利用 BERT 语言模型实现文本分类

姓名：陈泽豪　学号：SA22001009

2022 年 12 月 31 日

摘要

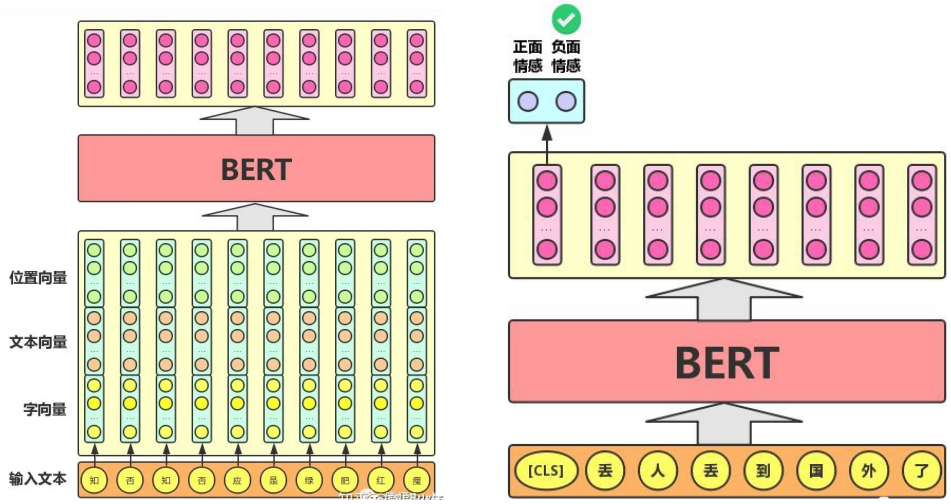
本报告首先介绍一下有关本次实验的大体实现框架，以及对 BERT 模型的介绍，然后给出简要的实验过程以及对超参数的实验分析，模型在测试集上的测试结果等。

一、BERT 网络介绍

1.1 BERT 的输入与输出

首先我们介绍一下有关 BERT 的相关内容，也方便笔者自己对这个网络有更深的理解。BERT 全称为 BidirectionalEncoder Representations from Transformer，因此 BERT 模型的目标是利用大规模无标注语料训练，获得文本的包含丰富语料信息的表示，更简单的去表达这个意思，就是获得文本的语义表示，然后利用这个语义表示去做进一步的 NLP 任务，例如我们这次的语言情感分类。

经过了第二次 RNN 的实验，我们了解到了一般的句子都会将句子中的词语用词向量进行表示，然后将词向量送入 RNN 网络中得到句子的一个向量表征，最后将这个向量表征输入到分类器中输出一个二维情感分类。在 BERT 中也是同样的思想，我们首先输入的就是一个句子的词向量表示，再经过了 BERT 后输出这个句子（也即文本）的多维向量表征。我们最后取出 BERT 的 *pooler_out* 输出作为分类器的输入。



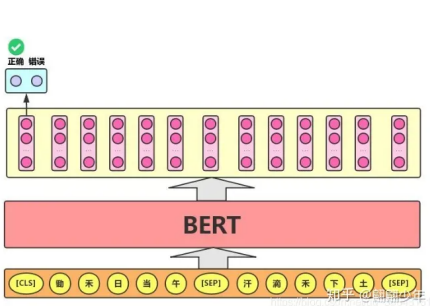
(a) bert 流程图　　(b) 利用 bert 去做语言情感分类

1.2 BERT 的预训练任务

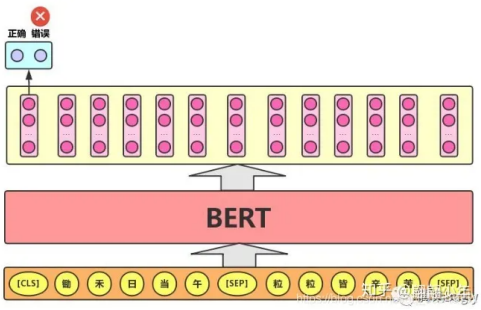
在本次实验中我们是直接使用了 BERT 的预训练模型作为网络结构之一，因此在这里稍微列出这个 BERT 模型具体的预训练任务都有哪些。它的预训练过程就是逐渐调整模型参数，使得模型输出的文本语义表示能够刻画语言的本质，便于后续针对具体 NLP 任务作微调。为了达成这个目的，原文的作者采用了两个预训练的任务：Masked LM 和 Next Sentence Prediction。

Masked LM 的任务就是当给定了一句话，擦去其中的几个单词，如何根据剩余词汇来预测这些被擦去的单词是什么，这么做的目的就是迫使整个模型会更多的去依赖上下文的关系，使得最终用于微调的模型更加的精确。

Next Sentence Prediction 的任务就是给定了前后两个语句，判断这两句是前后语义相连的两个句子，这个任务使得模型更加注重整体的语意连贯性，也进一步加强了模型的表达能力。因此在经过了这两个任务之后，后续 NLP 任务可以获得更好的初始化条件。



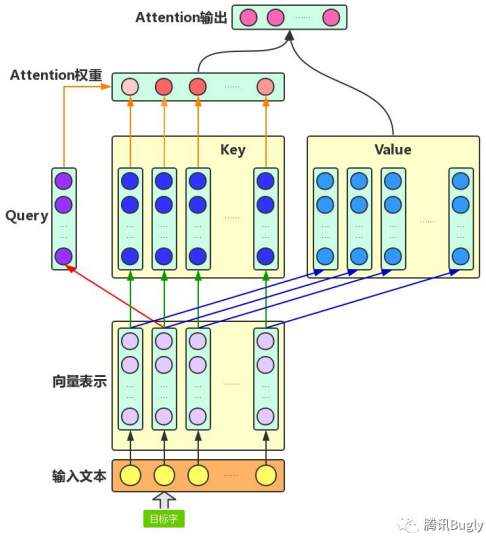
(c) Masked LM



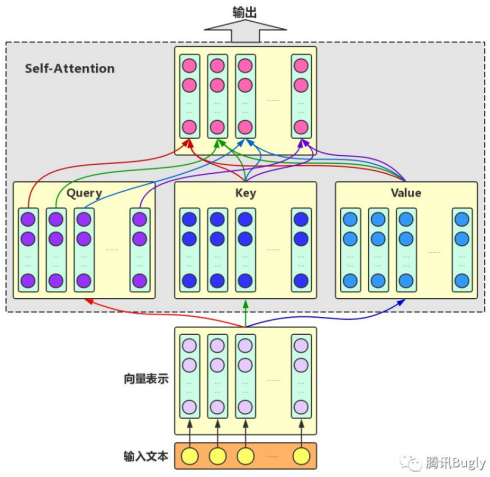
(d) Next Sentence Prediction

1.3 BERT 的 Attention 机制

注意力机制，就是希望神经网络将一部分的注意力放在输入上，区别输入的不同部分对于这个部分的影响，因为对于文本来说，任取它其中的一个字段，都应当结合上下文来确认别的字段对这一字段的影响程度大小。从这个角度出发，注意力机制涉及到了三个概念：Query，Key 与 Value。注意力机制将目标的字作为 Query，将它的上下文作为 Key，将 Query 与 Key 的相似性作为权重与 Value 做加权，得到结合了上下文的这个字的增强语义词向量。

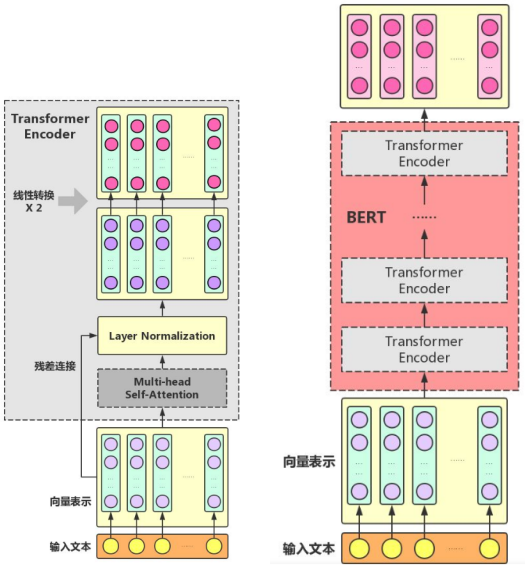


(e) 单个字输出



(f) 一句话的输出

在得到了整个自注意力机制的流程之后，再根据句子在不同语义空间下的结果得到不同的 Query，Key 与 Value 值，然后根据所有不同的输出去做变换得到最终的多重自注意力机制的输出。有了多重自注意力模型，就可以将它放在整个 BERT 的架构中作为某一个中间层，多层堆叠之后就得到了最终的 BERT 模型，具体的图示如下图 (g,h) 所示：



(g) Transformer Encoder (h) BERT

二、实验配置

本次实验通过 Anaconda 搭建出了虚拟环境，整个代码在 python 3.9.15，cuda 11.3.1，numpy 1.23.4，pytorch 1.12.1，torchtext 0.13.1 环境下进行实验，GPU 使用的是 NVIDIA Geforce RTX 3060。

三、对数据集的处理

下面简要介绍一下本次实验如何对数据集进行处理，这里主要的处理方式就是利用 torchtext 库内的函数去对整个数据集进行处理，主要代码参考了网络上的公开代码。与上一次的处理不同，这一次需要将 IMDB 词库中的句子处理为适合 BERT 的输入形式，因此首先需要得到我们所用的 BERT 模型的词汇表，由这个词汇表来得到每个句子中单词在那张词汇表中的 *index*，同时需要在句子的前端加入开始的单词 [*CLS*]，在结尾加入结束的单词 [*SEP*]，具体的实现如下所示，首先需要从路径 *trained_bert_vocab* 中读取当前的 BERT 模型的 Vocab 词表：

```
# 从bert模型中读取字典
def load_acimdb(trained_bert_vocab):
review_train_list, label_train_list = read_acimbd(is_train=True)
review_test_list, label_test_list = read_acimbd(is_train=False)

with open(trained_bert_vocab, 'r', encoding='utf-8') as vocab_file:
token_list = []
```

```
for token in vocab_file.readlines():
token_list.append(token.strip())
_vocab = vocab(OrderedDict([(token, 1) for token in token_list]))
# 设置未登录词的索引
_vocab.set_default_index(_vocab['[UNK]'])

dataset_train = build_dataset(review_train_list, label_train_list, _vocab=_vocab)
dataset_test = build_dataset(review_test_list, label_test_list, _vocab=_vocab)
return dataset_train, dataset_test, _vocab
```

然后就是进行上一步提到过的对句子两端的处理，以表示句子的开头与结尾：

```
# 将review_list与label_list经过vocab的index转化后用TensorDataset打包
def build_dataset(review_list, label_list, _vocab, max_len=256):
# 建立一个词表转化,vocab里存储词汇与它的唯一标签，利用VocabTransform将词汇转化为对应的数字
# 利用Truncate将所有的句子的最长长度限制在了max_len
# 利用ToTensor将所有的句子按照此时最长的句子进行填充为张量，填充的内容为'[PAD]'对应的数字
# 利用PadTransform将ToTensor里所有的句子长度均填充为max_len
# 与第二次实验不同的点是要将句子的开头与结尾加入开始[CLS]以及结束[SEP],
# 需要按照bert模型的vocab创建
seq_to_tensor = T.Sequential(
T.VocabTransform(vocab=_vocab),
T.Truncate(max_seq_len=max_len-2),
T.AddToken(token=_vocab['[CLS]'], begin=True),
T.AddToken(token=_vocab['[SEP]'], begin=False),
T.ToTensor(padding_value=_vocab['[PAD]']),
T.PadTransform(max_length=max_len, pad_value=_vocab['[PAD]'])
)
dataset = TensorDataset(seq_to_tensor(review_list), torch.tensor(label_list))
return dataset
```

四、实验过程

本次实验的内容包括对 BERT 网络的调整以及与上一次 LSTM 实验结果的对比，下面我们先给出在本次实验中搭建的 BERT 网络模型：

```
class MyBERT(nn.Module):
def __init__(self, vocab, bert_model):
super().__init__()
self.vocab = vocab
self.bert = BertModel.from_pretrained(bert_model)

self.bert_config = self.bert.config
out_dim = self.bert_config.hidden_size
self.classifier = nn.Linear(out_dim, 2)

self.mlp = nn.Sequential(
nn.Linear(out_dim, 500),
nn.ReLU(),
nn.Linear(500, 300),
nn.ReLU(),
nn.Linear(300, 2)
)

def forward(self, input_ids):
attention_mask = (input_ids != self.vocab['[PAD]']).long().float()
output = self.bert(input_ids=input_ids, attention_mask=attention_mask)
result = self.classifier(output.pooler_output)
return result
```

代码中的初始化输入 *bert_model* 方便我们修改使用的预训练 BERT 模型。接下来我们利用 BERT 的预训练模型在给定超参数条件下的实验结果，与上一次实验利用 LSTM 的实验结果进行对比。首先选择 BERT 模型为 $bert-base-uncased$，同时由于本电脑 GPU 内存的限制，在选用此 BERT 模型的情况下只能将最大的 *batch_size* 设为 18，设置 *learning_rate* 为 $1e-4$，设置 *epoch_num* 为 3 时得到的结果为：

epoch_num	Average loss
1	0.313163
2	0.147865
3	0.031142

表 1: 实验中平均交叉熵误差随训练次数的变化表格

最终在测试集上的准确率为 90.876%。整个实验的参数设置与结果记录如上所述，下面我们就实验中可能会影响结果的变量进行调整。

五、超参数测试

5.1 不同模型不同学习率的调整

这一次需要进行测试的主要就是超参数学习率的调整是否会影响最终的结果以及不同 BERT 模型的选择是否会影响最终的结果，下面我们选择了两种不同的模型分别为 $bert-base-uncased$，$bert-base-cased$ 做了不同学习率下的测试，下面依次给出这三个的实验结果。

首先选用了 $bert-base-uncased$，得到的实验结果如下所示：

$epoch\_num$	$ln\_rate:1e-5$	$ln\_rate:5e-5$	$ln\_rate:1e-4$
1	0.25583	0.264109	0.313163
2	0.115106	0.090546	0.147865
3	0.037858	0.014494	0.031142
$Acc\_on\_test(\%)$	91.736	92.004	90.870

表 2: $bert-base-uncased$ 模型在不同学习率下的误差变化以及在测试集上的正确率

然后我们选择 $bert-base-cased$，得到的实验结果如下：

$epoch\_num$	$ln\_rate:1e-5$	$ln\_rate:5e-5$	$ln\_rate:1e-4$
1	0.27086	0.281609	0.352405
2	0.120607	0.108770	0.197969
3	0.034534	0.014477	0.067858
$Acc\_on\_test(\%)$	91.352	91.604	89.708

表 3: $bert-base-cased$ 模型在不同学习率下的误差变化以及在测试集上的正确率

因此根据上面两个模型的选择结果，我们最终使用模型 $bert-base-uncased$ 以及学习率 $5e-5$ 进行实验，下面需要观察提高训练次数是否可以进一步的提高正确率。

5.2 提高训练次数对结果的影响

下面我们分别对上面选择的模型训练 4 次的结果来反映测试集上正确率的变化是否随训练次数还可以显著提高，结果如下表 4 所示。

可以发现即使我们再多训练一轮，它的在测试集上的准确率也不会发生很大的变化了，因此实际上不需要训练较多的次数，直接采用训练 3 次的结果即可。因此最终我们得到的模型就是 $bert-base-uncased$ 模型，选择的学习率为 5e-5，训练次数为 3 轮，选择的 $batch\_size$ 为 18，这个模型在测试集上的测试结果准确率为 92.004%。

$epoch\_num$	$ln\_rate : 5e-5$
1	0.265033
2	0.101118
3	0.021260
4	0.003419
$Acc\_on\_test(\%)$	92.068

表 4: $bert-base-uncased$ 模型的误差变化以及在测试集上的正确率

六、与 RNN 网络的实验对比

在这里我们列出上一次实验最好的实验结果，是利用 LSTM 网络做的实验，实验结果如下，其在测试集上的准确率结果为 89.40%，在训练集上的误差图像如下所示：

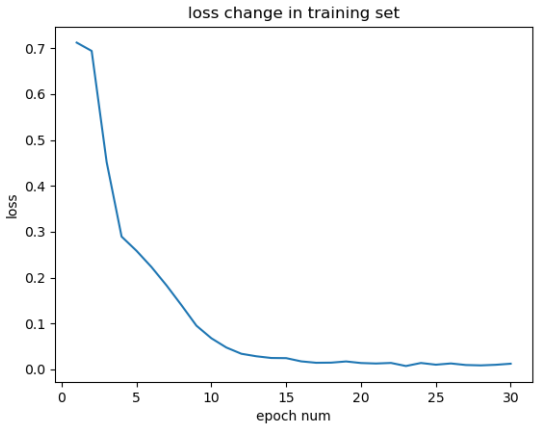


图 1: 最终的测试结果

可以根据最终在测试集上的精度看出采用 BERT 模型微调的方式可以取得比 RNN 网络更好的效果，而且 BERT 网络可以取得更好的效果，因为设备限制无法提高句子的 padding 之后的大小，因此在本次实验中仅选择了句长为 256 进行实验，而且 $batch\_size$ 也无法进一步提高，使得本次 BERT 的实验在精度上只比 LSTM 的方法略有提升。