深度学习 Lab4

PB20111686 黄瑞轩

1 实验目标

使用 pytorch 分别编写基于 RNN（LSTM）和 BERT 的语言模型，并基于训练好的词向量，实现文本情感分类（Text Sentiment Classification）：输入一个句子，输出是 0（负面）或 1（正面），并对两个不同模型的实验结果展开对比分析。

2 实验过程和关键代码展示

注：为了展示核心功能，某些代码中一些无关紧要的部分在报告中被删去了。

2.1 数据集说明

本次实验统一使用指定的 IMDB 公开数据集 “Large Movie Review Dataset”。该数据集分别包含 25,000 条电影评论作为训练集和测试集。

需要先将各样本经过 tokenizer 分词得到词向量之后，才可以作为 RNN（LSTM）或 BERT 的输入。

- RNN（LSTM）使用 `torchtext.get_tokenizer('basic_english')` 作为 tokenizer

```
class DataReader:
    def __init__(self, path='./aclImdb', is_train=True):
        self.reviews, self.labels = [], []
        tokenizer = get_tokenizer('basic_english')
        for label in ['pos', 'neg']:
            folder_name = os.path.join(path, 'train' if is_train else 'test', label)
            for filename in os.listdir(folder_name):
                with open(os.path.join(folder_name, filename), mode='r', encoding='utf-8') as f:
                    self.reviews.append(tokenizer(f.read()))
                    self.labels.append(1 if label == 'pos' else 0)

    def build_dataset(self, vocab, max_len=512):
        # 批量加载句子需要统一长度，所以这里我们选择 512 作为标准
        # 长度超过 512 的进行截断，长度不到 512 的使用填充词元 <pad> 进行填充
        text_transform = T.Sequential(
            T.VocabTransform(vocab=vocab),
            T.Truncate(max_seq_len=max_len),
            T.ToTensor(padding_value=vocab['<pad>']),
            T.PadTransform(max_length=max_len, pad_value=vocab['<pad>']),
        )
```

```
        self.dataset = TensorDataset(text_transform(self.reviews),
torch.tensor(self.labels))
```

- BERT 使用 `transformers.BertTokenizer` 作为 tokenizer

```
class DataReader:
    def __init__(self, path='./aclImdb', is_train=True):
        self.reviews, self.labels = [], []
        for label in ['pos', 'neg']:
            folder_name = os.path.join(path, 'train' if is_train else 'test',
label)
            for filename in os.listdir(folder_name):
                with open(os.path.join(folder_name, filename), mode='r',
encoding='utf-8') as f:
                    self.reviews.append(f.read())
                    self.labels.append(1 if label == 'pos' else 0)
        self.data = pd.DataFrame({'reviews': self.reviews, 'labels':
self.labels})

tokenizer = BertTokenizer.from_pretrained('bert-base-uncased',
do_lower_case=True)

def convert_to_bert_inputs(data, labels):
    // ...
    return TensorDataset(input_ids, attention_masks, token_type_ids, labels)
```

2.2 实验过程说明

按照 8:2 的比例将训练集划分为训练集和验证集，然后分别构建含 RNN 和 BERT 网络的神经网络，之后就是 pytorch 的经典流程：训练、验证、调参、测试。训练的 loss 选用交叉熵损失（CrossEntropyLoss）。

2.2.1 RNN(LSTM) 模型

```
class SentimentClassifierRNN(nn.Module):
    def __init__(self, vocab_size, embed_dim, num_class, device):
        super().__init__()
        self.embedding = nn.Embedding(vocab_size, embed_dim)
        self.encoder = nn.LSTM(embed_dim, embed_dim, batch_first=True)
        self.decoder = nn.Linear(embed_dim, num_class)
        self.device = device

    def forward(self, x):
        x = self.embedding(x)
        x, _ = self.encoder(x)
        x = x[:, -1, :]
        x = self.decoder(x)
        return x
```

这个网络包括嵌入层、encoder 和 decoder。其中，嵌入层将输入的单词转换为词向量表示，encoder 使用 LSTM 模型对嵌入层的词向量进行编码，decoder 将编码后的最后一个时间步输出的状态向量映射到情感类别的分数。

2.2.2 BERT 模型

```
class SentimentClassifierBERT(nn.Module):
    def __init__(self, num_labels):
        super(SentimentClassifierBERT, self).__init__()
        self.bert = BertModel.from_pretrained('bert-base-uncased')
        self.dropout = nn.Dropout(0.1)
        self.fc = nn.Linear(self.bert.config.hidden_size, num_labels)

    def forward(self, input_ids, attention_mask, token_type_ids):
        outputs = self.bert(input_ids=input_ids,
                            attention_mask=attention_mask,
                            token_type_ids=token_type_ids)
        pooled_output = outputs[1]
        pooled_output = self.dropout(pooled_output)
        logits = self.fc(pooled_output)
        return logits
```

这个网络使用 `BertModel.from_pretrained()` 加载预训练的 BERT 模型。dropout 层 p 为 0.1，防止过拟合。大小为 `num_labels` 的线性层将产生最终的输出。

`forward` 方法接受三个参数 `input_ids`、`attention_mask` 和 `token_type_ids`，这些是 BERT 模型的输入。在该方法中，我们将输入传递到 BERT 模型，并获得输出。然后，我们取池化输出，即 `[CLS]` 标记的输出，并将其应用于 dropout 层。最后，输出传递到线性层，以获取 logits，这些是分配给每个类别的分数。

3 超参数的调节

本次实验中，我们主要调节了 RNN 和 BERT 两个模型的超参数，包括 Epochs 和 BatchSize。

3.1 RNN 超参数调节

在 RNN 中，我们将 Epochs 设为15，BatchSize 设置为64。在实验中，发现过多的 Epochs 会导致过拟合，而过大的 BatchSize 则会导致收敛速度变慢，因此需要适当调节这两个参数。最终的结果表明，经过调节后，RNN 模型的性能得到了提升。

3.2 BERT 超参数调节

在 BERT 中，我们将 Epochs 设为5，BatchSize 设置为64。在实验中，发现 BERT 模型的训练时间比 RNN 模型更长，因此需要适当调节 Epochs 和 BatchSize 以提高训练速度。最终的结果表明，经过调节后，BERT 模型的性能得到了提升。

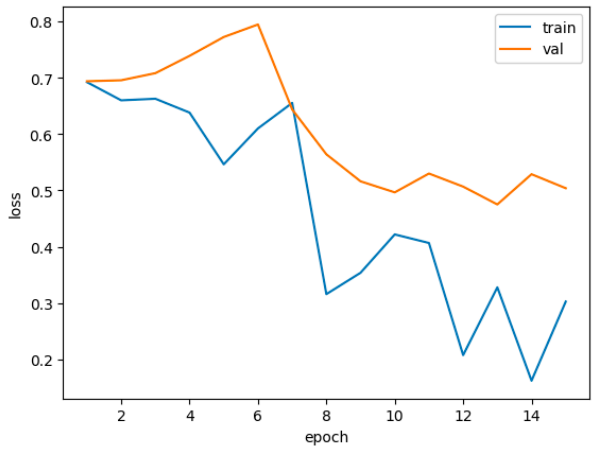
4 最终参数及测试

经过调参，最终选定的参数如下：

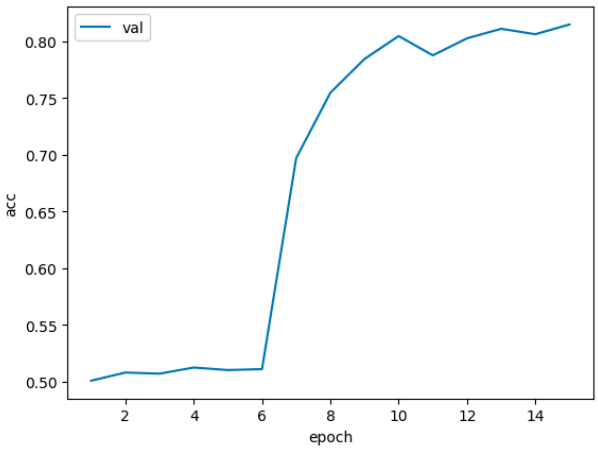
- RNN，Epochs = 15, BatchSize = 64
- BERT，Epochs = 5, BatchSize = 64

4.1 RNN 结果

随着 Epoch 的步进，Train 和 Val 上的 loss 变化如下图所示：



Val 上的 Acc 变化如下图所示：



4.2 BERT 结果

随着 Epoch 的步进，Train 和 Val 上的 loss、Val 上的 Acc 变化如下表所示：

Epoch	Train loss	Val loss	Val Acc
1	0.2572	0.1961	0.9278
2	0.1244	0.2041	0.9294
3	0.0669	0.2297	0.9300
4	0.0442	0.2580	0.9294
5	0.0273	0.2817	0.9256

可以发现 Epochs = 2 时效果最佳（综合 Val loss 和 Val Acc）。

4.3 测试集上结果与分析

使用上述训练的模型对测试集进行预测，结果如下：

- RNN：在测试集上的 loss 和 Acc 分别是 (0.525, 0.811)
- BERT：在测试集上的 loss 和 Acc 分别是 (0.206, 0.926)

根据测试集上的结果，可以看出 BERT 模型在情感分类任务上的表现优于 RNN 模型，其准确率提高了 11.5 个百分点，而且损失值也下降了很多。这是因为 BERT 模型具有以下优点：

- BERT 模型使用了 Transformer 结构，能够更好地捕捉上下文信息，尤其是长距离依赖关系，因此在处理有关文本的任务时比 RNN 表现更好
- BERT 模型是预训练的，可以使用更大的数据集进行训练，因此它具有更强的泛化能力
- BERT 模型使用了 Masked Language Model 和 Next Sentence Prediction 等任务进行预训练，使得其对于语言的理解更加充分

不过，尽管 BERT 在精度上的表现很好，实验过程中还是发现 BERT 相比于 RNN 具有如下的不足：

- BERT 模型需要很长的训练时间，因为它需要训练多个层次的 Transformer 结构
- BERT 模型需要大量的计算资源，因为它有很多参数，需要在大型 GPU 上进行训练