.word_ids(batch_index=i)
        previous_word_idx = None
        label_ids = []
        for word_idx in word_ids:
            if word_idx is None:
                label_ids.append(-100)
            elif word_idx != previous_word_idx:
                label_ids.append(label[word_idx])
            else:
                label_ids.append(-100)
            previous_word_idx = word_idx
        labels.append(label_ids)

    tokenized_inputs["labels"] = labels
    return tokenized_inputs
```

prompt 内容形如：{"tokens": ["Since","HUVECs","released","superoxide","anions","in", "response","to","TNF",",","and","H2O2","induces","VCAM-1",",","PDTC","may","act", "as", "a","radical","scavenger","."], "tags": [0, 7, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0]}

1. 训练微调：该部分通过使用 Huggingface 的 peft 库来实现，调用 PeftConfig 来对微调参数进行设置。我们可以通过打印可训练的参数来查看 PeftModel 的训练效率比完全训练基础模型高多少，通过运行我们可以得到训练参数量为 1855499，而模型总参数量为 355894283，因此训练比例为 0.5213624069370061%，这大大加快了模型训练的速度。核心代码如下：

```
model = GPTNeoForTokenClassification.from_pretrained(
    model_checkpoint, num_labels=11, id2label=id2label, label2id=label2id
)
if tokenizer.pad_token is None:
    tokenizer.add_special_tokens({'pad_token': '[PAD]'})
    model.resize_token_embeddings(len(tokenizer))

peft_config = LoraConfig(
    task_type=TaskType.TOKEN_CLS, inference_mode=False, r=16, lora_alpha=16, lora_dropout=0.1, bias="all"
)

model = get_peft_model(model, peft_config)
model.print_trainable_parameters()

training_args = TrainingArguments(
    output_dir="roberta-large-lora-token-classification",
    learning_rate=lr,
    per_device_train_batch_size=batch_size,
    per_device_eval_batch_size=batch_size,
    num_train_epochs=num_epochs,
    weight_decay=0.01,
    evaluation_strategy="epoch",
    save_strategy="epoch",
    load_best_model_at_end=True,
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_bionlp["train"],
    eval_dataset=tokenized_bionlp["validation"],
    tokenizer=tokenizer,
    data_collator=data_collator,
    compute_metrics=compute_metrics,
)

trainer.train()
```

2. 模型效果评估：本实验是一个命名实体识别任务，具体评判准则为：precision、recall、f1 和 accuracy。该部分核心代码如下：

```
def compute_metrics(p):
    predictions, labels = p
    predictions = np.argmax(predictions, axis=2)

    true_predictions = [
        [label_list[p] for (p, l) in zip(prediction, label) if l != -100]
        for prediction, label in zip(predictions, labels)
    ]
    true_labels = [
        [label_list[l] for (p, l) in zip(prediction, label) if l != -100]
        for prediction, label in zip(predictions, labels)
    ]

    results = seqeval.compute(predictions=true_predictions, references=true_labels)
    return {
        "precision": results["overall_precision"],
        "recall": results["overall_recall"],
        "f1": results["overall_f1"],
        "accuracy": results["overall_accuracy"],
    }
```

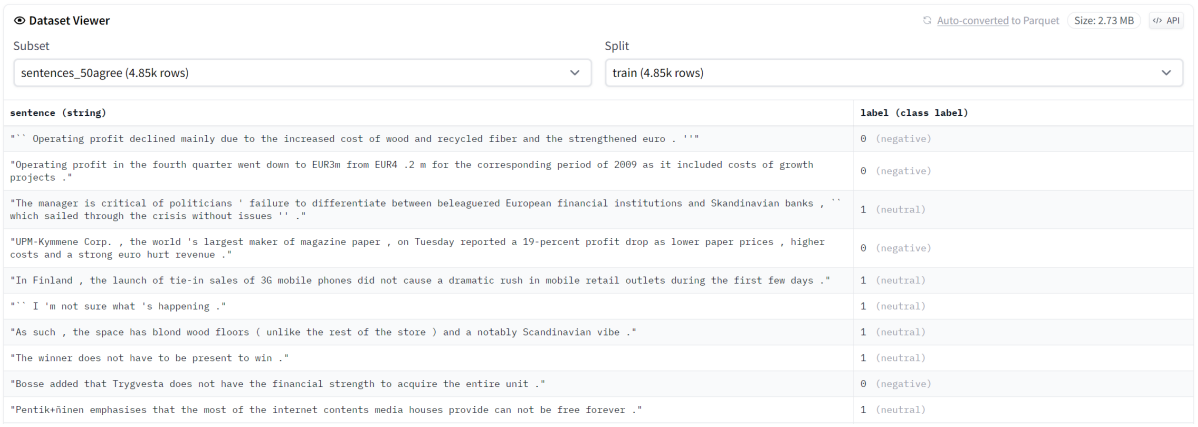
3.3 情感分类实验

3.3.1 实验环境

- 硬件环境：
  - CPU：14 vCPU Intel(R) Xeon(R) Gold 6330 CPU @ 2.00GHz
  - GPU：RTX 3090(24GB) * 1
- 软件环境：
  - Python：3.8.10
  - PyTorch：1.13.1
  - Ubuntu：20.04
  - Cuda：11.7

3.3.2 数据集介绍

使用 financial_phrasebank 数据集，包括 sentences_50agree、sentences_66agree、sentences_75agree、sentences_allagree 四部分，内容是金融短语，可以作为训练和评估替代模型的基准。用 sentences_50agree 作为训练集，sentences_66agree 作为测试集。item 的格式为 {'sentence': 'It will use the proceeds from the transaction to achieve its target .', 'label': 1, 'text_label': 'neutral'} 。情感有三种：positive、negative、neutral。


Dataset Viewer

Auto-converted to Parquet | Size: 2.73 MB | API

Subset: sentences_50agree (4.85k rows)

Split: train (4.85k rows)

sentence (string)	label (class label)
"`` Operating profit declined mainly due to the increased cost of wood and recycled fiber and the strengthened euro . ''"	0 (negative)
"Operating profit in the fourth quarter went down to EUR3m from EUR4 .2 m for the corresponding period of 2009 as it included costs of growth projects ."	0 (negative)
"The manager is critical of politicians ' failure to differentiate between beleaguered European financial institutions and Skandinavian banks , `` which sailed through the crisis without issues '' ."	1 (neutral)
"UPM-Kymmene Corp. , the world 's largest maker of magazine paper , on Tuesday reported a 19-percent profit drop as lower paper prices , higher costs and a strong euro hurt revenue ."	0 (negative)
"In Finland , the launch of tie-in sales of 3G mobile phones did not cause a dramatic rush in mobile retail outlets during the first few days ."	1 (neutral)
"`` I 'm not sure what 's happening ."	1 (neutral)
"As such , the space has blond wood floors (unlike the rest of the store) and a notably Scandinavian vibe ."	1 (neutral)
"The winner does not have to be present to win ."	1 (neutral)
"Bosse added that Trygvesta does not have the financial strength to acquire the entire unit ."	0 (negative)
"Pentik+ñinen emphasises that the most of the internet contents media houses provide can not be free forever ."	1 (neutral)

3.4.3 实验过程

该实验使用 Prompt-Tuning 的方式来对模型进行微调。大致分类数据集处理、微调训练和效果评估三个部分。

1. 数据集处理：根据 financial_phrasebank 数据集，把模型的输入 prompt 处理为 sentence + label 的形式并通过 tokenizer 转化为 token 对应的索引，从而方便模型进行训练和学习。核心代码如下：

```python
text_column = "sentence"
label_column = "text_label"


dataset = load_dataset("financial_phrasebank", "sentences_50agree")
dataset = dataset["train"].train_test_split(test_size=0.1)
dataset["validation"] = dataset["test"]
del dataset["test"]
# {"Tweet text": "@HMRCcustomers No this is my first job", "ID": 0, "Label": 2}

classes = dataset["train"].features["label"].names
print(classes)
dataset = dataset.map(
    lambda x: {"text_label": [classes[label] for label in x["label"]]},
    batched=True,
    num_proc=1,
)
print(dataset['train'][0])
# {"Tweet text": "@HMRCcustomers No this is my first job", "ID": 0, "Label": 2, "text_label": "no complaint"}

tokenizer = AutoTokenizer.from_pretrained(model_name_or_path)
if tokenizer.pad_token_id is None:
    tokenizer.pad_token_id = tokenizer.eos_token_id
target_max_length = max([len(tokenizer(class_label)["input_ids"]) for class_label in classes])
print(target_max_length)
# 1

def preprocess_function(examples):
    batch_size = len(examples[text_column])
    inputs = [f"{text_column} : {x} Label : " for x in examples[text_column]]
    targets = [str(x) for x in examples[label_column]]
    model_inputs = tokenizer(inputs)
    labels = tokenizer(targets)
    for i in range(batch_size):
        sample_input_ids = model_inputs["input_ids"][i]
        label_input_ids = labels["input_ids"][i] + [tokenizer.pad_token_id]
        # print(i, sample_input_ids, label_input_ids)
        model_inputs["input_ids"][i] = sample_input_ids + label_input_ids
        labels["input_ids"][i] = [-100] * len(sample_input_ids) + label_input_ids
        model_inputs["attention_mask"][i] = [1] * len(model_inputs["input_ids"][i])
    # print(model_inputs)
    for i in range(batch_size):
        sample_input_ids = model_inputs["input_ids"][i]
        label_input_ids = labels["input_ids"][i]
        model_inputs["input_ids"][i] = [tokenizer.pad_token_id] * (
            max_length - len(sample_input_ids)
        ) + sample_input_ids
        model_inputs["attention_mask"][i] = [0] * (max_length - len(sample_input_ids)) + model_inputs[
            "attention_mask"
        ][i]
        labels["input_ids"][i] = [-100] * (max_length - len(sample_input_ids)) + label_input_ids
        model_inputs["input_ids"][i] = torch.tensor(model_inputs["input_ids"][i][:max_length])
        model_inputs["attention_mask"][i] = torch.tensor(model_inputs["attention_mask"][i][:max_length])
        labels["input_ids"][i] = torch.tensor(labels["input_ids"][i][:max_length])
    model_inputs["labels"] = labels["input_ids"]
    return model_inputs


processed_datasets = dataset.map(
    preprocess_function,
    batched=True,
    num_proc=14,
    remove_columns=dataset["train"].column_names,
    load_from_cache_file=False,
    desc="Running tokenizer on dataset",
)

train_dataset = processed_datasets["train"]
eval_dataset = processed_datasets["validation"]
```

prompt 内容形如：{'sentence': 'It will use the proceeds from the transaction to achieve its target .', 'label': 1, 'text_label': 'neutral'}

2. 微调训练：该部分通过使用 Huggingface 的 peft 库来实现。调用 PeftConfig 来对微调参数进行设置，可以得到训练参数量为 16384，而模型总参数量为 1315592192，因此训练比例为 0.0012453707235136 89%，这大大加快了模型训练的速度。核心代码如下：

```python
train_dataset = processed_datasets["train"]
eval_dataset = processed_datasets["validation"]


train_dataloader = DataLoader(
    train_dataset, shuffle=True, collate_fn=default_data_collator, batch_size=batch_size, pin_memory=True
)
eval_dataloader = DataLoader(eval_dataset, collate_fn=default_data_collator, batch_size=batch_size, pin_memory=True)

model = AutoModelForCausalLM.from_pretrained(model_name_or_path)
model = get_peft_model(model, peft_config)
# print("asd")
print(model.print_trainable_parameters())
# print("asdasd")
# "trainable params: 16384 || all params: 1315592192 || trainable%: 0.001245370723513689"

optimizer = torch.optim.AdamW(model.parameters(), lr=lr)
lr_scheduler = get_linear_schedule_with_warmup(
    optimizer=optimizer,
    num_warmup_steps=0,
    num_training_steps=(len(train_dataloader) * num_epochs),
)

model = model.to(device)

print('==========================[Training]==================================')
for epoch in range(num_epochs):
    model.train()
    total_loss = 0
    for step, batch in enumerate(tqdm(train_dataloader)):
        # print(len(batch))
        batch = {k: v.to(device) for k, v in batch.items()}
        outputs = model(**batch)
        loss = outputs.loss
        total_loss += loss.detach().float()
        loss.backward()
        optimizer.step()
        lr_scheduler.step()
        optimizer.zero_grad()

    model.eval()
    eval_loss = 0
    eval_preds = []
    for step, batch in enumerate(tqdm(eval_dataloader)):
        # print(batch)
        batch = {k: v.to(device) for k, v in batch.items()}
        with torch.no_grad():
            outputs = model(**batch)
        loss = outputs.loss
        eval_loss += loss.detach().float()
        eval_preds.extend(
            tokenizer.batch_decode(torch.argmax(outputs.logits, -1).detach().cpu().numpy(), skip_special_tokens=True)
        )

    eval_epoch_loss = eval_loss / len(eval_dataloader)
    eval_ppl = torch.exp(eval_epoch_loss)
    train_epoch_loss = total_loss / len(train_dataloader)
    train_ppl = torch.exp(train_epoch_loss)
    print(f"{epoch=}: {train_ppl=} {train_epoch_loss=} {eval_ppl=} {eval_epoch_loss=}")

peft_model_id="prompt-tuning-model-{}epoch".format(num_epochs)
model.save_pretrained(peft_model_id)
tokenizer.save_pretrained(peft_model_id)

print('==========================[Model Saved]==================================')
```

3. 效果评估：本实验会对微调前后模型均进行评估。由于该任务可视为二分类任务，评估指标可简单设置为正确率（ACC）。在微调前的模型上进行评估时，由于该模型未经过微调，直接输入 sentence + label 形式的 prompt，模型大概率无法理解。因此在评估时，会在 prompt 最前面加上“Classify if the sentence is neutral, negative, positive:”的内容来进行提示。对于模型生成的内容，会截取关键词"Label:"后的内容是否与测试集中对应 label 吻合来计

算正确率。评估过程会对不同微调程度（训练的 epoch 数量不同）的模型进行评估。核心代码如下：

```
print('====================[Evaluate]=========================')

total_num = 0
hit_num = 0
for item in test_dataset:
    total_num += 1
    # print(item)
    print('[No.{} test case] '.format(total_num), end=' ')
    true_label = item['text_label']
    # print(true_label)

    # print('<True Label>: {}'.format(true_label), end=' ')
    input_ids = torch.tensor(item['input_ids']).view(-1,len(item['input_ids'])).to(device)
    attention_mask = torch.tensor(item['attention_mask']).view(-1,len(item['attention_mask'])).to(device)
    with torch.no_grad():
        outputs = model.generate(
            input_ids=input_ids, attention_mask=attention_mask, max_new_tokens=10, eos_token_id=tokenizer.eos_token_id, pad_token_id=tokenizer.eos_token_id,
        )
        # print(outputs)
        # print(type(true_label))
        # true_label=tokenizer.batch_decode([0 if x == -100 else x for x in true_label], skip_special_tokens=True)
        # if "positive" in true_label:
        #     true_label="positive"
        # elif "negative" in true_label:
        #     true_label="negative"
        # else:
        #     true_label="neutral"
        decode_outputs = tokenizer.batch_decode(outputs, skip_special_tokens=True)
        # print(decode_outputs)
        # outputs = re.search(pattern=r'Label : (.*)$', string=decode_outputs[0])
        # print("output:")
        # print(decode_outputs[0])
        if decode_outputs[0].find("\"Label\" : positive")!=-1:
            outputs="positive"
        elif decode_outputs[0].find("\"Label\" : negative")!=-1:
            outputs="negative"
        elif decode_outputs[0].find("\"Label\" : neutral")!=-1:
            outputs="neutral"
            # outputs="neutral"
            # print("begin try")
            # print(outputs)
            # outputs = outputs.group().split(':')
            # print("ho")
            # outputs = outputs[1].strip(' ')
            # outputs=outputs.replace("Q","")

        else:
            # print(decode_outputs, end=' ')
            print('[Wrong ...]')
            continue
        print('<Predict Label>: {}'.format(outputs), end=' ')
        print('<True Label>: {}'.format(true_label), end=' ')
        if outputs == true_label:
            hit_num += 1
            print('[Right !!!]')
        else:
            print('[Wrong ...]')


print('====================[Result]=========================')
print('Test total num: {}'.format(total_num))
print('Test hit num: {}'.format(hit_num))
print('Test hit accuracy: {}'.format(hit_num/total_num))
```

3.4 语义相似实验

3.4.1 实验环境

- 硬件环境：
  - CPU：14 vCPU Intel(R) Xeon(R) Gold 6330 CPU @ 2.00GHz
  - GPU：RTX 3090(24GB) * 1
- 软件环境：
  - Python：3.8
  - PyTorch：2.0.0
  - Ubuntu：20.04

Cuda：11.8

3.4.2 数据集介绍

本实验数据集使用 GLUE（General Language Understanding Evaluation），出自《GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding》，是纽约大学、华盛顿大学等机构创建的一个多任务的自然语言理解基准和分析平台。

GLUE 包含九项 NLU 任务，语言均为英语。GLUE 九项任务涉及到自然语言推断、文本蕴含、情感分析、语义相似等多个任务。大体上可以分为三类任务，分别是单句任务，相似性和释义任务。具体分别是 CoLA、SST-2、MRPC、STS-B、QQP、MNLI、QNLI、RTE、WNLI。像 BERT、XLNet、RoBERTa、ERINE、T5 等知名模型都会在此基准上进行测试。

Corpus	|Train|	|Test|	Task	Metrics	Domain
			Single-Sentence Tasks		
CoLA	8.5k	**1k**	acceptability	Matthews corr.	misc.
SST-2	67k	1.8k	sentiment	acc.	movie reviews
			Similarity and Paraphrase Tasks		
MRPC	3.7k	1.7k	paraphrase	acc./F1	news
STS-B	7k	1.4k	sentence similarity	Pearson/Spearman corr.	misc.
QQP	364k	**391k**	paraphrase	acc./F1	social QA questions
			Inference Tasks		
MNLI	393k	**20k**	NLI	matched acc./mismatched acc.	misc.
QNLI	105k	5.4k	QA/NLI	acc.	Wikipedia
RTE	2.5k	3k	NLI	acc.	news, Wikipedia
WNLI	634	**146**	coreference/NLI	acc.	fiction books

本实验使用的是 GLUE 数据集中的 MRPC（Microsoft Research Paraphrase Corpus）。该数据集是从在线新闻源中自动提取的句子对语料库，并带有人工注释以判断句子对中的句子是否在语义上等同，可以用来做语义相似任务。

sentence1 (string)	sentence2 (string)	label (class label)	idx (int32)
"PCCW 's chief operating officer , Mike Butcher , and Alex Arena , the chief financial officer , will report directly to Mr So ."	"Current Chief Operating Officer Mike Butcher and Group Chief Financial Officer Alex Arena will report to So ."	1 (equivalent)	0
"The world 's two largest automakers said their U.S. sales declined more than predicted last month as a late summer sales frenzy caused…	"Domestic sales at both GM and No. 2 Ford Motor Co. declined more than predicted as a late summer sales frenzy prompted a larger-than-…	1 (equivalent)	1
"According to the federal Centers for Disease Control and Prevention (news - web sites) , there were 19 reported cases of measles in th…	"The Centers for Disease Control and Prevention said there were 19 reported cases of measles in the United States in 2002 ."	1 (equivalent)	2
"A tropical storm rapidly developed in the Gulf of Mexico Sunday and was expected to hit somewhere along the Texas or Louisiana coasts by…	"A tropical storm rapidly developed in the Gulf of Mexico on Sunday and could have hurricane-force winds when it hits land somewhere…	0 (not_equivalent)	3
"The company didn 't detail the costs of the replacement and repairs ."	"But company officials expect the costs of the replacement work to run into the millions of dollars ."	0 (not_equivalent)	4
"The settling companies would also assign their possible claims against the underwriters to the investor plaintiffs , he added ."	"Under the agreement , the settling companies will also assign their potential claims against the underwriters to the investors , he adde…	1 (equivalent)	5
"Air Commodore Quaife said the Hornets remained on three-minute alert throughout the operation ."	"Air Commodore John Quaife said the security operation was unprecedented ."	0 (not_equivalent)	6
"A Washington County man may have the countys first human case of West Nile virus , the health department said Friday ."	"The countys first and only human case of West Nile this year was confirmed by health officials on Sept . 8 ."	1 (equivalent)	7
"Moseley and a senior aide delivered their summary assessments to about 300 American and allied military officers on Thursday ."	"General Moseley and a senior aide presented their assessments at an internal briefing for American and allied military officers at Nelli…	1 (equivalent)	8
"The broader Standard & Poor 's 500 Index < .SPX > was 0.46 points lower , or 0.05 percent , at 997.02 ."	"The technology-laced Nasdaq Composite Index .IXIC was up 7.42 points , or 0.45 percent , at 1,653.44 ."	0 (not_equivalent)	9
"Consumers would still have to get a descrambling security card from their cable operator to plug into the set ."	"To watch pay television , consumers would insert into the set a security card provided by their cable service ."	1 (equivalent)	10

该数据集包含 4 列，分别为标号、句子 1、句子 2 和标签。标签是来自人工的标注，判断句子 1 和句子 2 是否等同。标签为 1 表示等同，为 0 则相反。

3.4.3 实验过程

该实验使用 Prompt-Tuning 的方式来对模型进行微调。大致分类数据集处理、微调训练和效果评估三个部分。

1. 数据集处理：根据 GLUE 中 MRPC 数据集的特性，把模型的输入 prompt 处理为 sentence1 + sentence2 + label 的形式并通过 tokenizer 转化为 token 对应的索引，从而方便模型进行训练和学习。核心代码如下：

```
def preprocess_function(examples):
    # print(examples)
    batch_size = len(examples[text1_column])
    inputs = [f"\"Sentence1\" : {examples[text1_column][index]} \"Sentence2\" : {examples[text2_column][index]} \"Label\" : " for index in range(batch_size)]
    # print(inputs)
    targets = [str(x) for x in examples[label_column]]
    # print(targets)
    model_inputs = tokenizer(inputs)
    labels = tokenizer(targets)
    # print(labels)
    for i in range(batch_size):
        sample_input_ids = model_inputs["input_ids"][i]
        label_input_ids = labels["input_ids"][i] + [tokenizer.pad_token_id]
        # print(i, sample_input_ids, label_input_ids)
        model_inputs["input_ids"][i] = sample_input_ids + label_input_ids
        labels["input_ids"][i] = [-100] * len(sample_input_ids) + label_input_ids
        model_inputs["attention_mask"][i] = [1] * len(model_inputs["input_ids"][i])
    # print(model_inputs)
    for i in range(batch_size):
        sample_input_ids = model_inputs["input_ids"][i]
        label_input_ids = labels["input_ids"][i]
        model_inputs["input_ids"][i] = [tokenizer.pad_token_id] * (
            max_length - len(sample_input_ids)
        ) + sample_input_ids
        model_inputs["attention_mask"][i] = [0] * (max_length - len(sample_input_ids)) + model_inputs[
            "attention_mask"
        ][i]
        labels["input_ids"][i] = [-100] * (max_length - len(sample_input_ids)) + label_input_ids
        model_inputs["input_ids"][i] = torch.tensor(model_inputs["input_ids"][i][:max_length])
        model_inputs["attention_mask"][i] = torch.tensor(model_inputs["attention_mask"][i][:max_length])
        labels["input_ids"][i] = torch.tensor(labels["input_ids"][i][:max_length])
    model_inputs["labels"] = labels["input_ids"]
    return model_inputs
```

prompt 内容形如：["Sentence1" : PCCW's chief operating officer, Mike Butcher, and Alex Arena, the chief financial officer, will report directly to Mr So. "Sentence2" : Current Chief Operating Officer Mike Butcher and Group Chief Financial Officer Alex Arena will report to So. "Label" :]

2. 微调训练：该部分通过使用 Huggingface 的 peft 库来实现。调用 PeftConfig 来对微调参数进行设置，可以得到训练参数量为 16384，而模型总参数量为 1315592192，因此训练比例为 0.001245370723513689%，这大大加快了模型训练的速度。核心代码如下：

```
model = AutoModelForCausalLM.from_pretrained(model_name_or_path)
model = get_peft_model(model, peft_config)
print(model.print_trainable_parameters())

optimizer = torch.optim.AdamW(model.parameters(), lr=lr)
lr_scheduler = get_linear_schedule_with_warmup(
    optimizer=optimizer,
    num_warmup_steps=0,
    num_training_steps=(len(train_dataloader) * num_epochs),
)

model = model.to(device)
```

```python
for epoch in range(num_epochs):
    model.train()
    total_loss = 0
    for step, batch in enumerate(tqdm(train_dataloader)):
        batch = {k: v.to(device) for k, v in batch.items()}
        outputs = model(**batch)
        loss = outputs.loss
        total_loss += loss.detach().float()
        loss.backward()
        optimizer.step()
        lr_scheduler.step()
        optimizer.zero_grad()

    model.eval()
    eval_loss = 0
    eval_preds = []
    for step, batch in enumerate(tqdm(eval_dataloader)):
        batch = {k: v.to(device) for k, v in batch.items()}
        with torch.no_grad():
            outputs = model(**batch)
        loss = outputs.loss
        eval_loss += loss.detach().float()
        eval_preds.extend(
            tokenizer.batch_decode(torch.argmax(outputs.logits, -1).detach().cpu().numpy(), skip_special_tokens=True)
        )

    eval_epoch_loss = eval_loss / len(eval_dataloader)
    eval_ppl = torch.exp(eval_epoch_loss)
    train_epoch_loss = total_loss / len(train_dataloader)
    train_ppl = torch.exp(train_epoch_loss)
    print(f"{epoch=}: {train_ppl=} {train_epoch_loss=} {eval_ppl=} {eval_epoch_loss=}")
```

3. 效果评估：本实验会对微调前后模型均进行评估。由于该任务可视为二分类任务，评估指标可简单设置为正确率（ACC）。在微调前的模型上进行评估时，由于该模型未经过微调，直接输入 sentence1 + sentence2 + label 形式的 prompt，模型大概率无法理解。因此在评估时，会在 prompt 最前面加上“Classify if the two sentences are equivalent or not:”的内容来进行提示。对于模型生成的内容，会截取关键词"Label:"后的内容是否与测试集中对应 label 吻合来计算正确率。评估过程会对不同微调程度（训练的 epoch 数量不同）的模型进行评估。核心代码如下：

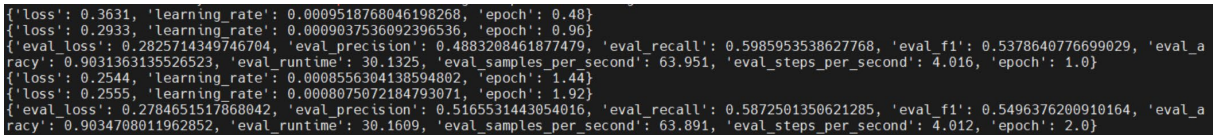
```python
total_num = 0
hit_num = 0
for item in test_dataset:
    total_num += 1
    print('[No.{} test case] '.format(total_num), end=' ')
    true_label = item['text_label']

    print('<True Label>: {}'.format(true_label), end=' ')
    input_ids = torch.tensor(item['input_ids']).view(-1,len(item['input_ids'])).to(device)
    attention_mask = torch.tensor(item['attention_mask']).view(-1,len(item['attention_mask'])).to(device)
    with torch.no_grad():
        outputs = model.generate(
            input_ids=input_ids, attention_mask=attention_mask, max_new_tokens=10, eos_token_id=tokenizer.eos_token_id, pad_token_id=tokenizer.eos_token_id,
        )
        decode_outputs = tokenizer.batch_decode(outputs, skip_special_tokens=True)
        outputs = re.search(pattern=r'Label" :(.*)$', string=decode_outputs[0])
        try:
            outputs = outputs.group().split(':')
            outputs = outputs[1].strip(' ')
            print('<Predict Label>: {}'.format(outputs), end=' ')
            if outputs == true_label:
                hit_num += 1
                print('[Right !!!]')
            else:
                print('[Wrong ...]')
        except:
            print(decode_outputs, end=' ')
            print('[Wrong ...]')
            continue


print('====================[Result]==========================')
print('Test total num: {}'.format(total_num))
print('Test hit num: {}'.format(hit_num))
print('Test hit accuracy: {}'.format(hit_num/total_num))
```

4 实验结果

4.1 命名实体识别实验

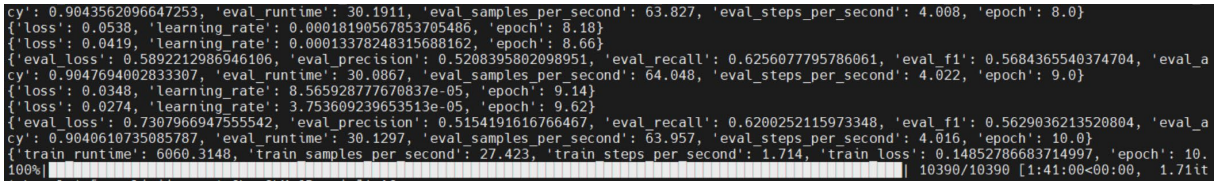
测试集总数 3856。

4.1.1 微调前实验结果



微调前，该模型对于命名实体识别任务表现较差，误差 loss 值较高，模型对于实体识别任务识别效果较差。

4.1.2 微调后实验结果

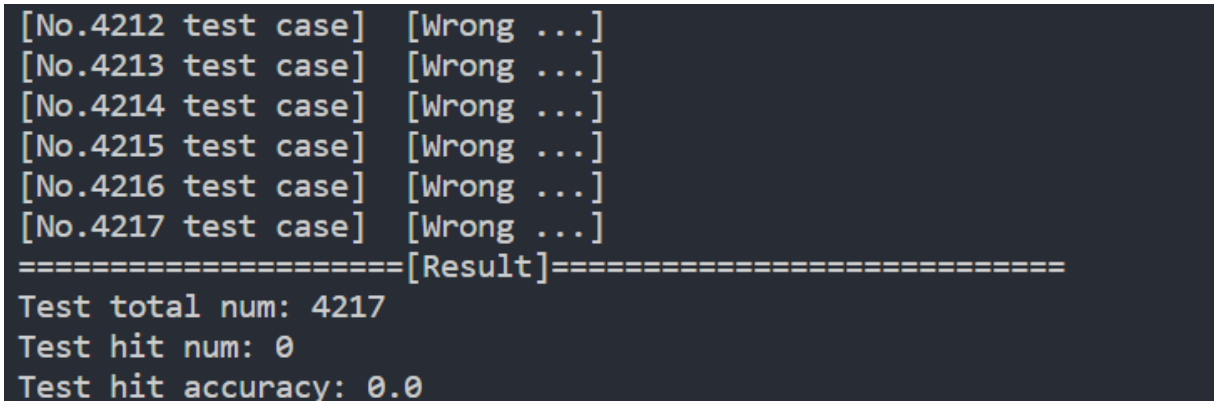


经过了 10 个 epoch 的模型微调后，误差 loss 显著降低，模型效果提升较大。

4.2 情感分类实验

测试集总数 4217，评估标准正确率（ACC）。

4.2.1 微调前实验结果



将 sentences_66agree 数据集直接输入到模型，会发现模型根本不会按照要求对句子情感进行分析，只是生成一些无意义的词句，模型的能力几乎为 0。

统计结果得到微调前模型预测正确率为 0%。

4.2.2 微调后实验结果

经过一个 epoch 之后，模型的效果有大幅度的提升，而且模型的输出大部分都是符合要求的，效果很好。

```
[No.4206 test case]  [Wrong ...]
[No.4207 test case]  <Predict Label>: neutral <True Label>: negative [Wrong ...]
[No.4208 test case]  <Predict Label>: neutral <True Label>: negative [Wrong ...]
[No.4209 test case]  [Wrong ...]
[No.4210 test case]  <Predict Label>: neutral <True Label>: negative [Wrong ...]
[No.4211 test case]  <Predict Label>: negative <True Label>: negative [Right !!!]
[No.4212 test case]  [Wrong ...]
[No.4213 test case]  [Wrong ...]
[No.4214 test case]  [Wrong ...]
[No.4215 test case]  <Predict Label>: neutral <True Label>: neutral [Right !!!]
[No.4216 test case]  <Predict Label>: neutral <True Label>: negative [Wrong ...]
[No.4217 test case]  <Predict Label>: neutral <True Label>: negative [Wrong ...]
====================[Result]==========================
Test total num: 4217
Test hit num: 2729
Test hit accuracy: 0.6471425183779939
```

经过 5 个 epoch 之后，模型的效果有更大的提升。

```
[No.4208 test case]  <Predict Label>: negative <True Label>: negative [Right !!!]
[No.4209 test case]  [Wrong ...]
[No.4210 test case]  <Predict Label>: negative <True Label>: negative [Right !!!]
[No.4211 test case]  <Predict Label>: negative <True Label>: negative [Right !!!]
[No.4212 test case]  <Predict Label>: negative <True Label>: negative [Right !!!]
[No.4213 test case]  <Predict Label>: negative <True Label>: negative [Right !!!]
[No.4214 test case]  <Predict Label>: negative <True Label>: negative [Right !!!]
[No.4215 test case]  <Predict Label>: negative <True Label>: neutral [Wrong ...]
[No.4216 test case]  <Predict Label>: negative <True Label>: negative [Right !!!]
[No.4217 test case]  <Predict Label>: negative <True Label>: negative [Right !!!]
====================[Result]==========================
Test total num: 4217
Test hit num: 3508
Test hit accuracy: 0.8318709983400522
```

本实验一共进行了三次，分别是 1epoch、5epoch、10epoch，结果如下：

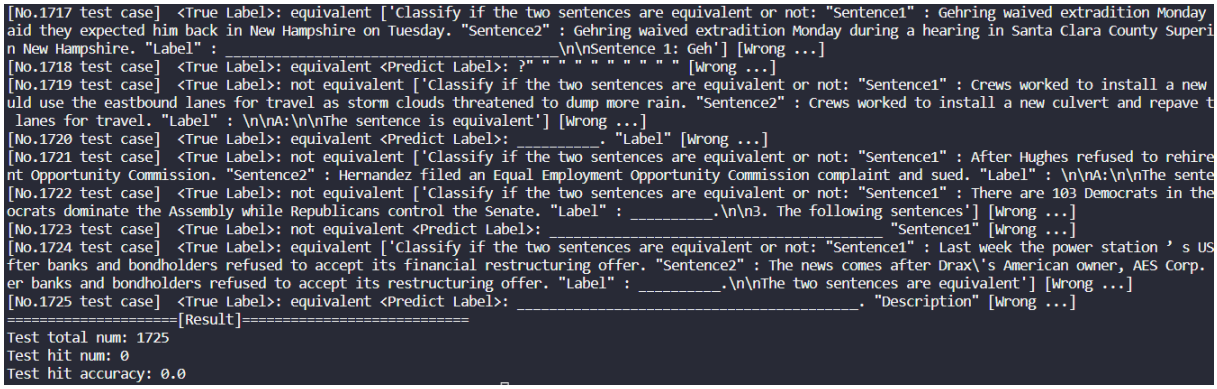
epoch_num	1	5	10
正确率	64.7%	83.2%	73.6%

从中可以看出，适当的微调可以大幅提高模型的效果，但是距离 SOTA 还有很大距离，可微调的参数量有限，所以模型的效果上限仍然较低。

4.3 语义相似实验

测试集总数 1725，评估标准正确率（ACC）。

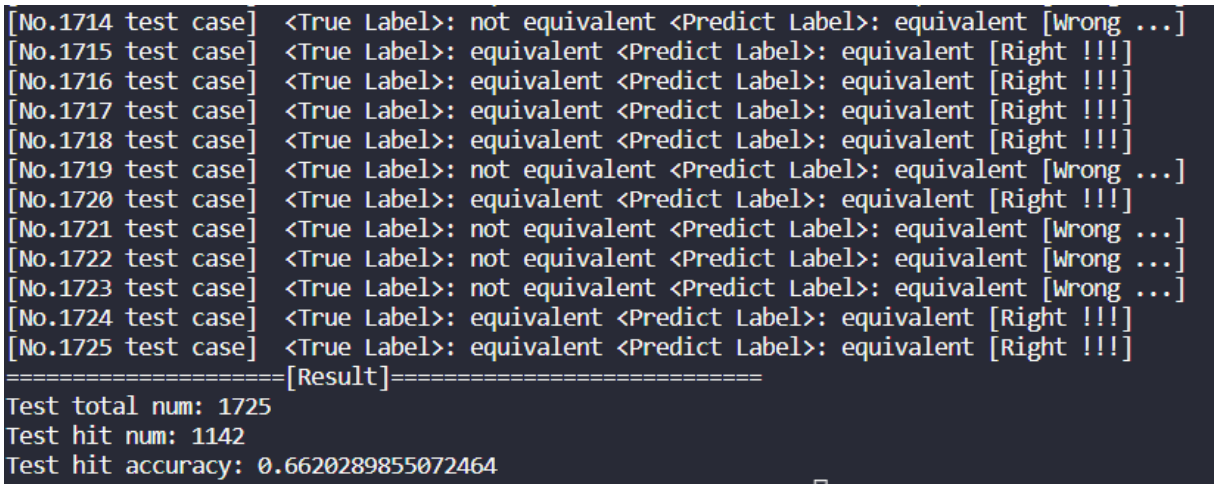
4.3.1 微调前实验结果



从图中可以看出，微调前的模型生成内容非常混乱，显然并未理解 prompt 中的内容，模型并不清楚实验想要它完成什么任务，仅仅是机械性地进行概率生成下一个 token，不明白 label 后应该输出些什么内容。

因此，最终统计结果：预测正确率：0%

4.3.2 微调后实验结果



上图为经过 1 个 epoch 微调后的效果，可以看出模型的输出已经出现明显的格式，可以正常的通过正则表达式匹配到所需的预测结果。此时模型的输出格式如下：

["Sentence1" : PCCW's chief operating officer, Mike Butcher, and Alex Arena, the chief financial officer, will report directly to Mr So. "Sentence2" : Current Chief Operating Officer Mike Butcher and Group Chief Financial Officer Alex Arena will report to So. "Label" : equivalent]

精准地在"Label"后生成了所需要的标签（无论正确与否），表明模型已经可以依照所需来进行任务回答。但经过 1 个 epoch 微调的正确率并不理想，测试集正确率为 66.2%。

为了尽可能达到更好的效果，本实验累计进行了 5 个 epoch，10 个 epoch，20 个 epoch 的多次微调实验，结果如下：

epoch_num	1	5	10	20
正确率	66.2%	71.3%	72.6%	72.4%

可以看出，在经过多个 epoch 的微调后，模型正确率在逐渐上升，但趋势逐渐减缓。然而从 GLUE benchmark 的网站上可以看到其他语言模型在 MPRC 任务上的表现可以达到 90%以上的正确率，本实验模型与之有不小的差距。其可能的原因如下：

- 1.3B 参数量的预训练模型对语言建模程度有限
- 在微调过程中，所设置的可微调参数量过小，很多参数未参与微调从而限制了模型效果
- 未对学习率等超参数进行较为详细的实验，该微调未必到达理想情况下的局部最优解

5 总结

本小组在本次实验中尝试了多种微调方式来对预训练模型 GPT-Neo 进行微调，包括 Prompt-Tuning 和 LoRA，且进行了多个 NLP 任务、多个数据集的训练与测试。从结果上看，预训练模型在不经过微调时，对人类所要求的任务理解程度不高，生成内容混乱，即使忽略掉正确率，其任务完成效果依然很差。而只需经过 1 个 epoch、几千样本量、微量参数的微调后，其效果就可以急剧提升，达到较高水准。说明预训练大模型本身对于文本语言是有一定建模和理解的，只是在微调前不能按照人类要求较好的完成所需任务。

此外，经过此次实验，对 NLP 相关模型的训练和测试有了不小的了解，对 PEFT 的理解也有了较大提升。很明显地感受到了 PEFT 在微调较大模型上的优势，只需训练非常少的参数就可以达到不错的效果，大大降低了大模型使用上的硬件环境要求和门槛。

此外，由于时间有限，本实验仍有很多地方存在不足：对模型微调的超参数没有进行尝试，对模型的训练程度也存在不足等等。因此目前模型在多个 NLP 任务上的表现并没有非常出色，仍有不小的进步空间，后面会继续学习 NLP 相关知识，把相关思维方式运用到各自的研究方向上去。整体而言，本次实验收货满满！