中国科学技术大学

University of Science and Technology of China

深度学习导论

实验 3

姓名 徐海阳　　学号 PB20000326　　院系 少年班学院

摘　要

本实验使用 pytorch 神经网络库编写基于 BERT 的预训练语言模型，利用少量的训练数据，微调模型用于文本情感分类；并和直接用 RNN 训练的文本分类器进行对比，研究训练数据量变化对性能带来的影响。发现 BERT 不容易出现过拟合，并且只需少量 epoch(3 个) 进行微调即可大幅优于 RNN 模型 ($93+\% \gg 70-\%$)。

目录

1 数据集与数据预处理

1.1 数据集简介

IMDB（Internet Movie Database）是一个广泛使用的电影数据库，而”Large Movie Review Dataset” 则是从 IMDB 中提取出来的一个公开数据集，用于情感分析任务。”Large Movie Review Dataset” 包含了 25,000 条电影评论，用于训练和测试情感分类模型。数据集被平均分为两个部分：训练集和测试集。每个部分包含了相等数量的正面和负面评论。数据集已经经过预处理和标记，使得每个评论都与一个情感类别相关联。

数据集中的评论是来自 IMDB 的用户对电影的个人评论。评论涵盖了各种类型的电影，包括喜剧、动作、剧情等。每个评论都有一个情感标签，表示评论者对电影的情感倾向。其中，正面评论被标记为 1，表示积极情感，而负面评论被标记为 0，表示消极情感。以下是一些示例数据，展示了数据集中的评论及其对应的情感标签：

1. 正面评论示例：- 评论：*I absolutely loved this movie! The acting was brilliant and the plot kept me engaged throughout. Highly recommended.* - 情感标签：1

2. 负面评论示例：- 评论：*This movie was a complete waste of time. The acting was terrible and the storyline was confusing. I would not recommend it.* - 情感标签：0

这些示例说明了数据集中的评论可以涵盖不同的观点和情感倾向，对于情感分类任务具有一定的挑战性。

1.2 数据预处理

本次实验用到了 2 种 Dataset 获取方式。第 1 种可以简单处理 BERT 所需数据，第 2 种可以深度自定义来处理 RNN 的数据。

第 1 种是利用 Huggingface 的 API，如图1所示。可以看到 IMDB 数据集已经在 load_dataset 里封装好了，我们只需要用 pretrain 好的 bert-base-uncased tokenizer 对 dataset 做 map 即可，dataset.map 函数会自动实现 trunction, padding 和 batch 操作。这样处理数据是因为可以更方便地使用 huggingface 的 Trainer 接口来训练 BERT 模型。

```python
from transformers import AutoTokenizer
from datasets import load_dataset

tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
dataset = load_dataset("imdb")
dataset = dataset.map(lambda x: tokenizer(x["text"], truncation=True, padding="max_length"), batched=True)
```

图 1: 利用 Huggingface API 得到封装好的 Dataset

第 2 种是助教提供的手撸 RawData 和 DatasetClass，如图2所示。首先在 RawData 类中定义了读取 IMDB 数据以及获取 token_embedding 的函数，用以编码 RNN 的输入数据；然后在自定义的 DatasetClassRNN 里重写了 ___init___, ___getitem___ 和 ___len___ 方法。

2 语言模型

RNN, LSTM 和 BERT 在上课都有介绍，以下再次简明扼要地介绍它们。

```python
class RawData:
    """原始数据静态类"""
    __type = None
    __data = []
    __vocab = None

    @staticmethod
    def read_imdb(type_):  # 读取原始数据，可以读取 训练集或者测试集 …

    @staticmethod
    def get_vocab():  # 利用 word2vec 的方法训练 token embedding …


class DatasetClassRNN(Dataset):
    def __init__(self, type_): …

    def text_pipeline(self, text):
        return [self.vocab[token] for token in self.tokenizer(text)]

    def __getitem__(self, item):
        text, label = self.raw_text[item]
        text_processed = self.text_pipeline(text)
        return text_processed, label

    def __len__(self):
        return len(self.raw_text)
```

图 2: 手撸 RawData 和 DatasetClass

2.1 RNN（Recurrent Neural Network）

RNN 是一种经典的神经网络架构，特别适用于处理序列数据。它通过在每个时间步重复使用相同的神经元，将先前的信息传递给后续步骤，从而捕捉序列的上下文信息。RNN 的结构包括一个循环的隐藏状态，它在每个时间步接收输入和前一个时间步的隐藏状态，并输出当前时间步的隐藏状态。然后，可以在最后一个时间步或每个时间步上使用输出进行相应的任务。它存在的问题是对于长序列的处理能力较弱，可能会遇到梯度消失或梯度爆炸的问题。

2.2 LSTM（Long Short-Term Memory）

LSTM 是一种 RNN 的变体，旨在解决传统 RNN 中的长期依赖问题。长期依赖问题是指在处理长序列时，传统 RNN 由于梯度消失或梯度爆炸的原因无法有效地传播信息。LSTM 通过引入门控机制，即遗忘门、输入门和输出门，来控制隐藏状态中的信息流动。遗忘门决定从前一个时间步传递哪些信息，输入门决定添加哪些新信息，输出门决定输出什么信息。这种门控机制使 LSTM 能够更好地捕捉和记忆长期依赖关系。它的缺点是相比于传统 RNN，LSTM 的计算复杂度较高，需要计算门控单元；同时，对于超长序列还是会出现“遗忘“现象。

2.3 BERT（Bidirectional Encoder Representations from Transformers）

BERT 是一种预训练的语言模型，基于 Transformer 架构。BERT 采用了 Transformer 中的自注意力机制，使得模型能够同时考虑到输入序列中的前后上下文信息。BERT 的核心思想是通过在大规模文本语料上进行无监督预训练，学习到丰富的语言表示。具体来说，BERT 模型是使用了包含了英文维基百科和 BookCorpus 等多个数据集的语料库进行预训练的。这些语料库包含了大量的文本数据，覆盖了各种不同的主题和领域。然后，通过微调（fine-tuning）BERT 模型，可以在特定任务上获得优秀的性能。BERT 的特点是无监督学习，能够同时利用上下文信息，使得模型在处理自然语言处理任务时取得了显著的进展。它的缺点是模型规模较大，需要大量的计算资源和时间进

行训练。

3 情感分类任务

情感分类模型是用于将文本进行情感分类的模型，通常可以将文本分为正面情感和负面情感两类，有时也可以进一步细分为多个情感类别。具体可参考1.1中对 IMDB 数据集的描述。

3.1 RNN 和 LSTM 模型用于情感分类任务

```
295         # 定义网络
296         self.net = RNN(self.model_type,
297                        self.vocab_size,
298                        self.embed_size,
299                        self.num_class,
300                        self.hidden_size,
301                        self.num_layers).to(self.device)
302 
303         if self.fix_embedding:
304             for k, v in self.net.named_parameters():
305                 if k == "embedding.weight":
306                     v.requires_grad = False
307 
308         self.optimizer = optim.Adam(filter(lambda p: p.requires_grad, self.net.parameters()),
309                                     lr=self.lr)
310 
311         total_params = sum([param.nelement() for param in self.net.parameters() if param.requires_grad])
312         print(f">>> model type: {self.model_type}, train rate: {train_rate}")
313         print(f">>> total parameters: {total_params}")
314         patience = 0
315         best_val_acc = 0.
316         best_train_acc = 0.
317         for epoch in range(self.epochs):
318             t1 = datetime.now()
319             for data in tqdm(train_loader):
320                 text_tensor, label_tensor, offsets_tensor = data
321                 text_tensor = text_tensor.to(self.device)
322                 label_tensor = label_tensor.to(self.device)
323                 offsets_tensor = offsets_tensor.to(self.device)
324                 predict_label = self.net(text_tensor, offsets_tensor)
325                 self.optimizer.zero_grad()
326                 loss = criterion(predict_label, label_tensor)
327                 loss.backward()
328                 self.optimizer.step()
```

图 3: RNN 模型输入输出

如图3所示，RNN 和 LSTM 模型可以将每个单词的词向量作为输入，然后直接将输出作为情感分类的预测。line 296 定义了 RNN 模型，line 324 可以看到直接将输入的词向量过 RNN 模型得到的输出就是预测的分类结果。

3.2 BERT 模型用于情感分类任务

而 BERT 模型不同，我们直接将 BERT 作为一个预训练的语言模型，将文本序列输入 BERT 模型中，获取文本的语义表示，然后将该表示输入一个分类器（如 MLP）进行微调训练。如图4所示，line 47 定义了 BERT 模型，line 48 定义了一个 MLP；在 forward 函数 line 52 可以看到，我们将经过 BERT 得到的 embedding 池化后过 MLP。

```python
class BERT(nn.Module):
    def __init__(self):
        super().__init__()
        self.model = BertModel.from_pretrained("bert-base-uncased")
        self.classifier = nn.Linear(768, 2)

    def forward(self, input_ids, attention_mask, labels=None):
        outputs = self.model(input_ids=input_ids, attention_mask=attention_mask)
        pooled_output = outputs.last_hidden_state[:, 0]
        logits = self.classifier(pooled_output)
        loss = None
        if labels is not None:
            criterion = nn.CrossEntropyLoss()
            loss = criterion(logits.view(-1, 2), labels.view(-1))
        return SequenceClassifierOutput(
            loss=loss,
            logits=logits,
            hidden_states=outputs.hidden_states,
            attentions=outputs.attentions,
        )
```

图 4: BERT 模型输入输出

4 实验结果

本实验的 RNN 和 LSTM 是助教给的 demo 的 copy-paste，BERT 采用 huggingface 的 pipeline 和 API 实现。其中，RNN 和 LSTM 模型较小，训练了 10 个 epoch；BERT 模型较大，训练了 5 个 epoch。它们训练的 batchsize 都是 16；RNN 和 LSTM 模型使用了 Adam 优化器，学习率为 1e-3；BERT 模型使用了 AdamW 优化器，学习率为 5e-5。以下是不同模型的性能分析以及数据规模对于性能的影响。

模型	数据规模 (%)	验证集正确率 (%) ↑
RNN	25	62.4
RNN	50	67.9
RNN	**75**	**69.8**
RNN	100	67.6
LSTM	25	83.8
LSTM	50	88.6
LSTM	**75**	**89.2**
LSTM	100	89.1
BERT	25	90.4
BERT	50	92.0
BERT	75	93.0
BERT	**100**	**93.5**

表 1: 不同语言模型在不同数据规模下在验证集上的正确率

首先，我们观察表1中可以看到 BERT 模型在所有数据比例下都展现出最高的正确率，其次是 LSTM，最后是 RNN。

然后，可以观察到，RNN 和 LSTM 模型在数据比例为 75% 时取得了最好的正确率，随后正确率有所下降。这可能是因为在数据量较少时，RNN 和 LSTM 欠拟合，在验证集上表现较好。然而，

当数据量超过 75% 时，RNN 和 LSTM 过拟合了，因而正确率下降。

然而，从 25% 到 100% 的数据量逐渐增加，BERT 的正确率一直提升。这应该是因为大模型的参数量更大，表征能力更强，更不容易过拟合，因此它在当前数据规模下（100% 是 25000 条）始终可以学到更加丰富的表征。

接下来，我们绘制这三种模型在不同数据规模下的训练集/验证集损失曲线来进一步分析。图中红色，绿色，黄色，蓝色分别代表数据规模 100%, 75%, 50%, 25%,；虚线代表在训练集上的结果，实线代表在验证集上的结果。

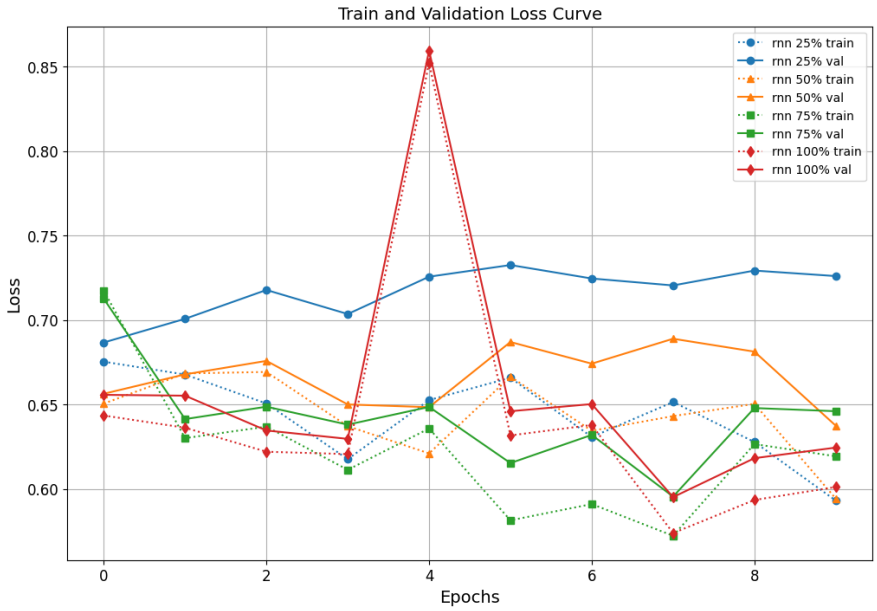

图 5: RNN 模型的损失曲线

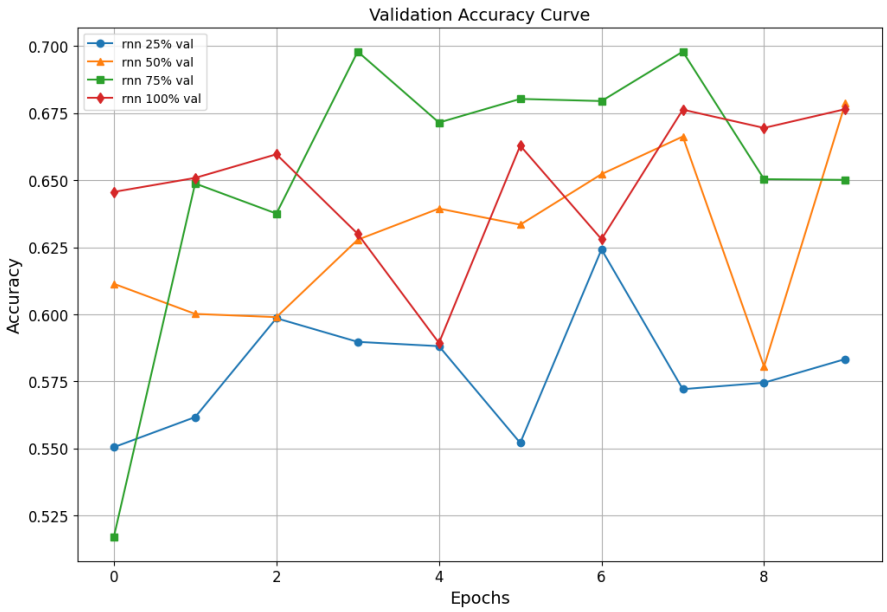

图 6: RNN 模型的验证准确率曲线

图5是 RNN 模型的损失曲线可视化，图6是 RNN 模型的验证准确率曲线。可以看到 RNN 模型较难训练好，甚至连 train loss 在 epoch=7 处就已经不再下降，更别提 validation loss 基本上一直

在震荡，下降得很小。在验证集上的准确率也是一直在震荡，效果不佳。通过观察数据集可知，本数据集的 review 语句大多是长句，这也印证了 RNN 模型对于长序列的处理能力较弱。RNN 模型在 2 分类问题上的正确率不到 70%，效果不佳。

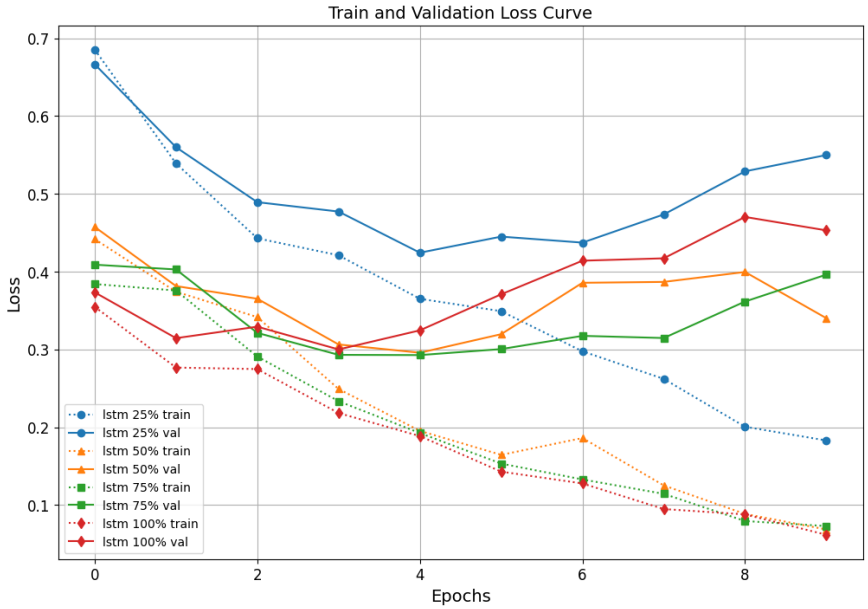

图 7: LSTM 模型的损失曲线

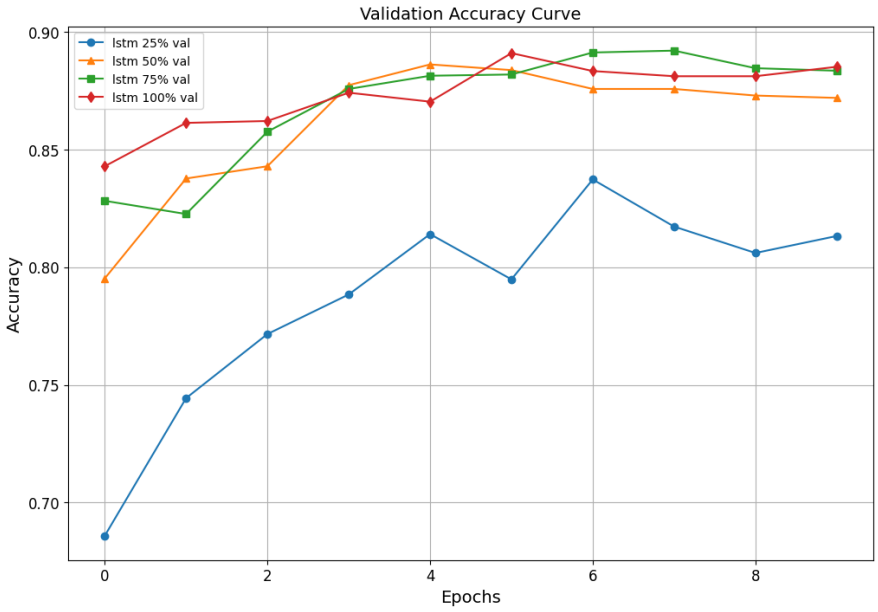

图 8: LSTM 模型的验证准确率曲线

图7是 LSTM 模型的损失曲线可视化，图8是 LSTM 模型的验证准确率曲线。从损失曲线可以明显看出过拟合：train loss 一直下降，而 validation loss 在后几个 epoch 开始上升。这不难解释，因为 LSTM 的参数量还是不够大，因此在较大规模训练数据集下容易发生过拟合。同时，可以看出 LSTM 模型在数据规模为 100% 和 75% 时的准确率其实差不太多，数据规模为 100% 时略低一些；但是数据规模为 100% 时验证集的 loss 更大一些，这也说明数据规模大的时候会出现过拟合。LSTM 模型在 2 分类问题上的正确率接近 90%，效果不错。

图9是 BERT 模型的损失曲线可视化，图10是 BERT 模型的验证准确率曲线。从损失曲线可以

图 9: BERT 模型的损失曲线

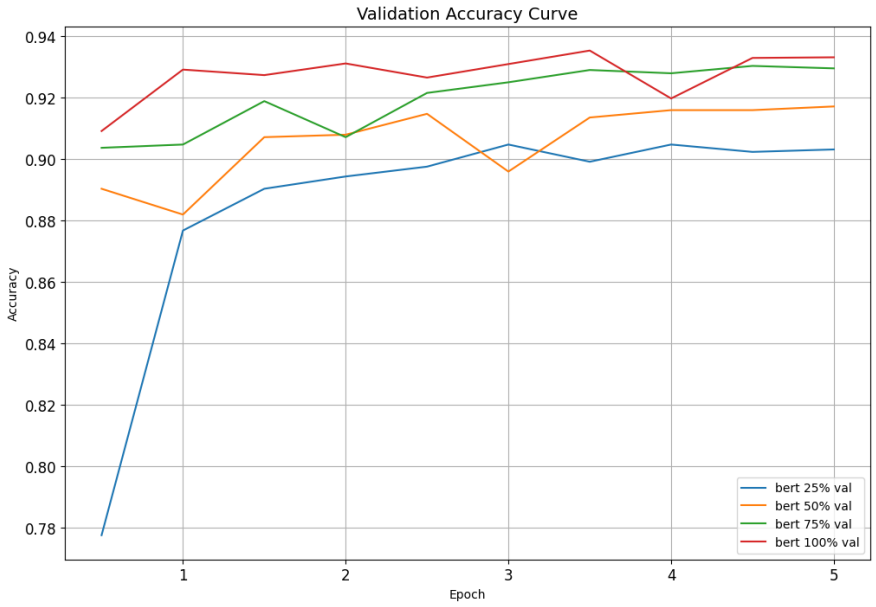

图 10: BERT 模型的验证准确率曲线

明显看 BERT 事实上只需要训练 3 个 epoch：在大于 3 个 epoch 时，train loss 非常接近于 0，而 validation loss 也逐步上升；从验证准确率曲线也可以看出，在第 3 个 epoch 之后逐渐震荡，没有明显提升。这说明 BERT 在预训练时学到的特征非常丰富及全面，只需要稍稍微调即可在下游任务上取得很好的性能。BERT 模型在 2 分类问题上的正确率超过 90%，效果很好。

事实上，本实验也尝试了 Transformer 模型的训练，是直接用 Transformer Encoder 进行训练。发现效果较差，一直是 50%。猜测原因是因为 Transformer 模型太大，小数据无法支撑训练；BERT 虽然也是 Transformer 模型的一种，但它已经在大量文本上进行预训练，文本表征能力很强，因此在下游微调时较为顺利。