中山大学计算机学院

人工智能

本科生实验报告

（2023 学年春季学期）

课程名称：Artificial Intelligence

教学班级	**信计系统班**	专业（方向）	**信息与计算科学**
学号	**22336059**	姓名	**董胜凡**

一、 实验题目

NLI（自然语言推理）

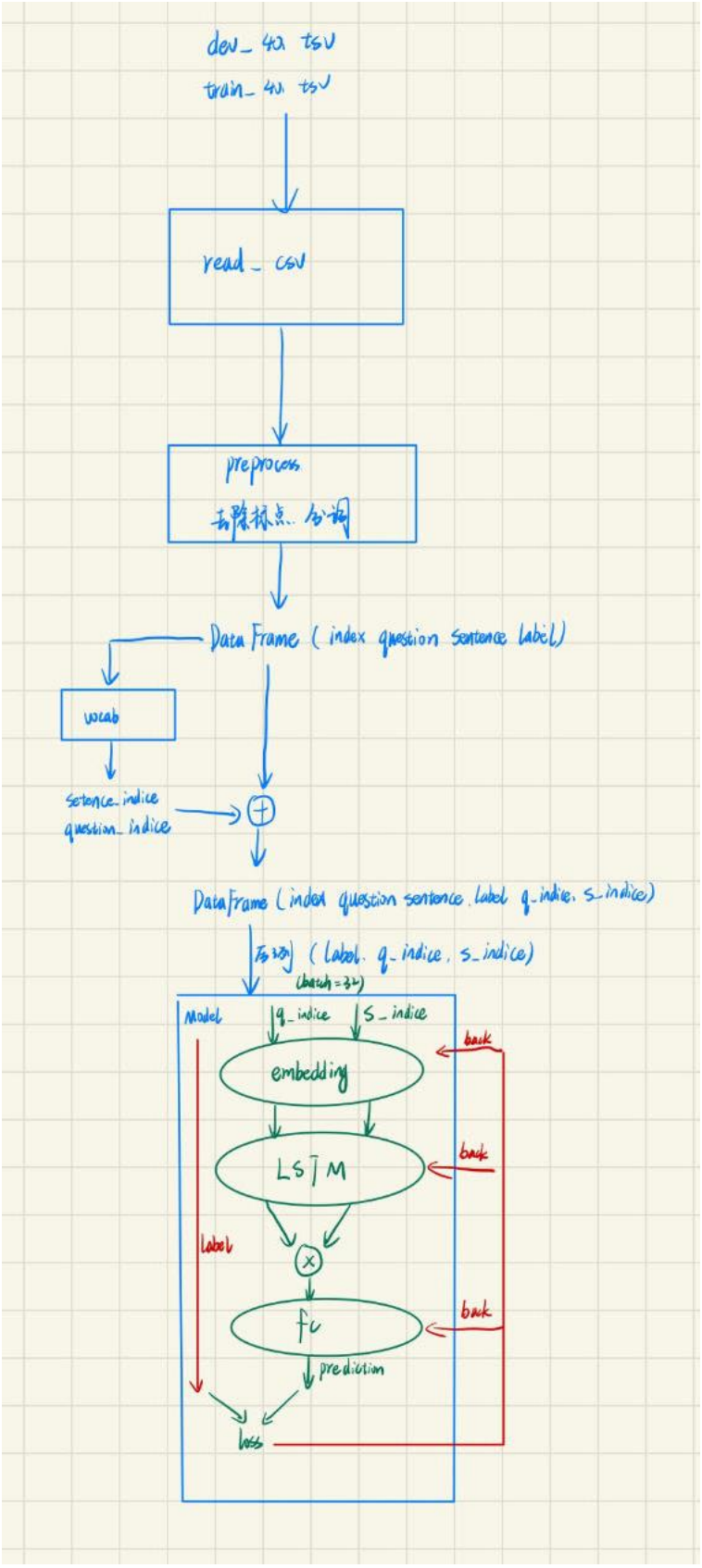
二、 实验内容

1. 算法原理

自然语言推理（NLI）是一种自然语言处理（NLP）任务，旨在让机器理解句子之间的逻辑关系。NLI 算法的核心目标是确定两个给定的句子——通常是一对前提和假设——之间的语义关系。

本代码模型首先将输入的文本序列进行预处理，之后构建一个词汇表，通过遍历训练集和测试集，将所有单词加入到 set 中，并将词汇表转换为词索引对应的字典。接着利用字典把词向量转换为索引向量，再经过一系列处理后送入神经网络（关于神经网络具体细节在下面关键代码部分给出），使用交叉熵损失函数和 Adam 优化器进行优化。

2. 流程图


dev_4w.tsv
train_4w.tsv
read_csv
preprocess
去除标点、分词
DataFrame (index question sentence label)
vocab
sentence_indice
question_indice
DataFrame (index question sentence label q_indice, s_indice)
(label, q_indice, s_indice)
(batch=32)
Model
q_indice
s_indice
embedding
LSTM
fc
prediction
loss
label
back
back
back

3. 关键代码展示（带注释）

数据预处理

使用 QUOTE_NONE，读取时忽略引号作为句子的分隔

```
# 下载 NLTK 的 punkt 模型
nltk.download('punkt')

# 读取数据集
def read_tsv(file_path):
    return pd.read_csv(file_path, sep='\t', header=None, names=['index',
'question', 'sentence', 'label'], quoting=csv.QUOTE_NONE)

train_df = read_tsv('QNLI/train_40.tsv')
dev_df = read_tsv('QNLI/dev_40.tsv')

# 数据预处理函数
def preprocess_text(text):
    text = text.lower()  # 转为小写
    text = re.sub(f"[{string.punctuation}]", "", text)  # 移除标点符号
    text = word_tokenize(text)  # 分词
    return text

# 应用预处理函数
train_df['question'] = train_df['question'].apply(preprocess_text)
train_df['sentence'] = train_df['sentence'].apply(preprocess_text)
dev_df['question'] = dev_df['question'].apply(preprocess_text)
dev_df['sentence'] = dev_df['sentence'].apply(preprocess_text)
```

词向量转换索引向量

```
# 将句子转换为索引序列
def convert_to_indices(sentence, word2idx):
    return [word2idx[word] if word in word2idx else 0 for word in
sentence]

train_df['question_indices'] = train_df['question'].apply(lambda x:
convert_to_indices(x, word2idx))
train_df['sentence_indices'] = train_df['sentence'].apply(lambda x:
convert_to_indices(x, word2idx))
dev_df['question_indices'] = dev_df['question'].apply(lambda x:
convert_to_indices(x, word2idx))
dev_df['sentence_indices'] = dev_df['sentence'].apply(lambda x:
convert_to_indices(x, word2idx))
```

```
# 将标签转换为数值
label2idx = {'entailment': 1, 'not_entailment': 0}
train_df['label'] = train_df['label'].map(label2idx)
dev_df['label'] = dev_df['label'].map(label2idx)

train_df = train_df.dropna(subset=['label'])
dev_df = dev_df.dropna(subset=['label'])

train_df['label'] = train_df['label'].astype(int)
dev_df['label'] = dev_df['label'].astype(int)

# 自定义数据集类
class NLIDataset(Dataset):
    def __init__(self, df):
        self.df = df

    def __len__(self):
        return len(self.df)

    def __getitem__(self, idx):
        question = torch.tensor(self.df.iloc[idx]['question_indices'])
        sentence = torch.tensor(self.df.iloc[idx]['sentence_indices'])
        label = torch.tensor(self.df.iloc[idx]['label']).long()
        return question, sentence, label

# 数据对齐（padding）
def collate_fn(batch):
    questions, sentences, labels = zip(*batch)
    questions_padded = pad_sequence(questions, batch_first=True,
padding_value=0)
    sentences_padded = pad_sequence(sentences, batch_first=True,
padding_value=0)
    labels = torch.stack(labels)
    return questions_padded, sentences_padded, labels
```

神经网络定义

```
class NLIModel(nn.Module):
    def __init__(self, vocab_size, embed_size, hidden_size,
output_size):
        super(NLIModel, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embed_size,
padding_idx=0)
        self.lstm = nn.LSTM(embed_size, hidden_size, batch_first=True,
bidirectional=True)
```

```
        self.fc = nn.Linear(hidden_size * 2, output_size)

    def forward(self, question, sentence):
        question_embed = self.embedding(question)
        sentence_embed = self.embedding(sentence)

        _, (question_hidden, _) = self.lstm(question_embed)
        _, (sentence_hidden, _) = self.lstm(sentence_embed)

        question_hidden = torch.cat((question_hidden[-2],
question_hidden[-1]), dim=1)
        sentence_hidden = torch.cat((sentence_hidden[-2],
sentence_hidden[-1]), dim=1)

        combined = question_hidden * sentence_hidden
        output = self.fc(combined)

        return output
```

超参数 vocab_size = 67605（单词数量），embed_size = 100，hidden_size = 128，output_size = 2（二分类），batch_size = 32。

以 question 为例，我们首先将形状为[batch_size, length]的 question 语句索引向量输入到嵌入层中，每个句子的单词被嵌入成一个高维向量，因此输出后形状变为[batch_size, length, embed_size]，接着输入到双向 LSTM 层，由于双向，所以形状为[2, batch_size, hidden_size]，调用 torch.cat 函数将正向和反向序列拼接，得到形状为[batch_size, hidden_size * 2]的张量，经过 combined 层进行 question 和 setence 的逐元素乘积，在经过全连接层，输出为[batch_size, 2]的张量，用于二分类。

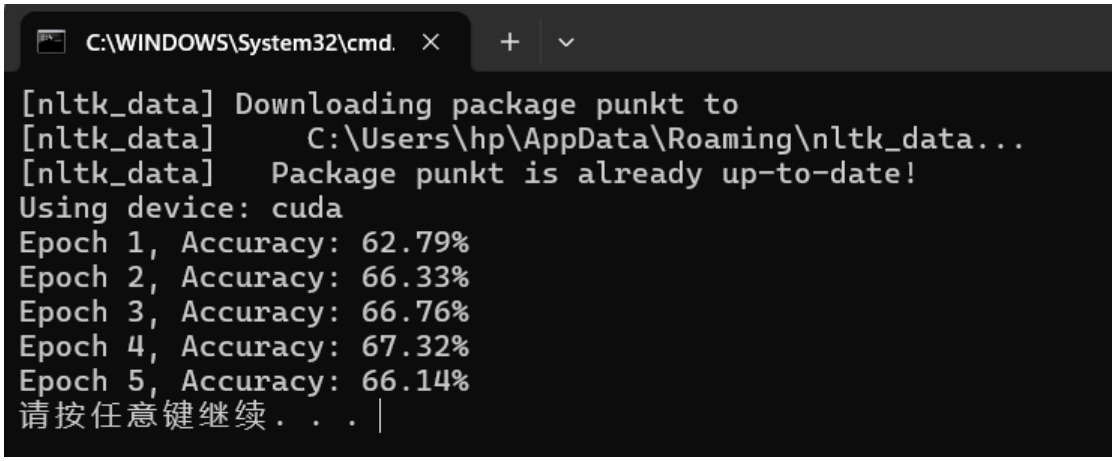
4. 创新点&优化（如果有）

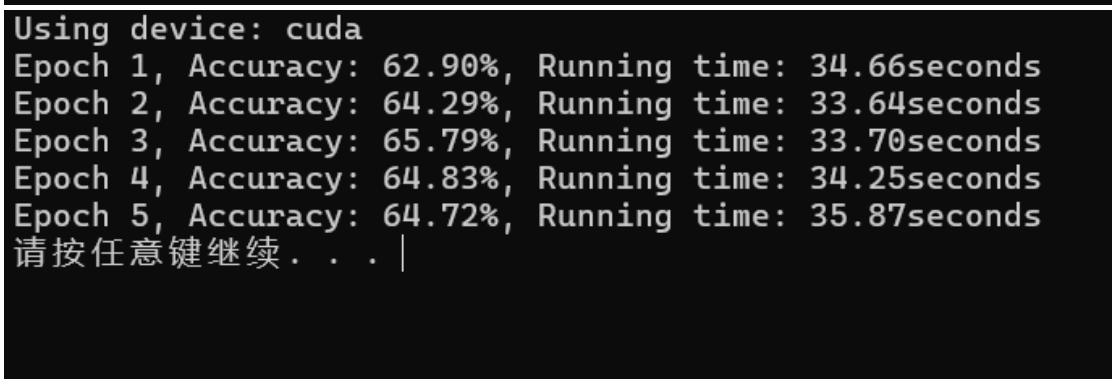
1、神经网络中，选择双向 LSTM 层（长短期记忆网络），通过两个不同的隐藏层进行处理，一个正向序列，另一个反向序列，捕捉前后文信息。

2、数据对齐（padding），在创建 batch 时，由于每个样本的长度不同，使用 pad_sequence 函数统一输入维度。

三、 实验结果及分析

|----------如有优化，请重复 1，2，分析优化后的算法结果----------|


C:\WINDOWS\System32\cmd.
[nltk_data] Downloading package punkt to
[nltk_data] C:\Users\hp\AppData\Roaming\nltk_data...
[nltk_data] Package punkt is already up-to-date!
Using device: cuda
Epoch 1, Accuracy: 62.79%
Epoch 2, Accuracy: 66.33%
Epoch 3, Accuracy: 66.76%
Epoch 4, Accuracy: 67.32%
Epoch 5, Accuracy: 66.14%
请按任意键继续. . .


Using device: cuda
Epoch 1, Accuracy: 62.90%, Running time: 34.66seconds
Epoch 2, Accuracy: 64.29%, Running time: 33.64seconds
Epoch 3, Accuracy: 65.79%, Running time: 33.70seconds
Epoch 4, Accuracy: 64.83%, Running time: 34.25seconds
Epoch 5, Accuracy: 64.72%, Running time: 35.87seconds
请按任意键继续. . .