隐式篇章分析报告

1. 内容

构建神经网络完成隐式篇章关系判别，并用 marco-f1 衡量判别的好坏。

2. 环境

Python3
pytorch 库
transformers 库
time 库

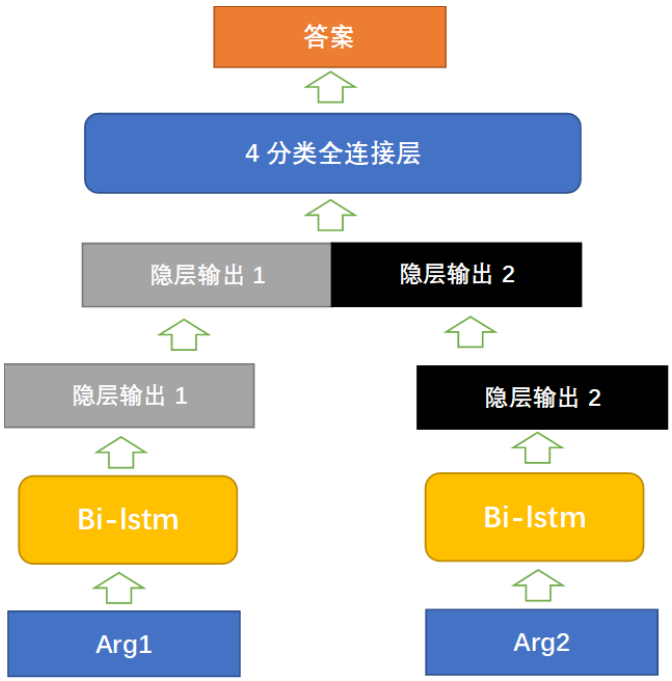
3. 代码构成

本实验代码有 3 个文件，分别为 load-data.py，model.py，main.py。
其中，load-data.py 其中的代码为加载数据集使用，model.py 中的代码为设定模型使用，main.py 中的代码为训练与测试代码。

4. 过程及结果

1.bi-lstm 模型

一开始时所使用的模型结构如下：



代码如下：

```python
class Bi_LSTMmodel(nn.Module):

    #vocab_size为一共有多少单词
    def __init__(self, vocab_size, embed_size, hidden_size, num_layers,max_sent_len):
        super(Bi_LSTMmodel, self).__init__()
        self.embed = nn.Embedding(vocab_size, embed_size)
        self.lstm = nn.LSTM(embed_size, hidden_size, num_layers, batch_first=True,bidirectional=True)
        self.linear = nn.Linear(hidden_size * 4 * max_sent_len, 4)
    def forward(self, x,y):

        x = self.embed(x)
        y = self.embed(y)
        out_x, _ = self.lstm(x)
        out_y, _ = self.lstm(y)
        out = torch.cat([out_x,out_y],1)
        out = out.reshape(out.shape[0],out.shape[1] * out.shape[2])
        out = self.linear(out)
        return out
```

因为隐式关系判别可以当作是句子的分类，所以在 nlp 处理中经常使用 bi-lstm 对句子进行编码，这种编码方式照顾到了一个句子的前后关系，那么我就可以对 arg1 与 arg2 分别进行编码然后将其拼接在一起。

其基本思路为先输入 arg1 与 arg2，全部都为 token 后的句子，然后做 embedding，分别输入 bi-lstm 层中进行编码，得到两个 bi-lstm 编码后隐层输出的句子，将两个隐层输出拼接在一起，arg1 在前 arg2 在后，一同输入进线性层进行分类，线性层的输入大小为 bi-lstm 隐层大小 * 4，输出大小为 4，对应 4 个标签。

在实际的操作中，超参数设置如下，其中的涵义都标明在注释之中，其中选择了交叉熵作为损失函数，选用了 Adam 算法作为了优化函数：

```python
embedding_dim = 100     #词向量长度
max_sent_len = 50       #一个句子最大长多少
batch_size = 15         #batch大小
epochs = 10             #重复次数
lr = 1e-4               #学习率
hidden_size = 80        #隐层的节点数
num_layers = 1          #几层隐层
```

```python
loss_function = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=lr, weight_decay=5e-4)
```

而在训练的时候，我会记录该模型在验证集上的准确率，选取在验证集上准确率最高的模型作为我测试的模型。

```python
if valid_acc > max_valid_acc:
    torch.save(model, "checkpoint.pt")
```

最后，在训练了 10 个 epoch 后，模型得到的 marco-f1 值为：34.40%：

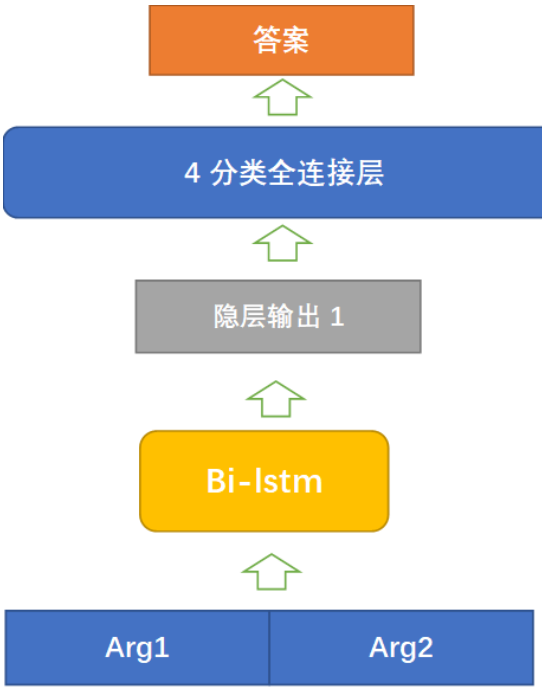
```
测试集上Marco_F1的值为:34.40%
```

之后我又进行了 20 个 epoch 的训练，得到的 marco-f1 值为：34.43%

```
测试集上Marco_F1的值为:34.43%
```

无明显提升。

后来考虑到不仅仅是句子之内的前后文需要关注，既然是隐式篇章关系分析，那么句子之间的前后文也是很重要的，于是改为以下的模型，将 arg1 与 arg2 中间添加“<eos>”，代表句子的分割，然后拼接在一起一起送入 bi-lstm 层进行编码然后送入全连接线性层进行分类。




```python
class Bi_LSTMmodel(nn.Module):

    #vocab_size为一共有多少单词
    def __init__(self, vocab_size, embed_size, hidden_size, num_layers,max_sent_len):
        super(Bi_LSTMmodel, self).__init__()
        self.embed = nn.Embedding(vocab_size, embed_size)
        self.lstm = nn.LSTM(embed_size, hidden_size, num_layers, batch_first=True,bidirectional=True)
        self.linear = nn.Linear(hidden_size * 2 * max_sent_len, 4)
    def forward(self, x):

        x = self.embed(x)
        out, _ = self.lstm(x)
        out = out.reshape(out.shape[0],out.shape[1] * out.shape[2])
        out = self.linear(out)
        return out
```

参数设置除了 max_sent_len 改为了 150，其他都没变化，最后得到的答案是：

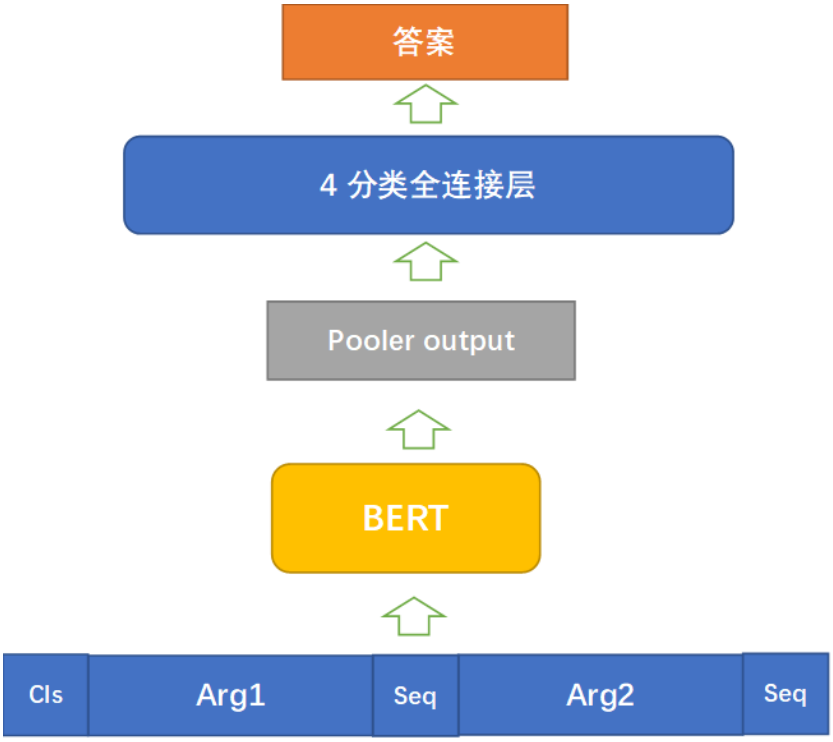
Marco-f1 值为 31.05%

```
测试集上Marco_F1的值为:31.05%
```

可以看出无明显提升，于是在这条路上到此为止放弃。

2. roberta 预训练模型

因为 bi-lstm 的效果达不到合格，所以转为以下模型：



首先将 arg1 与 arg2 中间添加 seq 与 cls 标识符，这一点 transformers 库中的 bert 的 tokenizer 自动可以做到，然后输入 bert 模型之中，得到两个输出一个 pooled output，一个 sequence output，这里尝试使用各种输入，输入进分类全连接层进行分类，最后得到答案。

接下来为代码的细节部分：

模型选用的为 roberta 模型，其下载地址为：

https://huggingface.co/roberta-base

需要导入 tokenizer，导入后对 arg1 和 arg2 使用 encode_plus 函数就能自动得到我们需要的句子表示向量，其中会返回两个向量，一个是 input_ids，还有一个是 attention_mask，都需要输入进模型之中。

```
30          vocab_file = 'model/vocab.json'
31          merges_file = 'model/merges.txt'
32          tokenizer = RobertaTokenizer(vocab_file, merges_file)
```

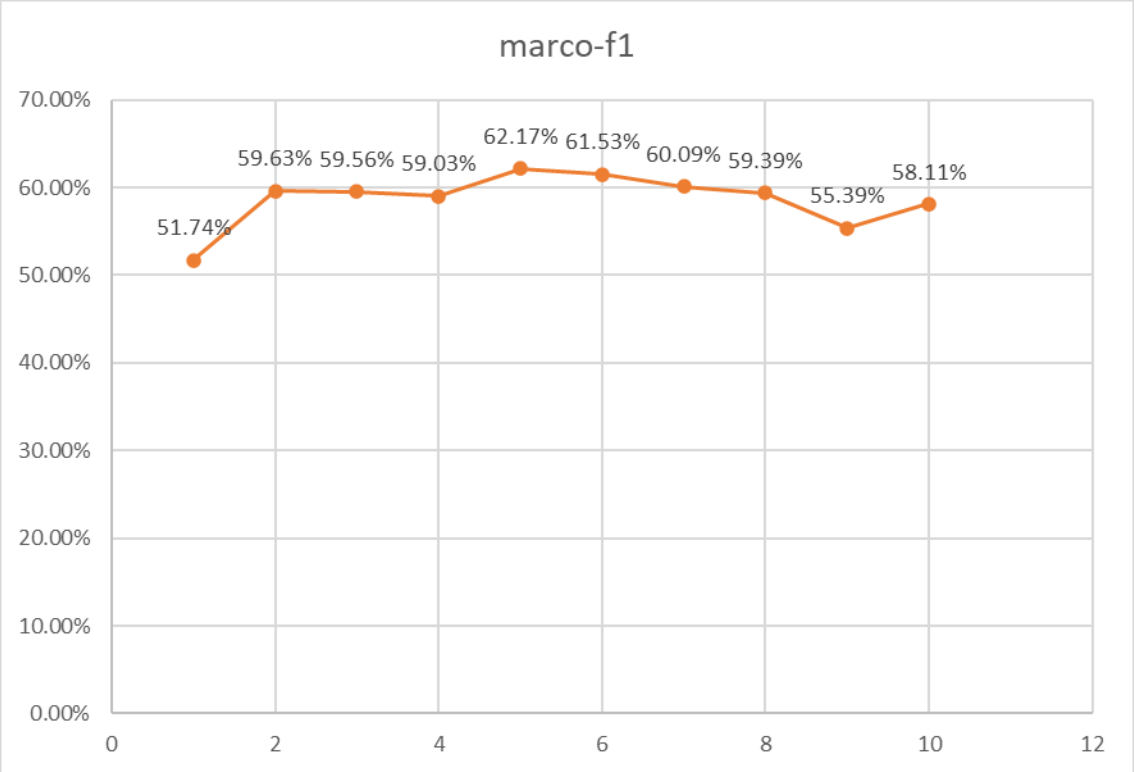
```
47              ids = tokenizer.encode_plus(arg1,arg2,
48                                          max_length=self.max_sent_len,
49                                          pad_to_max_length=True)
50              #会返回一个input_ids，一个attention_mask，分别储存
51              idss.append(ids['input_ids'])
52              masks.append(ids['attention_mask'])
```

以下为模型的构建代码，其中输入的 x 为 input_ids，z 为 attention_mask，out 取 out[1]即为 pooler output，out[0]即为 sequence output，输入进输入大小为 bert 的设定隐层大小，输出为 4 的线性层之中。

```python
class Transformer_model(nn.Module):

    #vocab_size为一共有多少单词
    def __init__(self):
        super(Transformer_model, self).__init__()
        self.bert = RobertaModel.from_pretrained("model\\roberta-base\\")
        self.linear = nn.Linear(self.bert.config.hidden_size, 4)
    def forward(self, x,z):

        out = self.bert(input_ids=x,attention_mask=z)
        out = out[1]
        # out = out[0]
        # out = out[:,0,:]
        out = self.linear(out)

        return out
```

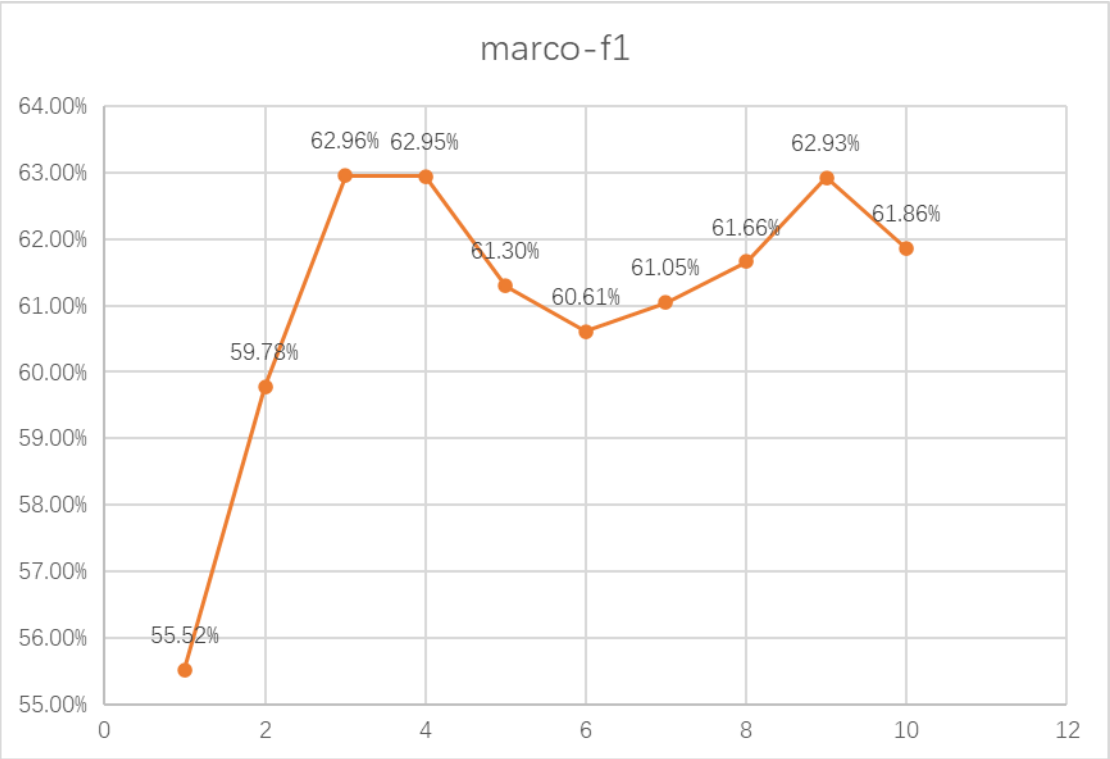
训练的过程中不记录模型，在每训练一个 epoch 之后就用测试集测试出 marco-f1 分数并输出。以下的设置不变，选用交叉熵损失函数与 torch 的 AdamW 优化器，修正了 Adam 中权重衰减的 bug。

第一次选用 sequence output，在训练了 10 个 epoch 后，每个 epoch 的分数如下：



可以看出最高为 62.17%，得到了明显的提升。

接下来使用 output[1]也就是 pooler output，在训练了 10 个 epoch 后，每个 epoch 的分数：

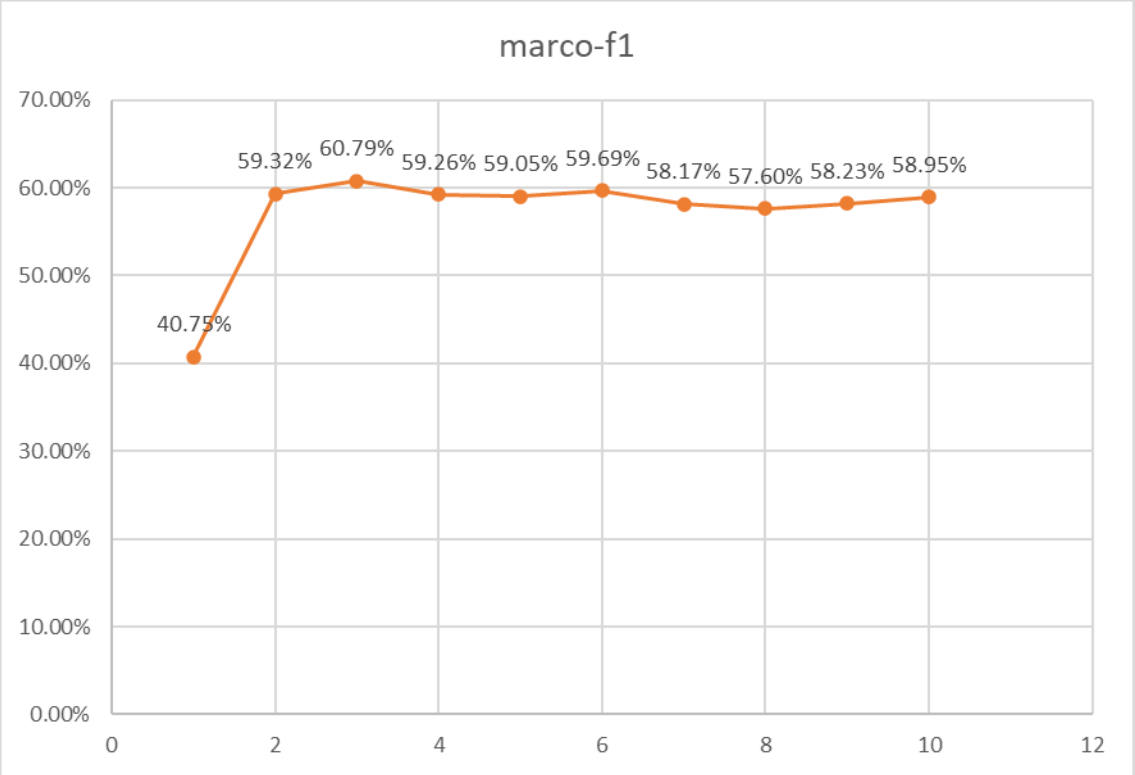


可以看出使用 pooler output 在总体上对于分数是有个提升的，最高可以到 62.95% ，于是后面就都使用 roberta 的 pooler output 输出。

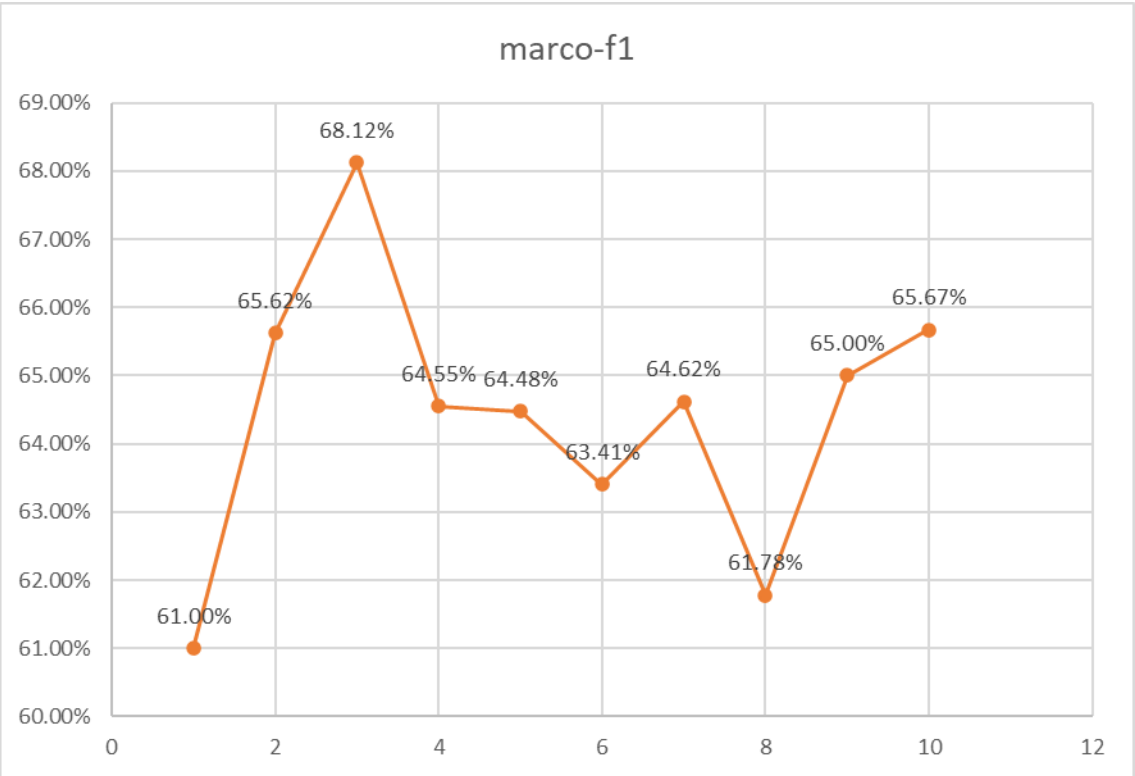
接下来我尝试修改线性层的构成，换成更加复杂的线性层，替换的理由就是更加复杂精细的线性层按理来说更加的具有可塑性，可以塑造成精度更高的分类器，具体修改代码如下，替换为先经过一个 hidden_size，输出为 400 的线性层，然后再经过一个 ReLu 激活层，最后再经过一个输入为 400，输出为 4 的分类层。

```
class Transformer_model(nn.Module):

    #vocab_size为一共有多少单词
    def __init__(self):
        super(Transformer_model, self).__init__()
        self.bert = RobertaModel.from_pretrained("model\\roberta-base\\")
        self.linear = nn.Linear(self.bert.config.hidden_size, 400)
        self.Relu = nn.ReLU()
        self.linear2 = nn.Linear(400,4)
    def forward(self, x,z):

        out = self.bert(input_ids=x,attention_mask=z)
        out = out[1]
        # out = out[0]
        # out = out[:,0,:]
        out = self.linear(out)
        out = self.Relu(out)
        out = self.linear2(out)

        return out
```

最后的结果如下，可以看出，其实效果有明显的下降，甚至训练时间还更长了，反而还是简单的单一线性层效果更好，于是就保留了原来线性层。



之后，为了追求更高的分数，我将模型替换为了 roberta-large 模型，下载地址为 https://huggingface.co/roberta-large
在这之后，我对模型进行了测试，得到的结果如下：



可以看出，roberta-large 对效果的提升是巨大的，最高可到 68.12%，至此模型就不再改动，到此结束。

5. 实验结果总结

Bi-lstm:最高 34.43%
Roberta 预训练：最高 62.95%
Roberta-large 预训练：最高 68.12%

最高分数：68.12%