人工智能第四次实验报告

杨帆-10215501416

实验描述

任务定义

目标：本实验旨在利用算法从长文本中提取关键信息，生成简短、凝练的摘要，以应用于医学领域。具体而言，我们要从病情描述中提取关键信息，生成诊断报告。

数据集

- **训练数据**：包含18000条样本，保存在`train.csv`中，可自行划分验证集。
- **测试数据**：包含2000条样本，保存在`test.csv`中。所有数据已脱敏。

任务需求

1. **构建模型**：建立seq2seq模型，选择Encoder为Transformer。
2. **评估指标**：可选BLEU-4、ROUGE、CIDEr等。

实验内容

- 选择一种Encoder-Decoder结构，我们选择了Transformer。
- 使用bleu1，bleu2 ,bleu3, bleu4, rouge-1进行结果分析。

实验环境

代码语言：Python3.10

需要安装的Python库：torch,pandas,scikit-learn,nltk,argparse,tqdm

编程工具：Pycharm

模型描述

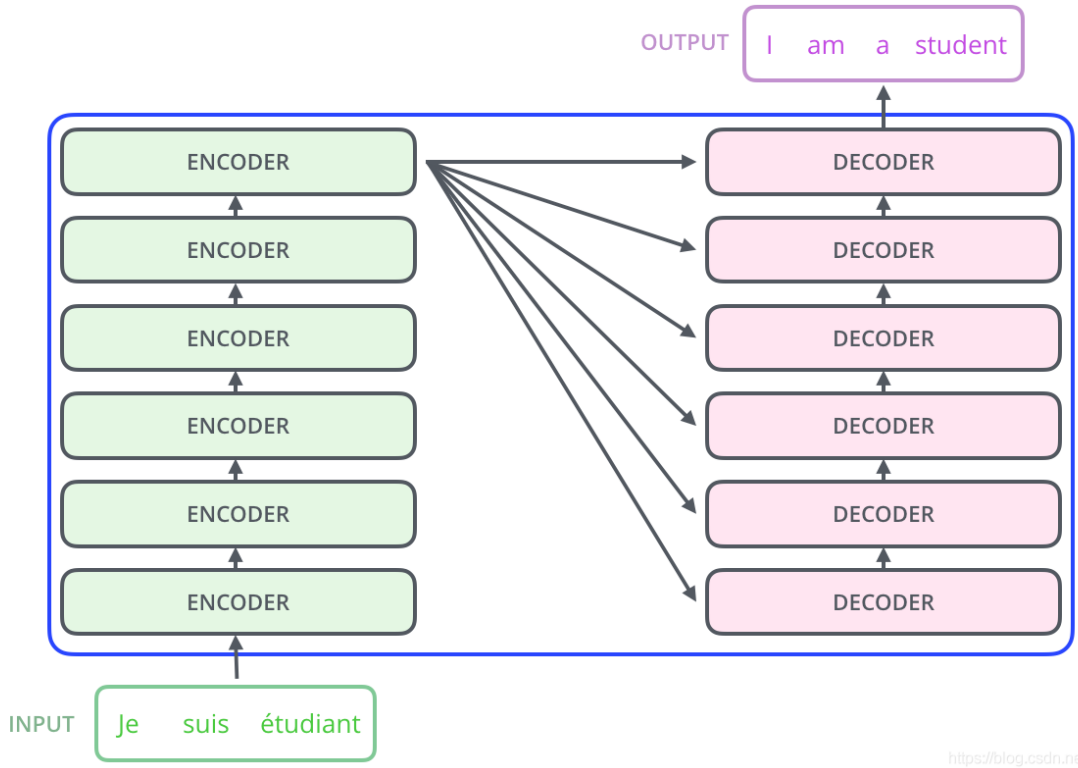
Transformer是一种深度学习模型架构，最初由Vaswani等人于2017年提出，用于自然语言处理任务，特别是机器翻译。相较于之前的循环神经网络（RNN）和长短时记忆网络（LSTM）等传统序列模型，Transformer采用了全新的结构，引入了自注意力机制（Self-Attention）。

以下是Transformer模型的基本概述：

1. **整体结构：** Transformer模型主要由两部分组成，即Encoder和Decoder。这两部分分别负责处理输入序列和生成输出序列。每个部分都包含多个相同结构的层，通常是6个。
2. **自注意力机制（Self-Attention）：** 这是Transformer的核心创新之一。自注意力机制允许模型在处理输入序列时对不同位置的信息分配不同的注意力权重。这使得模型能够更好地捕捉序列中的长距离依赖关系，而无需像传统的固定窗口大小的卷积或循环结构那样受限。
3. **多头注意力机制：** 为了进一步提高模型的表达能力，自注意力机制被扩展为多头注意力机制。这意味着模型可以同时关注输入序列中的不同部分，以获得更全面的信息。
4. **前馈神经网络：** 每个注意力子层后面都连接着一个前馈神经网络。这个网络独立地处理每个位置的特征，增强了模型对局部模式的建模能力。

5. **残差连接和层归一化：** 每个子层的输出都通过残差连接与其输入相加，然后进行层归一化。这有助于缓解训练中的梯度消失问题，提高了模型的训练稳定性。
6. **位置编码：** 由于Transformer没有显式的顺序信息处理，需要引入位置编码来为模型提供关于词的位置信息。
7. **Encoder-Decoder结构：** 在机器翻译等序列转换任务中，模型需要使用两个Transformer部分，一个用于编码源语言序列，另一个用于生成目标语言序列。

总体而言，Transformer模型的创新之处在于引入了自注意力机制，使其在处理序列数据时更加灵活和高效。这一结构的成功应用不仅限于自然语言处理领域，还包括计算机视觉等多个领域。



实验步骤

1 数据预处理(`preprocess_data` 函数)

- 读取CSV文件，该文件包含两列：`description`（描述）和 `diagnosis`（诊断）。使用 `lambda` 函数和 `split()` 方法将字符串中的数字分割，并转换为整数列表。
- 将描述和诊断的字符串转换为整数列表，每个数字代表一个标记（token）。
- 对描述和诊断的序列进行填充或截断，使它们的长度达到 `max_seq_length`。如果长度超过了设定的最大长度，就进行截断；如果长度不足，则用零填充。
- 将数据集划分为训练集和验证集（80%训练，20%验证）。
- 构建词汇表（vocabulary），将每个标记映射到一个唯一的整数，并创建标记到索引和索引到标记的映射。并训练集中的描述和诊断序列，将所有数字组成一个总的列表 `all_tokens`。然后基于这个列表创建了一个词汇表 `vocab`，包含了所有唯一的数字。接着建立了从数字到索引的映射 `token2idx` 和从索引到数字的映射 `idx2token`，这些映射用于将数字序列转换为索引序列以便模型处理。

```python
def preprocess_data(df, max_seq_length=100):
    df['description'] = df['description'].apply(lambda x: [int(num) for num in x.split()])
    df['diagnosis'] = df['diagnosis'].apply(lambda x: [int(num) for num in x.split()])
    df['description'] = df['description'].apply(lambda x: x[:max_seq_length] + [0] * (max_seq_length - len(x)))
    df['diagnosis'] = df['diagnosis'].apply(lambda x: x[:max_seq_length] + [0] * (max_seq_length - len(x)))

    train_df, val_df = train_test_split(*arrays: df, test_size=0.2, random_state=42)

    all_tokens = [token for seq in train_df['description'].tolist() + train_df['diagnosis'].tolist() for token in seq]
    vocab = set(all_tokens)

    token2idx = {token: idx + 1 for idx, token in enumerate(vocab)}  # 0 is reserved for padding
    idx2token = {idx + 1: token for idx, token in enumerate(vocab)}

    train_df['description'] = train_df['description'].apply(lambda x: [token2idx[token] for token in x])
    train_df['diagnosis'] = train_df['diagnosis'].apply(lambda x: [token2idx[token] for token in x])

    val_df['description'] = val_df['description'].apply(lambda x: [token2idx.get(token, 0) for token in x])
    val_df['diagnosis'] = val_df['diagnosis'].apply(lambda x: [token2idx.get(token, 0) for token in x])

    return train_df, val_df, token2idx, idx2token
```

2. 模型构建（`TransformerSeq2Seq` 类）

- 使用 PyTorch 定义了一个包含嵌入层、Transformer模块和全连接层的序列到序列模型。
- 嵌入层用于将标记索引映射到密集的词嵌入向量。
- Transformer模块包括多头自注意力机制，用于捕捉输入序列的全局依赖关系。
- 输出层是一个全连接层，用于将Transformer的输出映射到词汇表上的标记。

```python
class TransformerSeq2Seq(nn.Module):
    def __init__(self, input_dim, output_dim, d_model=256, nhead=4, num_encoder_layers=2, num_decoder_layers=2):
        super(TransformerSeq2Seq, self).__init__()

        self.embedding = nn.Embedding(input_dim, d_model)
        self.transformer = nn.Transformer(
            d_model=d_model,
            nhead=nhead,
            num_encoder_layers=num_encoder_layers,
            num_decoder_layers=num_decoder_layers,
            batch_first=True
        )
        self.fc_out = nn.Linear(d_model, output_dim)

    def forward(self, src, tgt):
        src = self.embedding(src)
        tgt = self.embedding(tgt)

        output = self.transformer(src, tgt)
        output = self.fc_out(output)

        return output
```

3. 模型训练（`train_model` 函数）

- 使用交叉熵损失函数（`nn.CrossEntropyLoss`）作为损失函数。
- 使用Adam优化器进行模型参数的优化。
- 进行多个epochs的训练循环。在每个epoch中：
    - 将模型设置为训练模式（`model.train()`）。
    - 遍历训练数据集的批次，对模型进行前向传播、计算损失、反向传播和参数更新。
    - 计算并输出训练集的平均损失。
    - 在训练过程中，生成预测序列并计算BLEU和ROUGE分数，以及准确度。
    - 打印每个epoch的训练结果。

- 模型参数保存： 在整个训练过程结束后，保存训练好的Transformer模型参数到文件中（'transformer_model.pth'）。

```
def train_model(model, train_loader, val_loader, criterion, optimizer, num_epochs=10):
    for epoch in range(num_epochs):
        model.train()
        train_loss = 0
        predictions = []
        references = []
        for src, tgt in tqdm(train_loader, desc=f'Epoch {epoch + 1}/{num_epochs}'):
            src, tgt_input = src[:, :-1], tgt[:, :-1]
            tgt_expected = tgt[:, 1:]
            optimizer.zero_grad()
            output = model(src, tgt_input)
            loss = criterion(output.reshape(-1, output.size(-1)), tgt_expected.reshape(-1))
            loss.backward()
            optimizer.step()

            train_loss += loss.item()
            with torch.no_grad():
                predictions.extend(output.argmax(dim=-1).cpu().numpy().tolist())
                references.extend(tgt_expected.cpu().numpy().tolist())
        bleu1, bleu2, bleu3, bleu4 = compute_bleu(predictions, references)
        rouge_1 = compute_rouge_n(predictions, references, n=1)
        avg_train_loss = train_loss / len(train_loader)
        accuracy = compute_accuracy(predictions, references)
        print(f'Epoch {epoch + 1}/{num_epochs}, Avg Train Loss: {avg_train_loss}')
        print(f'BLEU-1: {bleu1}, BLEU-2: {bleu2}, BLEU-3: {bleu3}, BLEU-4: {bleu4}')
        print(f'ROUGE-1: {rouge_1}')
        print(f'Accuracy: {accuracy}')

    torch.save(model.state_dict(), f: 'transformer_model.pth')
```

4. 主函数（`main` 函数）

- 通过命令行参数解析器（`argparse`）获取文件路径、最大序列长度、训练时的epochs数量和批次大小等超参数。
- 读取CSV文件并进行数据预处理。
- 将处理后的数据转换为PyTorch张量，并创建训练和验证集的数据加载器。
- 定义Transformer模型、优化器和损失函数。
- 调用 `train_model` 函数进行模型训练。

```
def main(args):
    df = pd.read_csv(args.file_path)

    train_df, val_df, token2idx, idx2token = preprocess_data(df, max_seq_length=args.max_seq_length)

    X_train, y_train = torch.tensor(list(train_df['description'])), torch.tensor(list(train_df['diagnosis']))
    X_val, y_val = torch.tensor(list(val_df['description'])), torch.tensor(list(val_df['diagnosis']))
    train_dataset = TensorDataset( *tensors: X_train, y_train)
    val_dataset = TensorDataset( *tensors: X_val, y_val)
    train_loader = DataLoader(train_dataset, batch_size=args.batch_size, shuffle=True)
    val_loader = DataLoader(val_dataset, batch_size=args.batch_size, shuffle=False)
    input_dim = len(token2idx) + 1
    output_dim = len(token2idx) + 1
    model = TransformerSeq2Seq(input_dim, output_dim)
    optimizer = optim.Adam(model.parameters(), lr=0.001)
    criterion = nn.CrossEntropyLoss(ignore_index=0)

    train_model(model, train_loader, val_loader, criterion, optimizer, num_epochs=args.epochs)
```

实验结果展示与分析

评价指标选择

1 BLEU

BLEU（Bilingual Evaluation Understudy）是一种用于自动评估机器翻译质量的评价指标。它最初由Papineni等人于2002年提出，并在机器翻译领域得到广泛应用。以下是对BLEU评价指标的主要描述：

1. **目标：** BLEU的主要目标是衡量机器生成的翻译与人工参考翻译之间的相似程度，以评估翻译系统的性能。
2. **计算方法：** BLEU的计算基于n-gram的匹配。它比较机器生成的翻译和参考翻译中的n-gram（通常是1-gram到4-gram）的重叠情况。BLEU的计算公式为：

$$BLEU = BP \times exp(\sum_1^N 1/N log(precision)_n)$$

   其中，{BP} 是一种长度惩罚项（Brevity Penalty），N 是选择的n-gram的最大长度，{precision}n是机器生成的翻译中与参考翻译重叠的n-gram的精确度。

3. **重叠精确度（Precision）：** 对于每个n-gram，BLEU计算机器翻译和参考翻译中相同n-gram的数量，并将其除以机器翻译中该n-gram的总数量。这可以看作是对翻译的准确性的一种度量。
4. **长度惩罚项（Brevity Penalty）：** 由于机器翻译系统可能倾向于生成较短的翻译，导致BLEU的不公平比较，所以引入了长度惩罚项。

```
4 usages
def compute_bleu_ngram(predictions, references, n):
    bleu_scores = [sentence_bleu( references: [ref], pred, weights=(1 / n,) * (n - 1) + (1.0 / n,),
                                  smoothing_function=SmoothingFunction().method1) for ref, pred in
                   zip(references, predictions)]
    return sum(bleu_scores) / (len(bleu_scores))
```

2 ROUGE

ROUGE是一组用于评价自动生成文本质量的指标，主要应用于自动文摘的评估。ROUGE的核心思想是通过比较生成文本与参考文本之间的重叠来评估生成文本的质量，其主要关注召回率（Recall）。

ROUGE包括多个指标，其中最常用的有Rouge-N、Rouge-L、Rouge-W。下面是对它们的简要描述和计算方式：

1. **Rouge-N（N-gram召回率）**：

   描述：Rouge-N通过比较生成文本和参考文本中N-gram（N个连续单词）的重叠情况来评估质量。

   计算方式：

$$Rouge\text{-}N = \frac{\text{匹配的N-gram数}}{\text{参考文本中的总N-gram数}}$$

2. **Rouge-L（最长公共子序列召回率）**：

   描述：Rouge-L使用最长公共子序列（Longest Common Subsequence，LCS）来度量生成文本和参考文本的相似性。

   计算方式：

$$Rouge\text{-}L = \frac{\text{LCS的长度}}{\text{参考文本的总长度}}$$

3. **Rouge-W（权重召回率）**：

描述：Rouge-W考虑了单词的权重，通过给予关键词更高的权重来评估生成文本的质量。

计算方式：

$$Rouge\text{-}W = \frac{\text{匹配的关键词权重总和}}{\text{参考文本中的关键词权重总和}}$$

```python
def compute_rouge_n(predictions, references, n=1):
    rouge_scores = []
    for ref, pred in zip(references, predictions):
        ref_ngrams = set(ngrams(ref, n))
        pred_ngrams = set(ngrams(pred, n))
        common_ngrams = ref_ngrams.intersection(pred_ngrams)
        rouge_scores.append(len(common_ngrams) / max(len(ref_ngrams), 1))
    return sum(rouge_scores) / len(rouge_scores)
```

模型运行结果

```
C:\Users\Lenovo\Desktop\作业\人工智能\project4\venv\Scripts\python.exe C:\Users\Lenovo\Desktop\作业\人工智能\project4\src\1.py
Epoch 1/10: 100%|██████████| 450/450 [14:23<00:00,  1.92s/it]
Epoch 1/10, Avg Train Loss: 0.6625734814008077
BLEU-1: 0.30184647399925124, BLEU-2: 0.2913738681158462, BLEU-3: 0.28456205754818603, BLEU-4: 0.2796900280778939
ROUGE-1: 0.6603085789015978
Accuracy: 0.8851437991021325
Epoch 2/10: 100%|██████████| 450/450 [10:58<00:00,  1.46s/it]
Epoch 2/10, Avg Train Loss: 0.43189646336767407
BLEU-1: 0.3135332491582467, BLEU-2: 0.3025257823299702, BLEU-3: 0.29495202330114784, BLEU-4: 0.2893771406118337
ROUGE-1: 0.7938676645584476
Accuracy: 0.9160823512906846
Epoch 3/10: 100%|██████████| 450/450 [11:01<00:00,  1.47s/it]
Epoch 3/10, Avg Train Loss: 0.3842060664958424
BLEU-1: 0.31521347736625355, BLEU-2: 0.3044262199545845, BLEU-3: 0.2968775892117155, BLEU-4: 0.2912658610294667
ROUGE-1: 0.8132109980013422
Accuracy: 0.9211054994388328
Epoch 4/10: 100%|██████████| 450/450 [11:04<00:00,  1.48s/it]
Epoch 4/10, Avg Train Loss: 0.35118453744384975
BLEU-1: 0.31609801720912756, BLEU-2: 0.30554895656707703, BLEU-3: 0.29805713081673696, BLEU-4: 0.2924484208436023
ROUGE-1: 0.8231970381947776
Accuracy: 0.9243090628507296
Epoch 5/10: 100%|██████████| 450/450 [11:31<00:00,  1.54s/it]
Epoch 5/10, Avg Train Loss: 0.32536164058579337
BLEU-1: 0.3166650299289169, BLEU-2: 0.3063203782938478, BLEU-3: 0.29884410083516655, BLEU-4: 0.2932220265676483
ROUGE-1: 0.8301462850570844
Accuracy: 0.9268076599326599
Epoch 6/10: 100%|██████████| 450/450 [11:07<00:00,  1.48s/it]
Epoch 6/10, Avg Train Loss: 0.30059114671415754
BLEU-1: 0.3172135708941254, BLEU-2: 0.30701979219863795, BLEU-3: 0.2995761449457459, BLEU-4: 0.2939481745750092
ROUGE-1: 0.8366406854884432
Accuracy: 0.9290523288439955
Epoch 7/10: 100%|██████████| 450/450 [11:17<00:00,  1.51s/it]
Epoch 7/10, Avg Train Loss: 0.27858326272832024
BLEU-1: 0.31771815375982043, BLEU-2: 0.3076466236008099, BLEU-3: 0.30020234907927856, BLEU-4: 0.29454516705425954
ROUGE-1: 0.8435547087072699
Accuracy: 0.9311258417508418
Epoch 8/10: 100%|██████████| 450/450 [10:22<00:00,  1.38s/it]
Epoch 8/10, Avg Train Loss: 0.259038374390867
```

```
Epoch 9/10: 100%|██████████| 450/450 [09:58<00:00,  1.33s/it]
Epoch 9/10, Avg Train Loss: 0.24228207882907654
BLEU-1: 0.318719603441825, BLEU-2: 0.3090401469042644, BLEU-3: 0.30171522723532473, BLEU-4: 0.29610111746224993
ROUGE-1: 0.8559340930335906
Accuracy: 0.9355695847362514
Epoch 10/10: 100%|██████████| 450/450 [09:53<00:00,  1.32s/it]
Epoch 10/10, Avg Train Loss: 0.22783208885126643
BLEU-1: 0.3191376730265622, BLEU-2: 0.3097149864585586, BLEU-3: 0.30249527375034313, BLEU-4: 0.2969284532651597
ROUGE-1: 0.8608010757304131
Accuracy: 0.9376437991021325
```

可以发现：

1. **训练损失下降**：随着训练的进行，平均训练损失（Avg Train Loss）逐渐降低。这表明模型在训练数据上逐步提高了预测的准确性。
2. **BLEU分数**：BLEU-1到BLEU-4分别表示N-gram的BLEU分数，其中BLEU-1代表单个词的匹配程度，BLEU-4考虑了四元语法的匹配情况。这些分数大致在0.30左右，随着训练次数增加略有改善，但改进不是非常显著。这可能表明模型在生成文本时在不同长度和语法结构上存在一定程度的不准确性。
3. **ROUGE分数**：ROUGE-1是一种评估生成文本与参考文本之间相似度的指标。随着训练次数增加，ROUGE-1分数逐步提高，这表示模型生成的文本与参考文本的重叠度在逐步增加。
4. **准确率（Accuracy）**：在评估生成文本与参考文本匹配时的准确率大约在0.88到0.93之间，这表明模型生成的文本中有相当一部分与参考文本匹配，但并非完全匹配。

遇到的困难

1 运行一次模型时间过久（一个epoch就需要20分钟），导致在debug和调节参数的时候十分不方便，但是没有找到一个能很好解决问题的办法，在调试时只能降低epoch数以及减少d_model的数目来减少运行时间。

2 以及在计算一些评估指标时候导入Rouge库和CIDEr时候失败，要么下载不了要么就是已经下载好了仍然导入失败，最后使用nltk库中的函数计算出了结果

学到的知识和心得

- Seq2Seq（Sequence-to-Sequence）模型是一种用于处理序列数据的框架，适用于将一个序列映射到另一个序列的任务，如文本摘要。
- Transformer是一种强大的深度学习模型，特别适用于处理序列数据。它使用自注意力机制来捕捉序列中的依赖关系，适用于长距离依赖性的任务。
- 学会了如何选择合适的模型结构，以及在实验中如何进行训练和调参，以达到更好的性能。