深度学习 Lab3 实验报告

PB19151769　　马宇骁

1　实验要求

利用实验二的数据编写 BERT 的语言模型，并基于训练好的词向量，利用少量的训练数据，微调 BERT 模型用于文本分类和之前的 RNN 模型进行对比分析。

2　实验原理

实验利用 BERT 模型进行分本分类，因此，先对 BERT 进行总结。

2.1　BERT 简介

BERT 全称为 Bidirectional Encoder Representation from Transformer，是 Google 以无监督的方式利用大量无标注文本「炼成」的语言模型，其架构为 Transformer 中的 Encoder（BERT=Encoder of Transformer）。它强调了不再像以往一样采用传统的单向语言模型或者把两个单向语言模型进行浅层拼接的方法进行预训练，而是采用新的 masked language model（MLM），以致能生成深度的双向语言表征。其有以下两个特点：

1. 这个模型非常的深，12 层，并不宽 (wide)，中间层只有 1024，而之前的 Transformer 模型中间层有 2048。这似乎又印证了计算机图像处理的一个观点——深而窄比浅而宽的模型更好；

2. MLM 同时利用左侧和右侧的词语。

BERT 的作者认为，bi-directional 仍然不能完整地理解整个语句的语义，更好的办法是用上下文全向来预测 [mask]，也就是用“能/实现/语言/表征/../的/模型”，来预测 [mask]。BERT 作者把上下文全向的预测方法，称之为 deep bi-directional。

这个模型的核心是聚焦机制，对于一个语句，可以同时启用多个聚焦点，而不必局限于从前往后的，或者从后往前的，序列串行处理。不仅要正确地选择模型的结构，而且还要正确地训练模型的参数，这样才能保障模型能够准确地理解语句的语义。BERT 用了两个步骤，试图去正确地训练模型的参数。第一个步骤是把一篇文章中，15

然后，用第二个步骤继续训练模型的参数。譬如从上述 1 万篇文章中，挑选 20 万对语句，总共 40 万条语句。挑选语句对的时候，其中 210 万对语句，是连续的两条上下文语句，另外 210 万对语句，不是连续的语句。然后让 Transformer 模型来识别这 20 万对语句，哪些是连续的，哪些不连续。

这两步训练合在一起，称为预训练 pre-training。

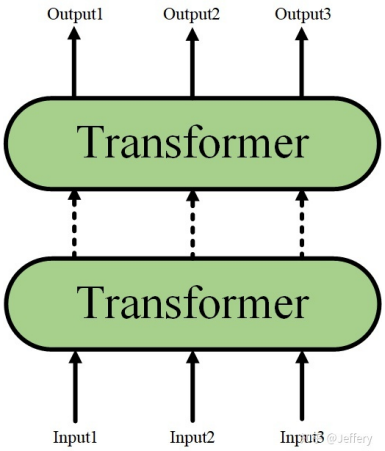

图 1: Transformer

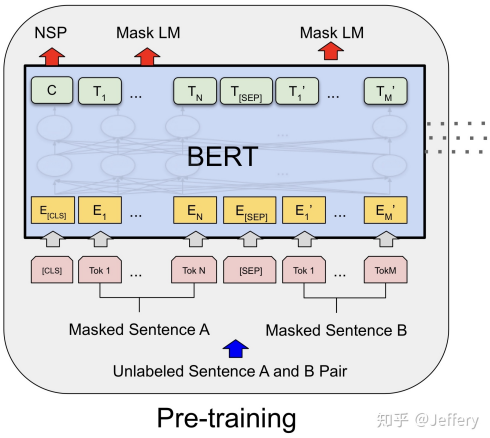

图 2: 预训练

2.2 模型解释

BERT 的文本分析原理：

1. 在序列 tokens 中把分割 token（[SEP]）插入到每个句子后，以分开不同的句子 tokens。
2. 为每一个 token 表征都添加一个可学习的分割 embedding 来指示其属于句子 A 还是句子 B。

2.2.1 输入

BERT 的输入为每一个 token 对应的表征，实际上该表征是由三部分组成的，分别是对应的 token，分割和位置 embeddings:

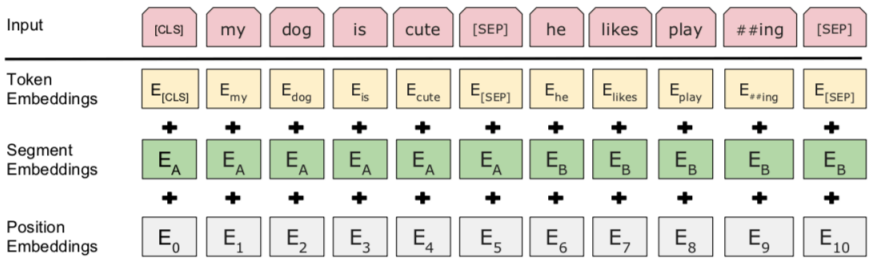

图 3: Embedding

- Token Embedding 就是正常的词向量，即 PyTorch 中的 nn.Embedding()
- Segment Embedding 的作用是用 embedding 的信息让模型分开上下句，我们给上句的 token 全 0，下句的 token 全 1，让模型得以判断上下句的起止位置，例如

```
[CLS]我的狗很可爱[SEP]企鹅不擅长飞行[SEP]
 0    0 0 0 0 0 0 0  1 1 1 1 1 1 1 1
```

- Position Embedding 和 Transformer 中的不一样，而是学习出来的

2.2.2 输出

Transformer 的特点就是有多少个输入就有多少个对应的输出。C 为分类 token（[CLS]）对应最后一个 Transformer 的输出，T_i 则代表其他 token 对应最后一个 Transformer 的输出。对于一些 token 级别的任务（如，序列标注和问答任务），就把 T_i 输入到额外的输出层中进行预测。对于一些句子级别的任务（如，自然语言推断和情感分类任务），就把 C 输入到额外的输出层中，这里也就解释了为什么要在每一个 token 序列前都要插入特定的分类 token。

3 代码完成及分析

3.1 代码完成流程

3.1.1 环境

通过在 Anaconda 中创建 Python3.7.13，主要搭建并使用 torch1.11.0+cu113，tensorflow-gpu 2.6.0 和 transformers 进行训练学习。

3.1.2 数据读取

利用 Lab2 中的 imdb 数据集，将训练集和测试集（train，test）各 25000 中的正向和负向评价（pos，neg）各 12500 分别读入并加入标签（1, 0）。

然后将读入的数据进行分词和编码，把一个句子分成一个一个的单词（一个一个的 token，包括标点）。因为神经网络接受的不可能是单词，肯定是一些数字。所以要把这些单词编码成数字（tokens to ids)。将其中一个结果输出查看如下：

Original: Bromwell High is a cartoon comedy. It ran at the same time as some other programs about school life, such as "Teachers". My 35 years in the teaching profession lead me to believe that Bromwell High's satire is much closer to reality than is "Teachers". The scramble to survive financially, the insightful students who can see right through their pathetic teachers' pomp, the pettiness of the whole situation, all remind me of the schools I knew and their students. When I saw the episode in which a student repeatedly tried to burn down the school, I immediately recalled ..... at ..... High. A classic line: INSPECTOR: I'm here to sack one of your teachers. STUDENT: Welcome to Bromwell High. I expect that many adults of my age think that Bromwell High is far fetched. What a pity that it isn't! Tokenized: ['bro', '##m', '##well', 'high', 'is', 'a', 'cartoon', 'comedy', '.', 'it', 'ran', 'at', 'the', 'same', 'time', 'as', 'some', 'other', 'programs', 'about', 'school', 'life', ',', 'such', 'as', '"', 'teachers', '"', '.', 'my', '35', 'years', 'in', 'the', 'teaching', 'profession', 'lead', 'me', 'to', 'believe', 'that', 'bro', '##m', '##well', 'high', "'", 's', 'satire', 'is', 'much', 'closer', 'to', 'reality', 'than', 'is', '"', 'teachers', '"', '.', 'the', 'scramble', 'to', 'survive', 'financially', ',', 'the', 'insight', '##ful', 'students', 'who', 'can', 'see', 'right', 'through', 'their', 'pathetic', 'teachers', "'", 'po', '##mp', ',', 'the', 'pet', '##tine', '##ss', 'of', 'the', 'whole', 'situation', ',', 'all', 'remind', 'me', 'of', 'the', 'schools', 'i', 'knew', 'and', 'their', 'students', '.', 'when', 'i', 'saw', 'the', 'episode', 'in', 'which', 'a', 'student', 'repeatedly', 'tried', 'to', 'burn', 'down', 'the', 'school', ',', 'i',

'immediately', 'recalled', '.', '.', '.', '.', '.', '.', '.', '.', '.', '.', 'at', '.', '.', '.', '.', '.', '.', '.', '.', '.', '.', '.', 'high', '.', 'a', 'classic', 'line', ':', 'inspector', ':', 'i', "'", 'm', 'here', 'to', 'sack', 'one', 'of', 'your', 'teachers', '.', 'student', ':', 'welcome', 'to', 'bro', '##m', '##well', 'high', '.', 'i', 'expect', 'that', 'many', 'adults', 'of', 'my', 'age', 'think', 'that', 'bro', '##m', '##well', 'high', 'is', 'far', 'fetch', '##ed', '.', 'what', 'a', 'pity', 'that', 'it', 'isn', "'", 't', '!'] Token IDs: [22953, 2213, 4381, 2152, 2003, 1037, 9476, 4038, 1012, 2009, 2743, 2012, 1996, 2168, 2051, 2004, 2070, 2060, 3454, 2055, 2082, 2166, 1010, 2107, 2004, 1000, 5089, 1000, 1012, 2026, 3486, 2086, 1999, 1996, 4252, 9518, 2599, 2033, 2000, 2903, 2008, 22953, 2213, 4381, 2152, 1005, 1055, 18312, 2003, 2172, 3553, 2000, 4507, 2084, 2003, 1000, 5089, 1000, 1012, 1996, 25740, 2000, 5788, 13732, 1010, 1996, 12369, 3993, 2493, 2040, 2064, 2156, 2157, 2083, 2037, 17203, 5089, 1005, 13433, 8737, 1010, 1996, 9004, 10196, 4757, 1997, 1996, 2878, 3663, 1010, 2035, 10825, 2033, 1997, 1996, 2816, 1045, 2354, 1998, 2037, 2493, 1012, 2043, 1045, 2387, 1996, 2792, 1999, 2029, 1037, 3076, 8385, 2699, 2000, 6402, 2091, 1996, 2082, 1010, 1045, 3202, 7383, 1012, 1012, 1012, 1012, 1012, 1012, 1012, 1012, 1012, 2012, 1012, 1012, 1012, 1012, 1012, 1012, 1012, 1012, 1012, 1012, 1012, 2152, 1012, 1037, 4438, 2240, 1024, 7742, 1024, 1045, 1005, 1049, 2182, 2000, 12803, 2028, 1997, 2115, 5089, 1012, 3076, 1024, 6160, 2000, 22953, 2213, 4381, 2152, 1012, 1045, 5987, 2008, 2116, 6001, 1997, 2026, 2287, 2228, 2008, 22953, 2213, 4381, 2152, 2003, 2521, 18584, 2098, 1012, 2054, 1037, 12063, 2008, 2009, 3475, 1005, 1056, 999]

发现已经如同 BERT 介绍案例中的分词和编码的效果。

3.1.3 BERT 处理

设置超参数 MAX_LEN 为 128 作为 BERT 接收的文本的长度。超过的会缩减，不足的会补齐（BERT 允许的最大值是 512）。

```
input_ids = [tokenizer.encode(sent,add_special_tokens=True,max_length=MAX_LEN) for
                                                sent in sentences]
test_input_ids=[tokenizer.encode(sent,add_special_tokens=True,max_length=MAX_LEN)
                                                for sent in test_sentences]
input_ids = pad_sequences(input_ids, maxlen=MAX_LEN, dtype="long",
                          value=0, truncating="post", padding="post")
test_input_ids = pad_sequences(test_input_ids, maxlen=MAX_LEN, dtype="long",
                          value=0, truncating="post", padding="post")
```

其中，input_ids 是转换为 id 后的文本，用 “[SEP]” 符号来分隔两个句子并在每一个样本前加上”[CLS]” 符号.

再处理 mask，通过 ids 中的 token_id 的值是否大于 0 来确定，在一个文本中，如果是 PAD 符号则是 0，否则就是 1，将所有 mask 放入 test_attention_masks.

然后切分数据集和验证集，

```
from sklearn.model_selection import train_test_split

# Use 90% for training and 10% for validation.
train_inputs, validation_inputs, train_labels, validation_labels = train_test_split(
                                                input_ids, labels,
                                            random_state=2020, test_size=0.1)
# Do the same for the masks.
```

```
train_masks, validation_masks, _, _ = train_test_split(attention_masks, labels,
                                                       random_state=2020, test_size=0.1)
```

将整个数据的 0.1 随机分出来作为验证集。

创建数据集和 dataloader 将 batch_size 设置为 16；通过 transformers 中的 BertForSequenceClassification.from_pretrained，并且优化器要选用 bert 专用的 AdamW。建立模型并放在 gpu 上。训练模型时，将每个 step 设置为 40batch，做 4 个 epoch，在每个 batch 记录损失和，每个 step 记录准确率。

3.2 结果

每一个 epoch 的结果如下:

- Average training loss: 0.33
  Training epcoh took: 0:03:31
  Accuracy: 0.88
  Validation took: 0:00:06
- Average training loss: 0.19
  Training epcoh took: 0:03:30
  Accuracy: 0.88
  Validation took: 0:00:06
- Average training loss: 0.10
  Training epcoh took: 0:03:30
  Accuracy: 0.88
  Validation took: 0:00:07
- Average training loss: 0.05
  Training epcoh took: 0:03:33
  Accuracy: 0.88
  Validation took: 0:00:07

看出来，基本上一个 epoch 的训练其实模型都准确率已经到 88% 且基本到顶点，说明训练在准确率方面收敛很快训练效果很好，而且每个 epoch 的损失积累值也在明显减少。

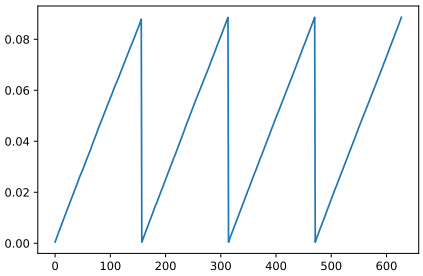

图 4: Acc

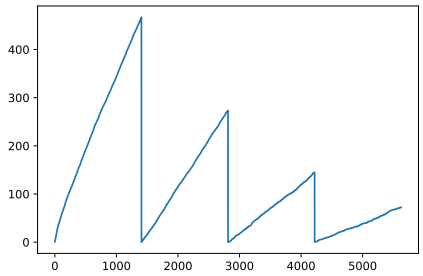

图 5: total_loss

将训练过程中的准确率和累计损失的值绘图见图4,5，训练过程确实如之前我的直观感受。对于预测测试结果如下：

- Accuracy: 0.8882
  Test took: 0:01:07

准确率接近 89%.

3.3 BERT 与 RNN 模型对比分析

上次 LAB2 的 RNN 模型的训练结果，256 隐藏单 RNN 与 256 隐藏 2RNN 当时的结果是：

Test Loss: 0.584 | Test Acc: 69.76%
Test Loss: 0.572 | Test Acc: 71.35%

对比此次实验的 BERT 模型，在第一个 epoch 结束之后的数据已经是：

Average training loss: 0.33 | Accuracy: 88%

训练损失值小于 RNN 但准确率远远高于 RNN（由于训练的方式有一些区别因此损失值的结论可能不严谨，但准确率是 RNN 达不到的）。

但是，鉴于 RNN 是一个仅仅与输入层，隐藏层，输出层的网络，而 BERT 实际上是一个语言模型。语言模型通常采用大规模、与特定 NLP 任务无关的文本语料进行训练，其目标是学习语言本身应该是什么样的的预训练模型。BERT 模型其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。

因此显然对于训练 imdb 这个特定的语言数据集和，BERT 拥有更好的训练结果。