10215501442报告

所使用的库以及实验过程中遇到的问题

好吧知道这次作业我才知道可以通过requirements.txt**的** `conda list --export > requirements.txt` **相关命令一次输出环境的包和版本以及通过** `conda create --name test --file requirements.txt` **安装环境，之前的这个文件都是自己打的，如果给助教带来不便，实在抱歉**

实验中主要遇到的问题基本都是些输入输出变张量的类型以及大小问题

本次实验中使用了`argparse`，可以在命令行中使用对应命令设置模型和参数，详情可见README.md

数据预处理

```
# 读取 CSV 文件
csv_path = "/kaggle/input/data111/train.csv"  # 替换为实际的文件路径
df = pd.read_csv(csv_path)

# 划分数据集
train_df, val_df = train_test_split(df, test_size=0.2, random_state=42)

# 创建数据集和 DataLoader，设置max_seq_length
max_seq_length = 150
train_dataset = CustomDataset(train_df, max_seq_length)
val_dataset = CustomDataset(val_df, max_seq_length)

train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False)

# 初始化模型和优化器
input_dim = len(set(df['description'].str.cat().split()))  # 根据数据集中的词汇数初始化输入维度
output_dim = len(set(df['diagnosis'].str.cat().split()))  # 根据数据集中的词汇数初始化输出维度
```

数据划分时选择了80：20的训练集和验证集比例

对于每条序列要设置其为固定维度，维度不足的要填充，不然会遇到训练时维度不一致的问题，导致报错，这个功能是通过 `CustomDataset类` 实现的。

CustomDataset类这个类是用于处理文本序列数据，以便将输入序列和输出序列传递给 Transformer 模型进行训练，

1. **数据加载：** 通过 `__init__` 方法，接收一个包含描述（description）和诊断（diagnosis）的 DataFrame (`df`) 以及最大序列长度 (`max_seq_length`)。这个类的实例在创建时就会加载整个数据集。
2. **数据预处理：** 通过 `__getitem__` 方法，在每次迭代中处理单个样本。它将描述和诊断的文本序列转换为对应的整数序列，并确保它们具有相同的长度。对于超过最大序列长度的部分，使用零进行填充。
3. **返回数据长度：** 通过 `__len__` 方法，定义了数据集的长度，即数据集中样本的总数。

模型选择

本次实验的主要工作选择了T5模型同时作为encoder和decoder，选择的原因是它推出的时间比较晚，应该能比较简单地取得较好的成绩。

模型的定义

```
class T5Model(nn.Module):
    def __init__(self, model_name):
        super(T5Model, self).__init__()
        self.t5 = T5ForConditionalGeneration.from_pretrained(model_name)

    def forward(self, input_ids, labels=None):
        if labels is not None:
            # During training, return the loss
            outputs = self.t5(input_ids, labels=labels)
            return outputs.loss
        else:
            # During inference, return the logits
            return self.t5(input_ids).logits
```

在 `__init__` 方法中，通过调用 `T5ForConditionalGeneration.from_pretrained(model_name)` 创建了一个 T5 模型的实例。 `model_name` 参数指定了要加载的预训练模型的名称，例如，"t5-small"、"t5-base" 等。这个实例被存储在 `self.t5` 中。

`forward` 方法定义了前向传播的过程。这个方法接受输入 `input_ids` 和可选的 `labels` ，并根据是否提供了 `labels` 参数来执行不同的操作。

- 如果提供了 `labels` ，则表示模型处于训练阶段，这时调用 `self.t5(input_ids, labels=labels)` 执行前向传播，并返回模型的损失值 (`outputs.loss`)。这是因为 T5 模型在

训练时需要计算损失值，用于反向传播和参数更新。

- 如果没有提供 `labels`，则表示模型处于推断阶段，这时调用 `self.t5(input_ids).logits` 返回模型的输出 logits。在这种情况下，模型的输出可以用于生成文本或进行其他任务的推断。

模型训练

```
# 初始化模型和优化器
model = T5Model('t5-small')
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

# 训练模型
num_epochs = 5
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

model.to(device)

for epoch in range(num_epochs):
    model.train()
    total_loss = 0

    for input_ids, target_ids in tqdm(train_loader, desc=f"Epoch {epoch + 1}/{num_epoc
hs}"):
        input_ids, target_ids = input_ids.to(device), target_ids.to(device)
        optimizer.zero_grad()

        # 计算损失
        loss = model(input_ids, labels=target_ids)
        loss.backward()
        optimizer.step()
        total_loss += loss.item()

    average_loss = total_loss / len(train_loader)
    print(f"Epoch {epoch + 1}/{num_epochs}, Loss: {average_loss}")
```

训练时使用的模型为小型的T5，使用了Adam，定义了一个 Adam 优化器，用于更新模型的参数。学习率为 0.001，但是这个值可以在命令行中选择。

其实本来想试下T5-base的，但是由于显存的限制，不能进行测试，另外在运行时，可以明显感觉即使是T5-small，也比transformer等模型要大不少，训练所需时间要多得多。

验证和评估

验证和评估的代码

```
# 在验证集上评估模型
model.eval()
total_val_loss = 0
predictions = []
references = []

with torch.no_grad():
    for input_ids, target_ids in tqdm(val_loader, desc="Validation"):
        input_ids, target_ids = input_ids.to(device), target_ids.to(device)

        # 生成序列而不是返回 logits
        generated_ids = model.t5.generate(input_ids, max_length=50, num_beams=2, lengt
h_penalty=0.6)

        # 计算损失
        loss = model(input_ids, labels=target_ids)
        total_val_loss += loss.item()

        # 记录模型的预测和真实参考
        predictions.extend(generated_ids.tolist())
        references.extend(target_ids.tolist())


average_val_loss = total_val_loss / len(val_loader)
print(f"Validation Loss: {average_val_loss}")

# 设置nltk的SmoothingFunction
smooth_func = SmoothingFunction().method1

# 将预测和参考转换为字符串
predictions_str = [' '.join(map(str, seq)) for seq in predictions]
references_str = [' '.join(map(str, seq)) for seq in references]

# 计算BLEU-4得分
bleu_scores = [sentence_bleu([reference], prediction, smoothing_function=smooth_func)
for
               reference, prediction in zip(references_str, predictions_str)]

# 输出平均BLEU-4得分
average_bleu_score = sum(bleu_scores) / len(bleu_scores)
print(f"Average BLEU-4 Score: {average_bleu_score}")

# 定义ROUGE评估函数
def rouge_evaluation(predictions, references):
    scorer = rouge_scorer.RougeScorer(['rouge1', 'rouge2', 'rougeL'], use_stemmer=True
)
    rouge1_scores = []
    rouge2_scores = []
    rougeL_scores = []

    for pred, ref in zip(predictions, references):
        pred_text = ' '.join(map(str, pred)).strip()
        ref_text = ' '.join(map(str, ref)).strip()
```

```
        scores = scorer.score(pred_text, ref_text)

        rouge1_scores.append(scores['rouge1'].fmeasure)
        rouge2_scores.append(scores['rouge2'].fmeasure)
        rougeL_scores.append(scores['rougeL'].fmeasure)

    average_rouge1 = sum(rouge1_scores) / len(rouge1_scores)
    average_rouge2 = sum(rouge2_scores) / len(rouge2_scores)
    average_rougeL = sum(rougeL_scores) / len(rougeL_scores)

    return average_rouge1, average_rouge2, average_rougeL

# 使用ROUGE评估
rouge1, rouge2, rougeL = rouge_evaluation(predictions, references)

print(f"ROUGE-1 Score: {rouge1}")
print(f"ROUGE-2 Score: {rouge2}")
print(f"ROUGE-L Score: {rougeL}")
```

在预测时，记录模型的预测和真实参考，这些序列将用于后续的 BLEU 和 ROUGE 评估。

BLEU的评估：使用 NLTK 的 `sentence_bleu` 函数计算每个样本的 BLEU-4 分数。

ROUGE的评估：使用了rouge_score的 `rouge_scorer.RougeScorer`

实验结果

对于不同的epoch以及T5-small，最终训练的结果如下

	1个epoch	3个epoch	5个epoch
最终训练损失	0.972	0.578	0.527
验证损失	0.655	0.536	0.511
BLEU	0.339	0.379	0.400
ROUGE-1	0.525	0.559	0.536
ROUGE-2	0.427	0.468	0.437
ROUGE-L	0.492	0.524	0.494

可以发现仅一个epoch损失就收敛到了1之下，不过随着epoch的增大，损失下降的就不是很多了，BLEU和ROUGE的值也基本都在增大，不过还是比我自己做的要好很多，自己之前手写的一个版本，3个epoch之后损失仍然有十几，最终因为无法收敛，被弃用了。

拓展探究

除了T5模型之外，我还尝试了其他的模型，都是同时作为encoder和decoder，有RNN，LSTM，GRU，Transformer，但是运行代码时，他们的BLEU和ROUGE评估分数都集中在0.83左右，令我感到十分疑惑，怀疑代码写错了，因此没有算在正文中，不过在主函数仍然提供了这一选项，可以使用这些模型训练。在主函数中使用if和elif写了总共五个分支的判断，每个分支中是对应的模型的代码，不过删去了BLEU和ROUGE分数的计算代码。除此之外，模型类型定义处的定义以及训练过程略有不同。

下面是这些模型1个epoch训练出的数据，没有给出评估分数，因为我认为无法使用

	训练损失	**验证损失**
RNN	0.154	2.626
LSTM	0.615	2.15
GRU	0.215	2.79
Transformer	1.376	1.325