实验数据的获取在实验二中已有描述。

模型定义

```
class TextCNN(nn.Module):
    # 词汇表大小 词向量维度
    def __init__(self, vocab_size, embed_size, kernel_sizes, num_channels,
                 **kwargs):
        super(TextCNN, self).__init__(**kwargs)
        # 两个嵌入层，一个用于训练，一个不用于训练
        self.embedding = nn.Embedding(vocab_size, embed_size)
        self.constant_embedding = nn.Embedding(vocab_size, embed_size)
        # dropout层，防止过拟合
        self.dropout = nn.Dropout(0.5)
        # 全连接层，将卷积后的特征映射到输出类别
        self.decoder = nn.Linear(sum(num_channels), 2)
        self.pool = nn.AdaptiveAvgPool1d(1)
        self.relu = nn.ReLU()
        # 创建多个一维卷积层
        self.convs = nn.ModuleList()
        for c, k in zip(num_channels, kernel_sizes):
            self.convs.append(nn.Conv1d(2 * embed_size, c, k))

    def forward(self, inputs):
        # 两个嵌入层的输出在词向量维度上进行拼接
        embeddings = torch.cat((
            self.embedding(inputs), self.constant_embedding(inputs)), dim=2)
        # 调整张量的维度
        embeddings = embeddings.permute(0, 2, 1)
        # 卷积层提取特征并进行池化
        encoding = torch.cat([
            torch.squeeze(self.relu(self.pool(conv(embeddings)))), dim=-1)
            for conv in self.convs], dim=1)
        outputs = self.decoder(self.dropout(encoding))
        return outputs
```

其基本步骤如下：

定义

1. 定义两个嵌入层，一个用于训练，一个不用于训练；
2. 定义多个一维卷积核，具有不同宽度的卷积核可以捕获不同数目的相邻词元之间的局部特征。

前向传播

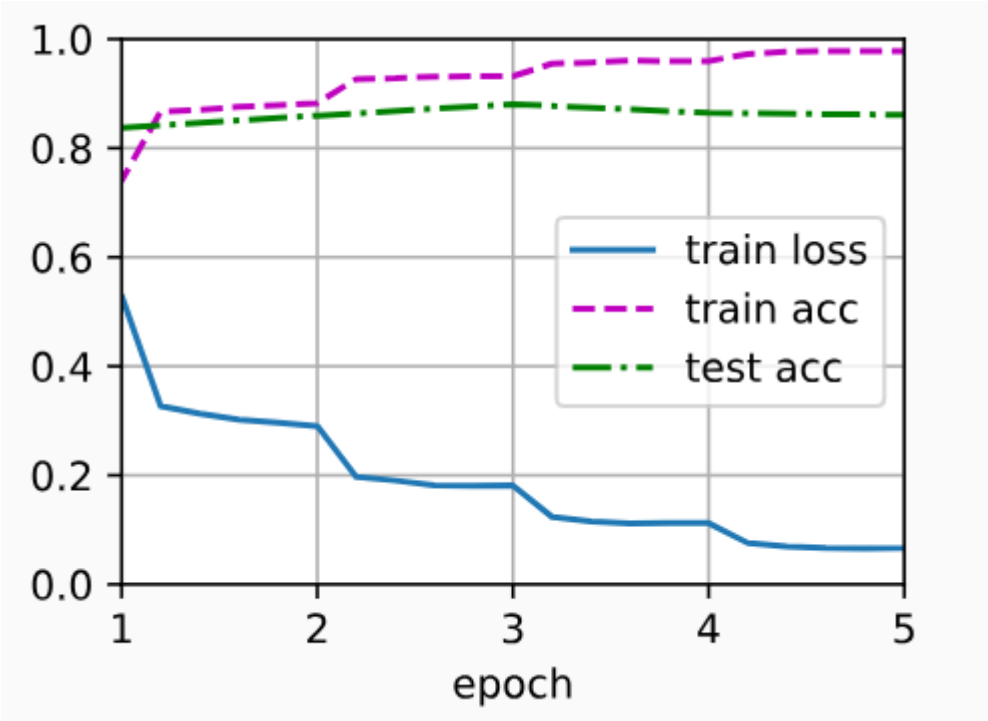
1. 将输入的词索引转化为词向量，并将两个嵌入层的输出在词向量上进行拼接；
2. 根据一维卷积核的输入格式，调整张量；
3. 每个一维卷积核在最大时间汇聚层合并，然后将所有标量汇聚输出连结为向量；
4. 使用全连接层将连结后的向量转换为输出类别。

训练和评估模型

使用后textCNN对模型进行情感分析。

```
lr, num_epochs = 0.001, 5
trainer = torch.optim.Adam(net.parameters(), lr=lr)
loss = nn.CrossEntropyLoss(reduction="none")
d2l.train_ch13(net, train_iter, test_iter, loss, trainer, num_epochs, devices)
```

模型训练过程如图：



对文本进行预测，结果如图：

```
predict_sentiment(net, vocab, 'this movie is so great')
```

```
'positive'
```

```
predict_sentiment(net, vocab, 'this movie is so bad')
```

```
'negtive'
```