文本情感识别实验报告

王一飞 2022010900 计28-经22

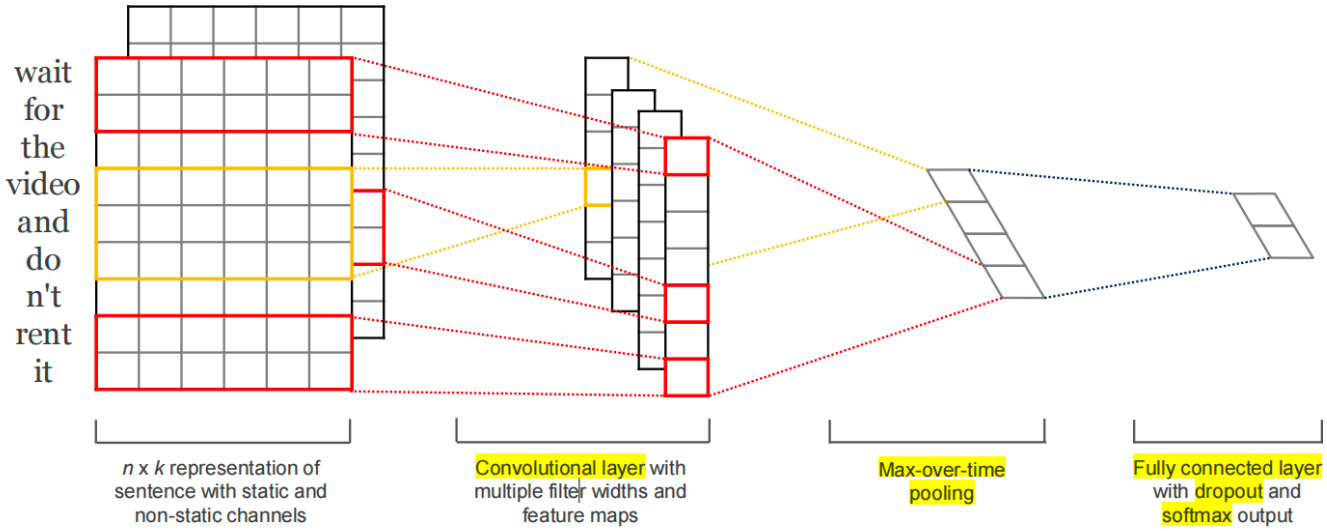
模型结构

本实验中实现了三个模型：CNN、RNN LSTM、MLP

CNN

> Yoon Kim. (2014). Convolutional Neural Networks for Sentence Classification. arXiv:1408.5882 [cs.CL].

CNN模型的实现参考了上面这篇论文，由词嵌入、卷积、最大池化、全连接四个模块组成。



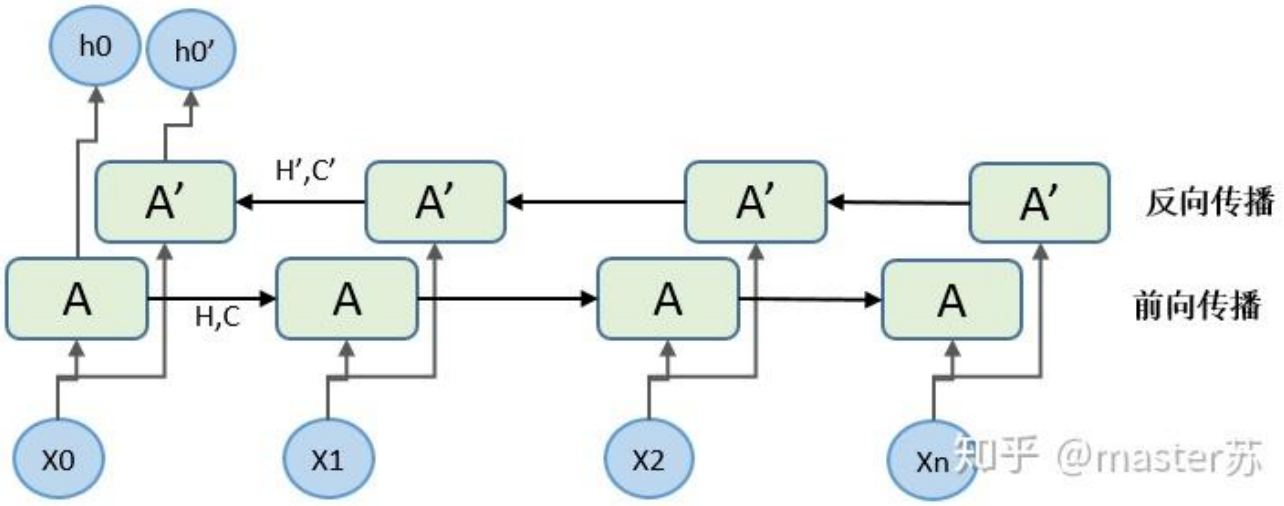

```python
class TextSentimentCNN(nn.Module):
    def __init__(self, n_filters, filter_sizes, output_dim, dropout):
        super().__init__()
        self.convs = nn.ModuleList([
            nn.Conv2d(in_channels=1, out_channels=n_filters, kernel_size=(fs, 50))
            for fs in filter_sizes
        ])
        self.fc = nn.Linear(len(filter_sizes) * n_filters, output_dim)
        self.dropout = nn.Dropout(dropout)
        # 初始化权重
        for conv in self.convs:
            nn.init.normal_(conv.weight, mean=0, std=0.02)
            nn.init.constant_(conv.bias, 0)
        nn.init.normal_(self.fc.weight, mean=0, std=0.02)
        nn.init.constant_(self.fc.bias, 0)

    def forward(self, text: torch.Tensor):
        text = text.unsqueeze(1)  # [batch size, 1, sent len, emb dim]

        conveds = []
        for conv in self.convs:
```

```
            conved = F.relu(conv(text)).squeeze(3)
            pooled = F.max_pool1d(conved, conved.shape[2]).squeeze(2)
            conveds.append(pooled)

        cat = self.dropout(torch.cat(conveds, dim=1))
        return self.fc(cat)
```

- Embedding：由于已经给定预训练词向量，在预处理文本时已经转化为`emb_dim=50`的词向量，故此处简化省略了该层。
- Convolutional Layers：采取了`n_filters=32, filter_sizes=[3, 4, 5]`的参数设置，用大小为3、4、5的卷积核进行三次卷积后直接拼接。
- Max-over-time Pooling：提取关键特征。
- Fully Connected Layers：采取了dropout来防止过拟合。

RNN LSTM

> https://zhuanlan.zhihu.com/p/139617364



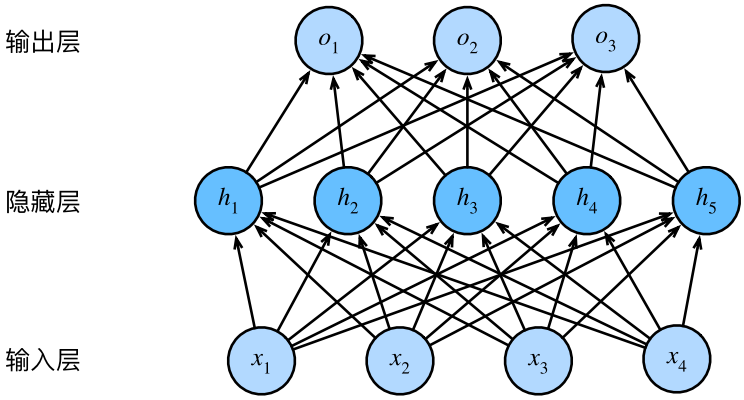
采取双向LSTM与全连接层

```
class TextSentimentRNN(nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim, n_layers, dropout):
        super(TextSentimentRNN, self).__init__()
        self.hidden_dim = hidden_dim
        self.n_layers = n_layers

        self.lstm = nn.LSTM(input_dim, hidden_dim, n_layers, bidirectional=True,
batch_first=True)
        self.fc = nn.Linear(2 * hidden_dim, output_dim)
        self.dropout = nn.Dropout(dropout)

        # 初始化权重
        nn.init.normal_(self.fc.weight, mean=0, std=0.01)
        nn.init.constant_(self.fc.bias, 0)

```

```
    def forward(self, text):
        # text shape: [batch size, seq len, emb dim] -> [64, 100, 50]
        output, (hidden, _) = self.lstm(text)
        hidden = hidden.view(self.n_layers, 2, -1, self.hidden_dim)
        hidden = torch.cat((hidden[-1, 0, :, :], hidden[-1, 1, :, :]), dim=1)
        hidden = self.dropout(hidden)
        hidden = self.fc(hidden)
        # hidden shape: [batch size, hidden_dim * num directions] -> [64, 256]
        return hidden  # [64, output_dim]
```

MLP

> https://zh.d2l.ai/chapter_multilayer-perceptrons/mlp.html



采用传统MLP架构，其中在输入时将`[batch size, seq len, emb dim]`的向量展平为`[batch size, seq len * emb dim]`以进行输入。

```
class TextSentimentMLP(nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim, dropout):
        super(TextSentimentMLP, self).__init__()
        self.fc1 = nn.Linear(input_dim, hidden_dim)
        self.fc2 = nn.Linear(hidden_dim, output_dim)
        self.dropout = nn.Dropout(dropout)

        # 初始化权重
        nn.init.normal_(self.fc1.weight, mean=0, std=0.01)
        nn.init.constant_(self.fc1.bias, 0)
        nn.init.normal_(self.fc2.weight, mean=0, std=0.01)
        nn.init.constant_(self.fc2.bias, 0)

    def forward(self, text):
        # text shape: [batch size, seq len, emb dim] -> [64, 100, 50]
        # 将输入的文本张量展平
        text = text.view(text.shape[0], -1)
        # text shape: [batch size, seq len * emb dim] -> [64, 5000]
        h1 = F.relu(self.fc1(text))
        h1 = self.dropout(h1)
        # h1 shape: [batch size, hidden dim] -> [64, 128]
        h2 = self.fc2(h1)
        return h2
```

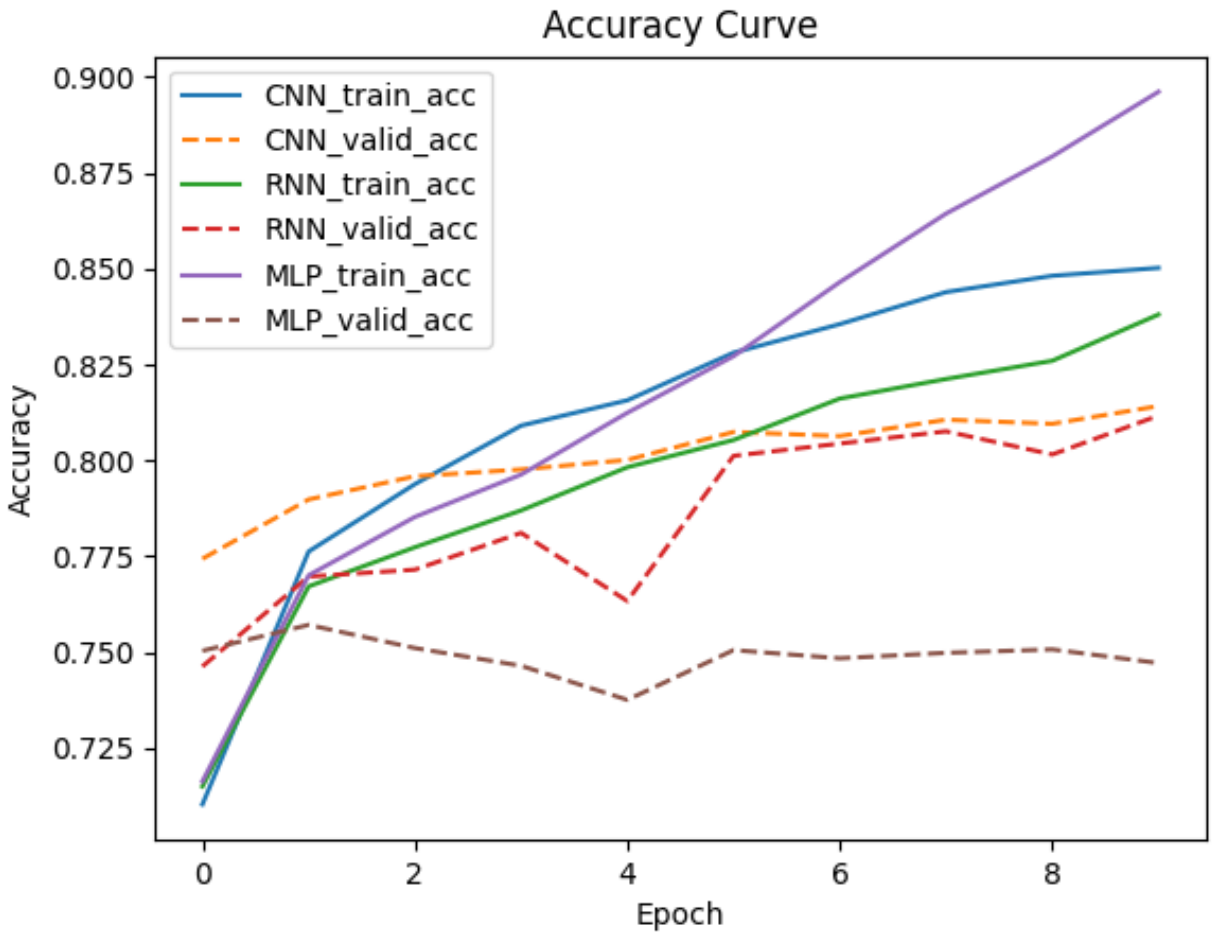
TextDataset

代码实现了自己的dataset类，实现了其中的`__len__`和`__getitem__`方法，因而可以直接由`torch. utils. data. dataloader. DataLoader`方法创建loader。

其中`__getitem__`方法取出`seq_len=100`的词向量序列，少则以`torch.zeros`补全，多则截取。

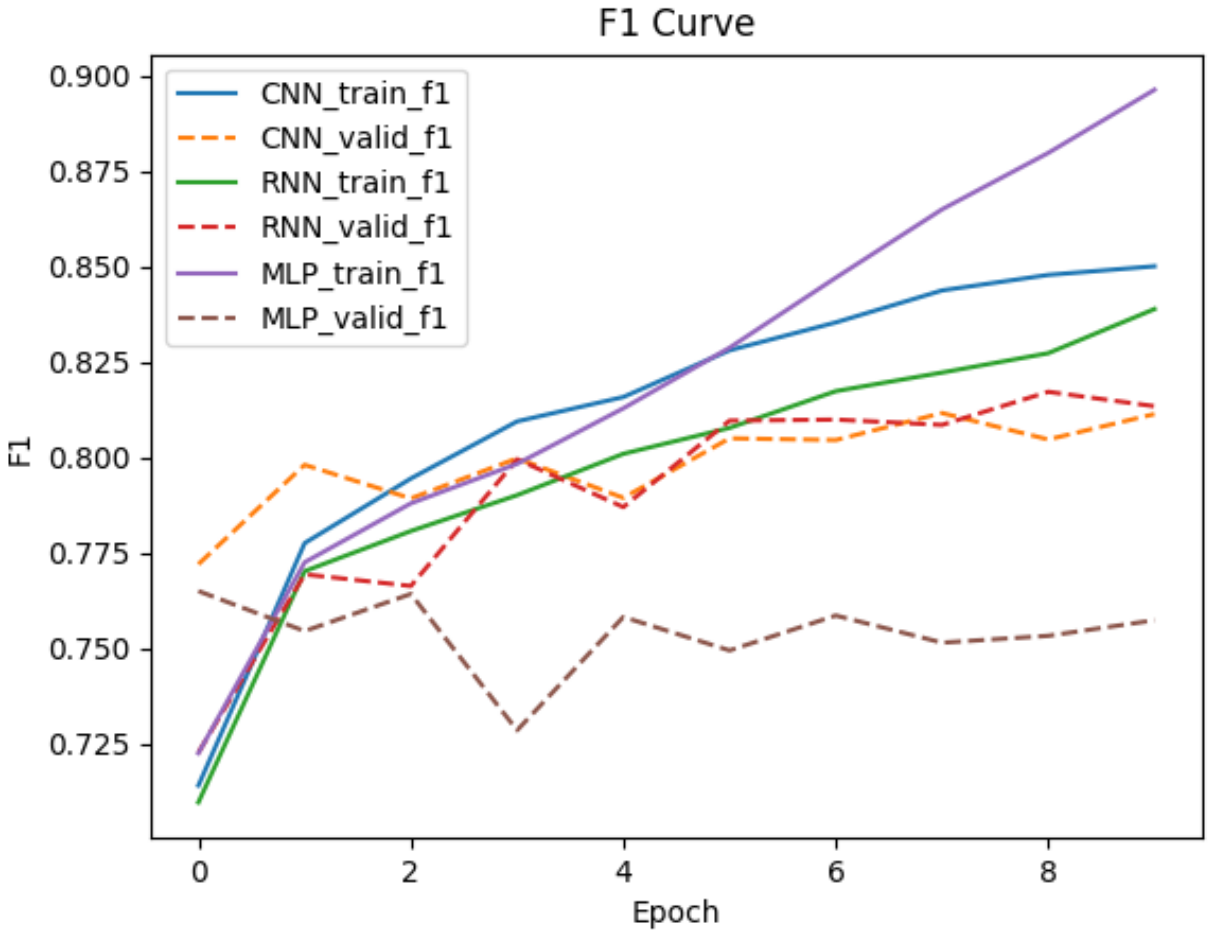
实验结果

Accuracy



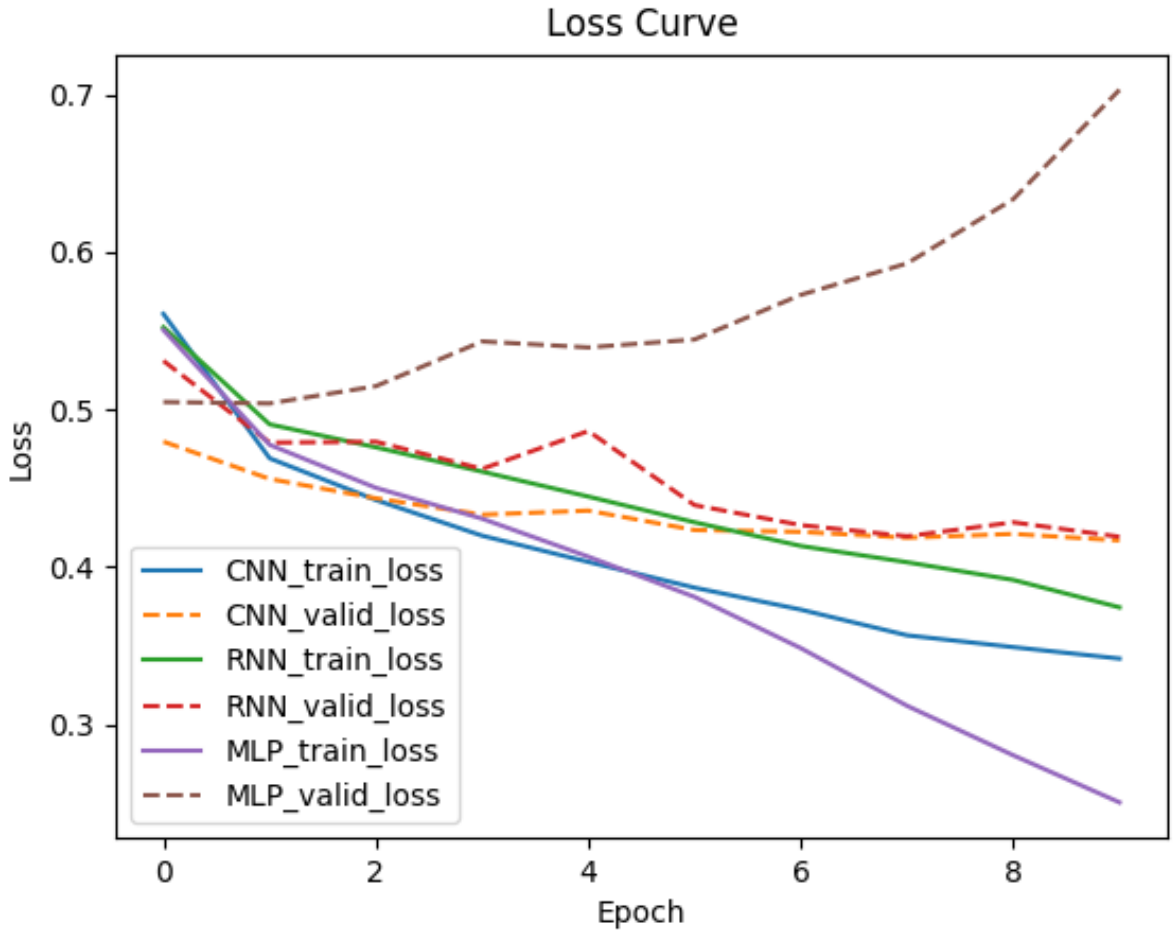
可以看出三个模型在训练集上的准确度都在逐步提高，其中MLP在训练集达到了90%的准确率而在验证集上提高并不明显，推测是模型过拟合导致。而CNN和RNN都表现出稳步提升的趋势。

F1 Score



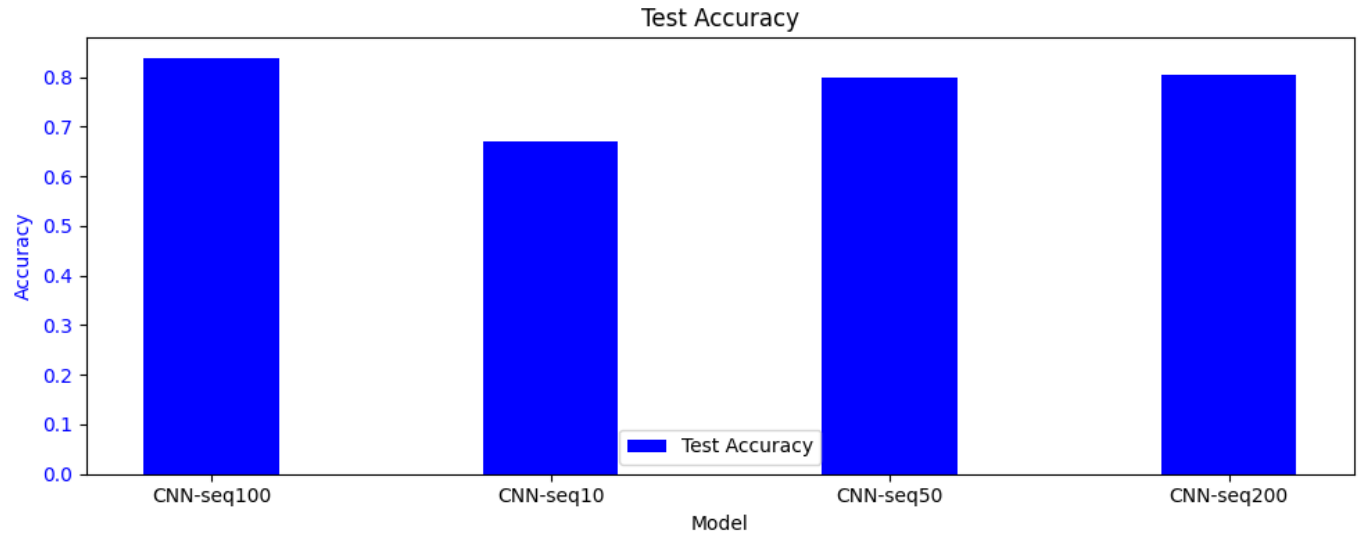
F1分数与Accuracy两个指标非常接近，曲线构型也颇为相同。

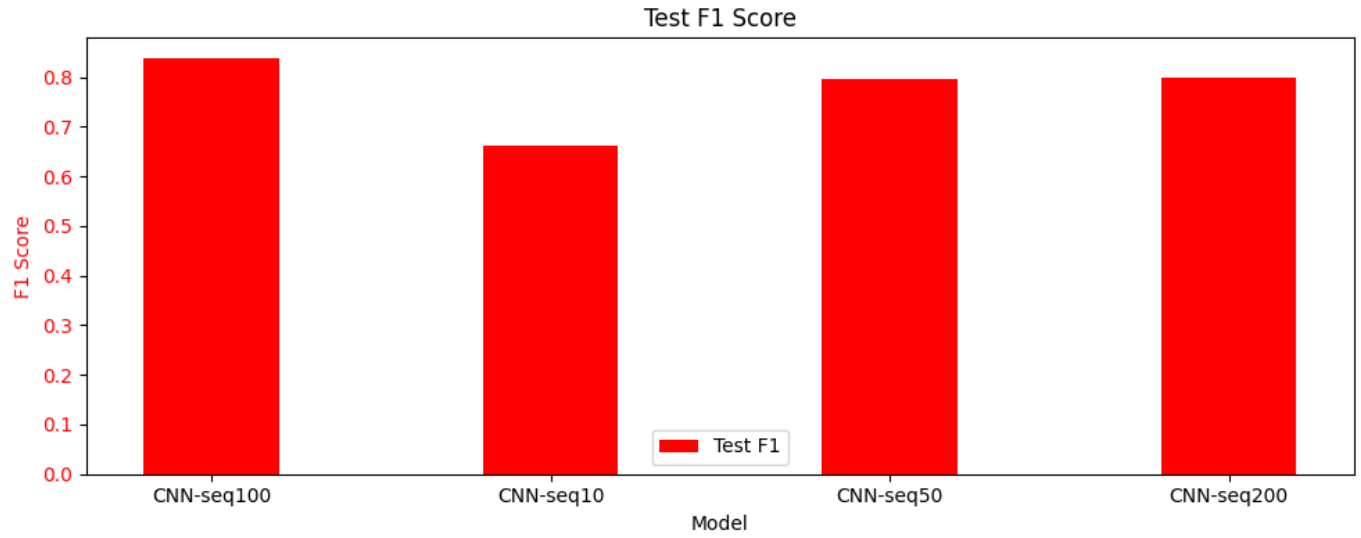
Loss

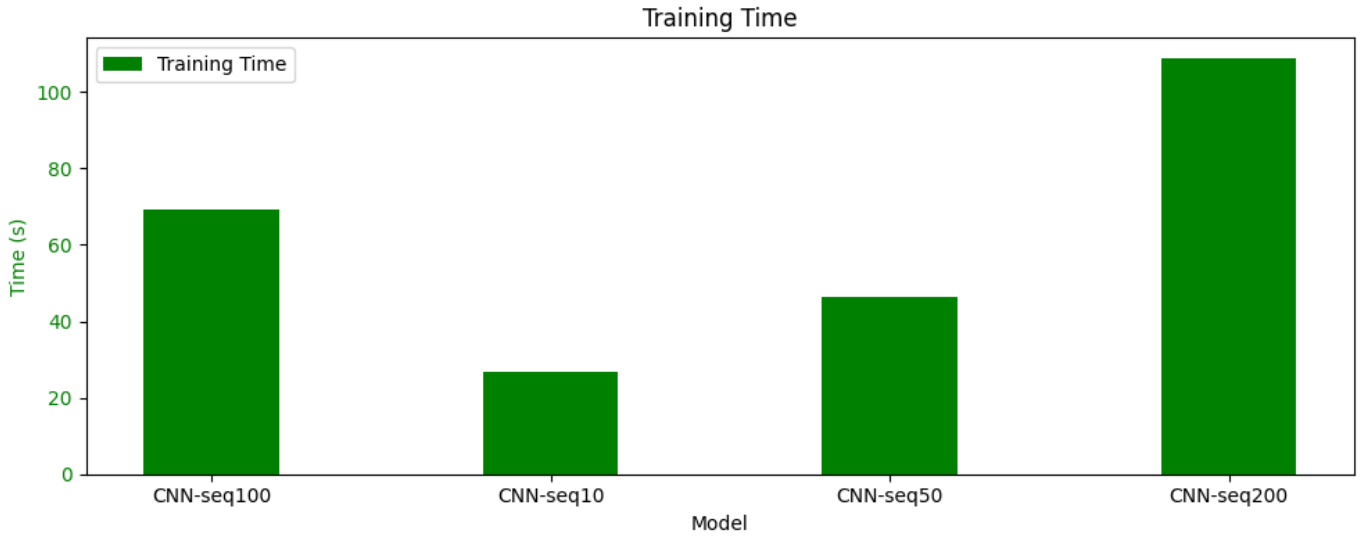


可以看出CNN和RNN都很好的完成了收敛，而MLP虽然在训练集上拟合的速度非常快，但在验证集上不降反增。

Test Dataset





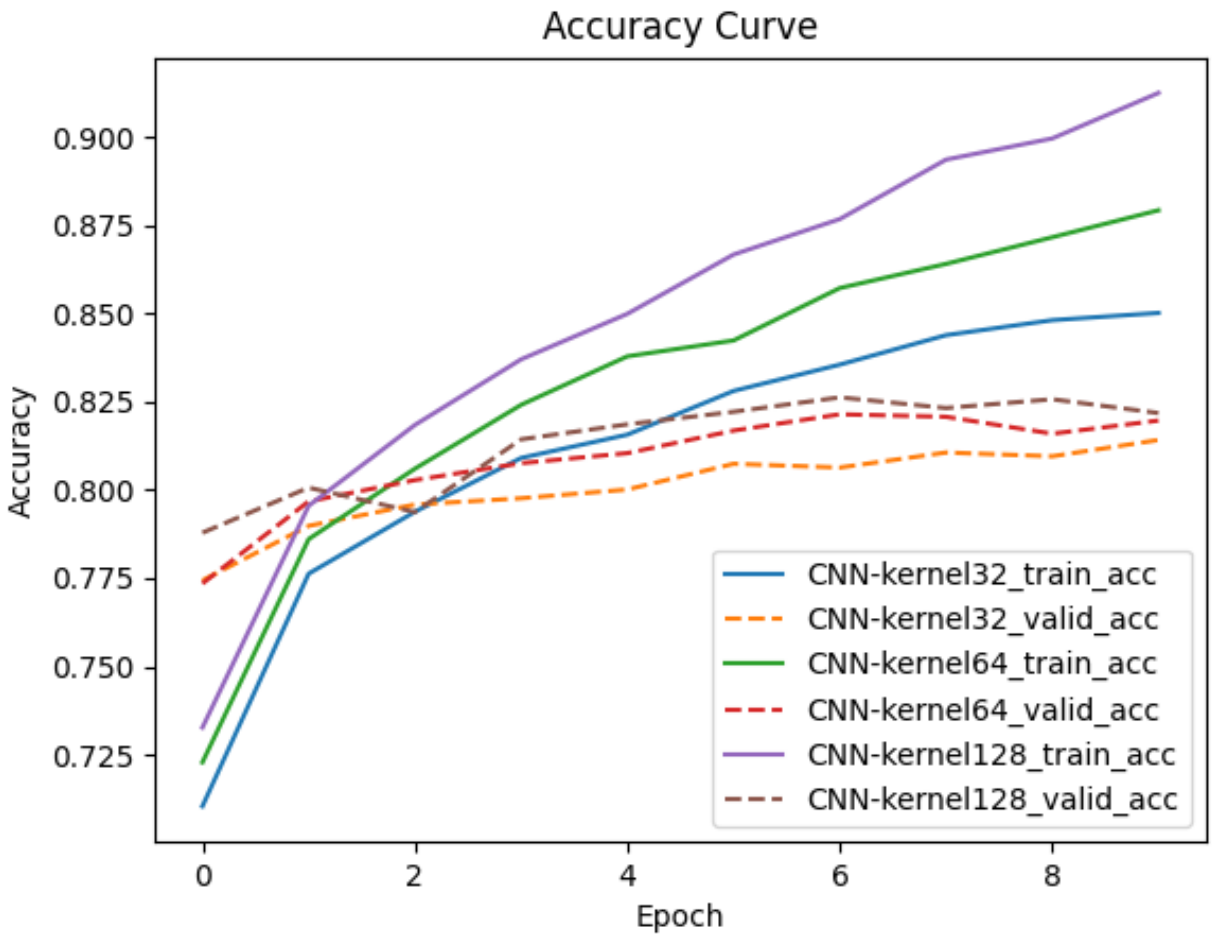


总体来说，三个模型都在测试集上达到了较高的准确率和f1分数，具体排序为RNN > CNN > MLP。推测RNN作为序列模型在处理文本时更加擅长。

在训练耗时方面，RNN的训练耗时远高于CNN和MLP，每一个epoch要花费接近一分钟，而CNN和MLP只需要5s和2s就能完成。

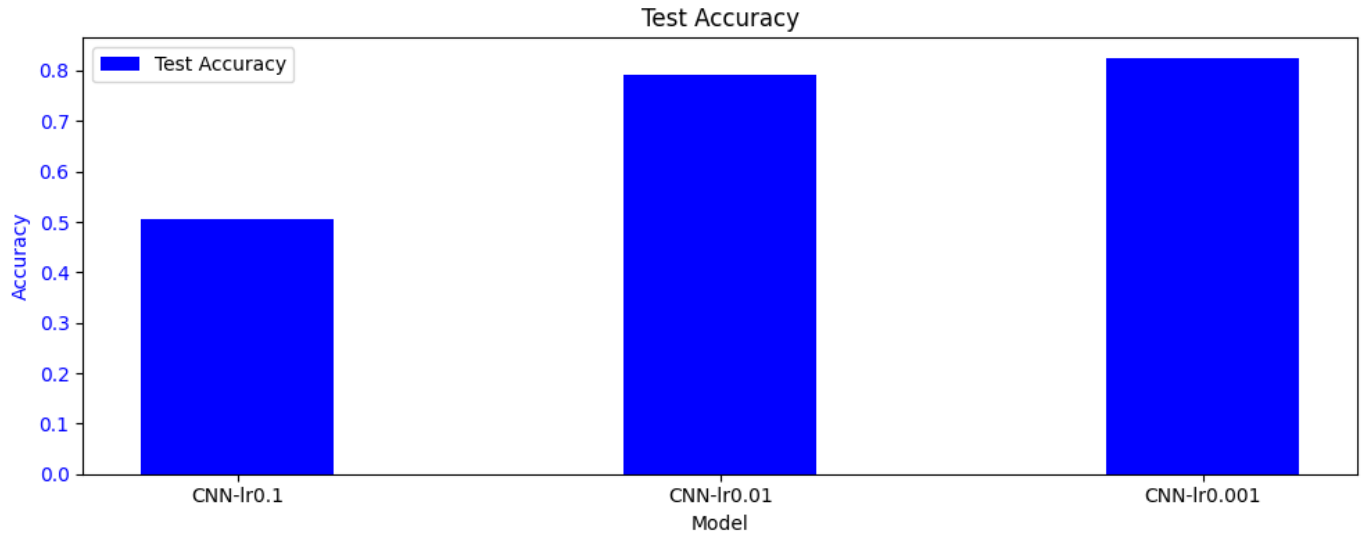
参数对比

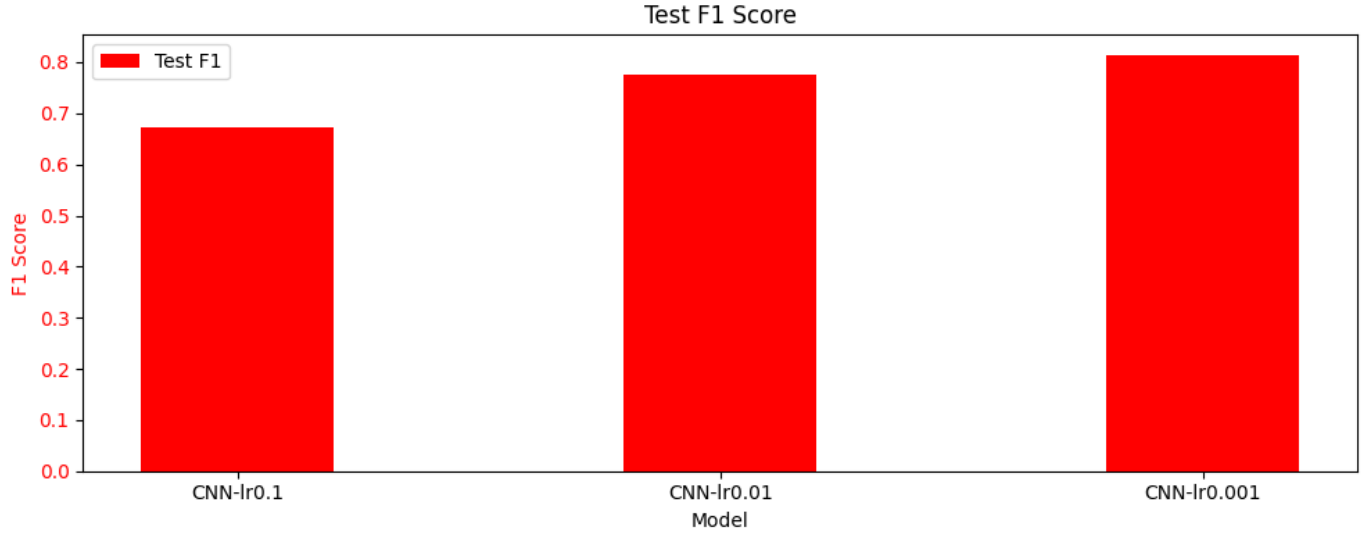
Convolutional Kernel

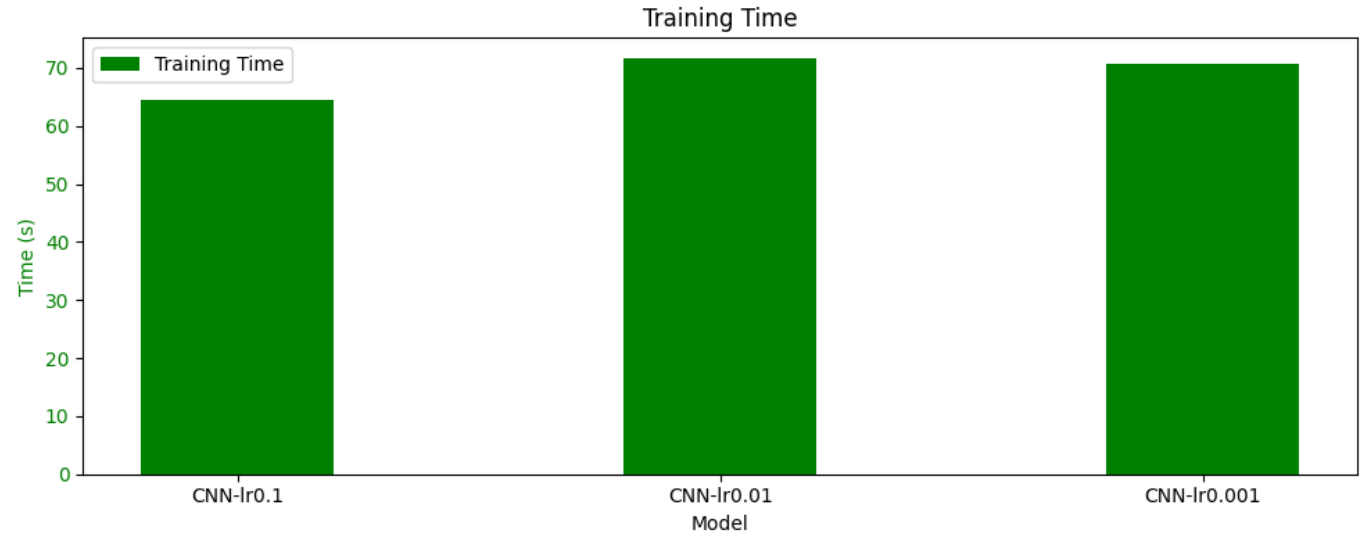


如图所示，`n_filters`在一定范围内的升高会提高模型的准确性。

Sequence length

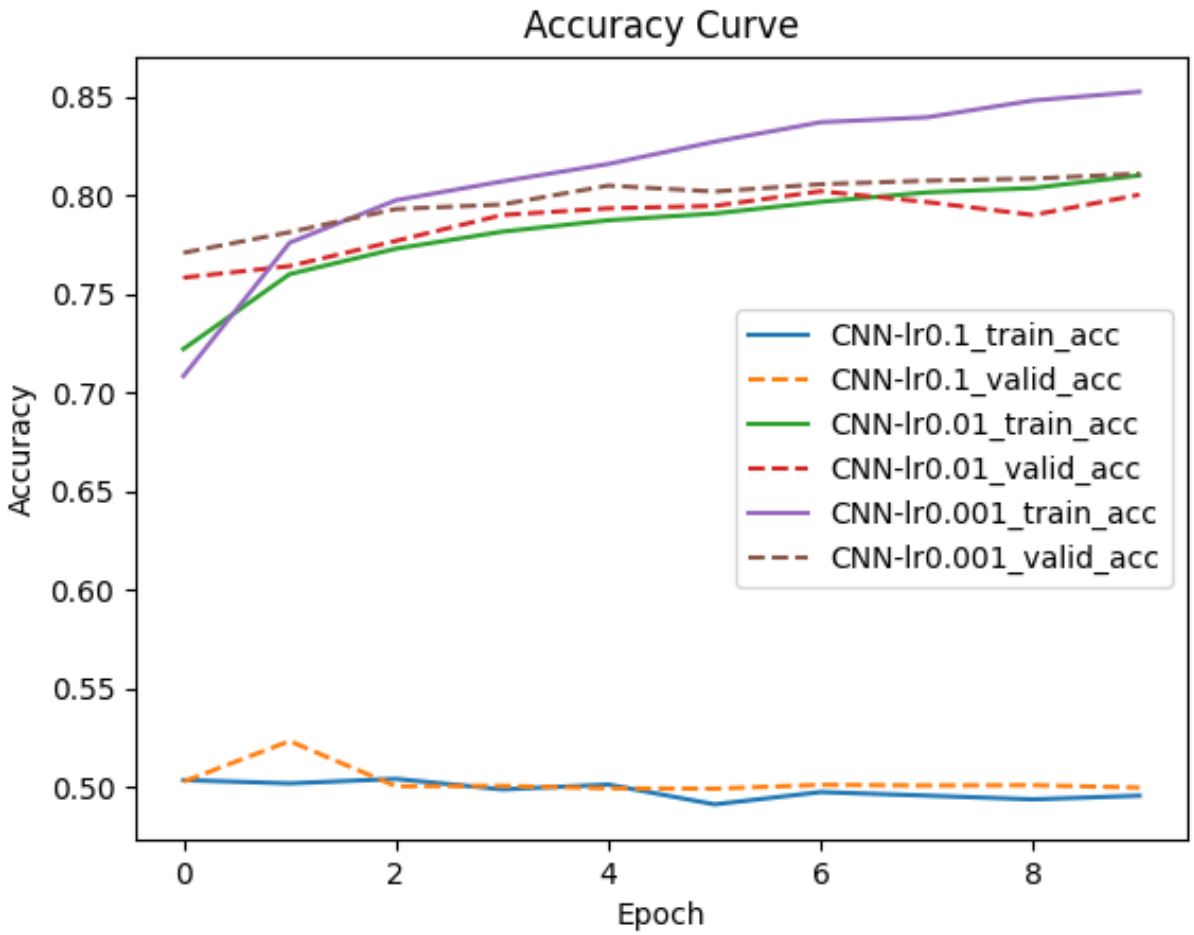






可以看出，随着序列长度的提升，模型准确性先升高后下降，训练时间持续增加。推测因为单个评论的长度位于50-200个词之间的概率较大，因此过少或者过多的次数会影响准确性，与直觉相符。

Learning rate



学习率过大会导致不收敛，符合先验直觉。

问题思考

训练停止时间

我目前的实现是固定十次训练迭代次数。由训练结果来看，模型在大约五次迭代后便进展缓慢。而如果采取当验证集数据下降的方法来控制训练次数，可能会导致训练时间方差过大。如果根据验证集收敛速度来控制训练次数，可能会有效避免这些问题。

参数初始化

我在实验中采取高斯分布来初始化我的权重W矩阵，以参数0初始化bias。

```
nn.init.normal_(self.fc.weight, mean=0, std=0.01)
nn.init.constant_(self.fc.bias, 0)
```

虽然有Xavier/Glorot 初始化、He 初始化、LeCun 初始化等多种方法，但多数参数初始化方法都以高斯分布为核心。

心得体会

通过本实验，我熟悉了pytorch的基本使用，更加深入的理解了几个深度学习模型在实现层面的种种细节。同时代码的框架、实验数据的整理也让我收获了很多技巧。