2022-2023 学年秋季学期东北大学

《自然语言处理》实践报告

基于 Transformer 的
机器阅读理解任务报告

姓　　名 常宇莹

专　　业 计算机科学与技术

学　　号 2201766

学　　院 计算机科学与工程学院

2023 年 1 月 15 日

基于 Transformer 的机器阅读理解任务

一、报告背景

如今，我们在图像识别、机器翻译、语音合成等研究领域已经看到了机器学习等计算科学带来的显著成果。例如，在口罩出行成为全民标配的情况下，宿舍门禁系统的人脸识别技术已经可以快速、准确、无接触地识别戴口罩的学生；另一方面，目前阿尔法系列的棋艺几乎无人能敌；甚至，当下成熟的商业机器翻译系统也已经可以达到“信”、“达”的标准……这一波狂热过后，当我们重新审视机器学习这一概念时，一个最基本的问题可能还尚未解决：计算机究竟能够将我们人类的语言理解到何种程度？我们总是希望机器能够替我们做一些我们可以做但又过于简单枯燥的事情。比如，我们希望机器可以阅读特定领域的文献资料，然后利用学习到的知识去生产，但在让机器上场之前，我们需要确定它们是否真的学会了文献中的知识。自然语言处理(Natural Language Processing, NLP)的一个分支——机器阅读理解(Machine Reading Comprehension，MRC)正在努力回答上述问题。

众所周知，阅读理解是语言课程中的常规考试内容，研究者学习了人民教师的做法：如果学生在不作弊的情况下能够正确回答老师精心设计的题目，那么可以确信学生具有较好的理解能力。同样，采用类似的做法提出了一种用于考察机器智能水平的任务：机器阅读理解。机器阅读理解旨在让计算机拥有人类的语言逻辑思维，使其可以像人类一样阅读和理解文本，同时根据该理解回答相关问题。例如，当读者阅读文献时，面对较长文本，往往需要阅读很久才能找到需要的信息。而如果利用机器阅读理解技术为计算机赋予阅读、分析和归纳文本的能力，就可以快速在整篇文章中找到答案，从而减轻人们的信息获取成本[1]。

基于 BERT 预训练模型实现的机器阅读理解在网上随处可见，包括 Google 在 SQuAD 数据集上官方发布的机器阅读理解任务的实现版本，还有 Huggingface 通过 BERT 以及 BERT 系列的各个版本封装的 QA 模型，但很少见直接将 Transformer 模型应用于机器阅读理解领域。本次实践正是在课上学长讲解的

Transformer 模型代码的基础上，在 SQuAD 1.0 数据集上实现的机器阅读理解。本报告将在后续章节介绍实验使用的数据集及实验实现的全过程；另外，根据实验结果简单谈谈自己对 Transformer 模型的理解与感悟。

二、数据集及数据预处理

本次实验使用的数据集为 MRC 领域较为常用的 SQuAD 数据集，该数据集是 Stanford Question Answering Dataset 的首字母缩写。这是一个抽取式 QA 数据集，该数据集包含 10 万个(问题，原文，答案)三元组，原文来自于 536 篇维基百科文章。对于每个文章的问题，有很多标注人员标注答案，且答案出现在原文中。SQuAD 最大的特点在于，它是斯坦福在自然语言处理上，意图构造一个类似“ImageNet”的测试集合，并且将分数实时展示在 leaderboard 上。实验使用的是从 datasets 上 load_dataset("squad")的数据集，SQuAD 1.0 数据集结构形式如图 2-1 所示。

```
DatasetDict({
    train: Dataset({
        features: ['id', 'title', 'context', 'question', 'answers'],
        num_rows: 87599
    })
    validation: Dataset({
        features: ['id', 'title', 'context', 'question', 'answers'],
        num_rows: 10570
    })
})

{
    'id': '5733be284776f41900661182',
    'title': 'University_of_Notre_Dame',
    'context': 'Architecturally, the school has a Catholic character. Atop the Main Building\'s
    'question': 'To whom did the Virgin Mary allegedly appear in 1858 in Lourdes France?',
    'answers': {
        'text': ['Saint Bernadette Soubirous'],
        'answer_start': [515]
    }
}
```

图 2-1 SQuAD 1.0 数据形式

由上图可知，SQuAD1.0 由 json 格式的数据组成，其中，训练数据和验证数据都被封装为 Dataset 格式，训练集共计 87599 条，验证集 10570 条。在每个数据集的 Dataset 内部是一个字典，包含“id”、“title”、“context”、“question”和“answers”字段，而“answers”字段也包含一个字典，包括字段“text”和

“answer_start”。这里的“answer_start”是答案在“context”中 character 层面的索引，而不是每个 token 在“context”中的位置，因此，数据预处理时需要进行转换。为方便理解，总结数据结构形式如图 2-2 框图，其中，验证集与训练集格式相同，而该数据集的测试集并没有公开发布，为保证实验正常进行，我们用 sklearn.model_selection 包中的 train_test_split 将训练集以 7:3 的比例切分为实验使用的训练集和验证集，而将验证集作为测试数据来使用。

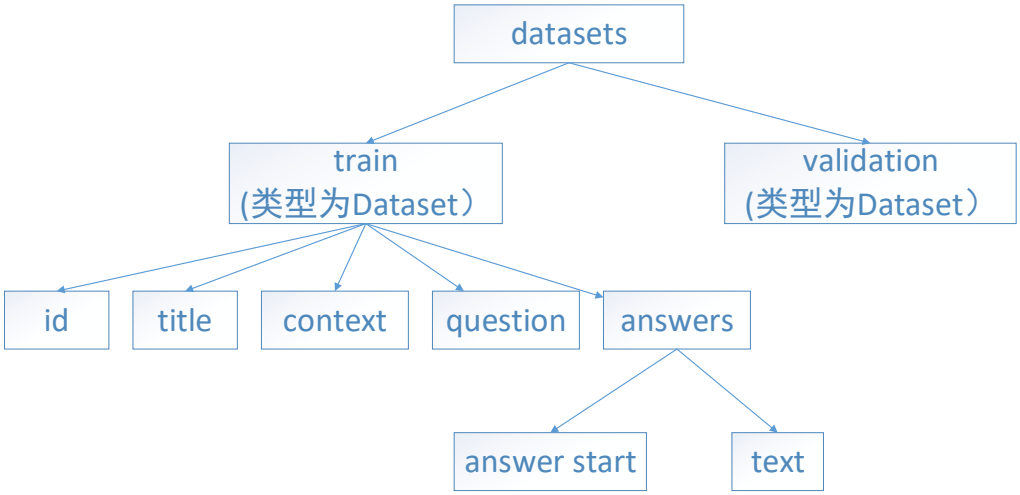

图 2-2 数据结构形式框图

SQuAD 数据集本身的结构略显复杂，这显著增加了实验处理整个数据集的难度，所以在预处理数据集时还需要费点功夫。本实验主要结合 BERT 给出预处理方式对 SQuAD 数据集进行重构。

其整体重构过程如下：首先，我们需要思考模型如何处理非常长的文本，最简便的方式就是直接将超长的输入截断，但这种方式很可能会损失大量信息，甚至会丢弃答案，因为答案都是从 context 中抽取出一个片段。所以我们在本实验中采用对于原始数据中的 context 按照指定的最大长度和重合部分大小进行切片处理，因为答案可能恰好在两个切片中间，所以我们用重合部分大小 doc_stride 来控制，让相邻切片之间有交集，然后再将 question 和 context 拼接在一起构造成为一个序列，同时需要添加对应的分隔符[CLS]和[SEP]；接着需要将上述序列转换得到 token 并做 padding，还要根据每个序列构造相应的 attention_mask 向量和 token_types_ids 向量。这里由于最初并不了解 Huggingface，因此没有使用

Huggingface 封装好的 Tokenizer 做处理，而是用上述最原始的处理方式，将 question+context 的序列转换为对应的 token 序列。

对输入模型的 question 和 context 做完处理后，还要对 answer 的开始和结束位置进行重构，因为 SQuAD 原始数据集给出的每个问题的答案并不是模型需要的 token 的开始和结束位置，而且我们还对长文本进行了切片，所以预处理时不得不重新编写函数来根据数据集中“answer_start”和“text”字段长度获得答案在原始上下文中 token 的 start_position 和 end_position。最初的主要思路就是依次遍历原始字符串中的每一个字符，然后判断其前一个字符是否为空格：如果前一个字符是空格，则表示当前字符是一个新单词的开始，将前面的多个字符 append 到列表中作为一个单词；如果不是空格，就表示当前的字符仍旧属于上一个单词的一部分，因此将当前字符继续追加到上一个单词的后面。另外还要额外记录当前字符所属单词的偏移量。这一套流程下来很是麻烦，而后面采用 tokenizer 就方便多了，tokenizer 中的 return_offsets_mapping 参数会得到切片和原始输入之间的对应关系 map。

在完成前面部分的内容后，还需要在构造每个 batch 前对输入序列进行 padding 并输入到模型中。

上述过程为本实验的数据预处理过程，使其适合于 SQuAD 1.0 数据集。程序源码对应 data_preprocess 函数，在使用 Tokenizer 前最原始的数据处理方式已用注释标记。

三、Transformer 实现过程

本实验的 Transformer 实现过程是在学长课上讲解的 transformer_wikitext2.py 程序的基础上完成的，另外还使用 Huggingface 中基于 BERT 封装好的 AutoModelForQuestionAnswering 模型开展对照实验。

在介绍本实验方法前，首先明确实验任务：在给模型输入一个 question 和一个 context 时，模型需要从 context 中预测出答案所在的位置(start_position 和 end_position)，并且，最终问题的答案一定是在 context 中，同时，答案也一定是一段连续的 span。有了上述限制，这类机器阅读理解任务也就变成了让模型预测

答案在给定文本中的起始位置以及结束位置。所以，问题最终变成在 Transformer 模型的基础上，再构建一个全连接层来对 Transformer 输出的每个 token 进行分类，判断它们是否属于 answer 的开始或结束位置。该实验本质上依旧可以归结为分类任务。

在实现 Transformer 的过程中对我的数学功底也提出了非常大的考验，尤其是对于维度控制。因此，我在程序中也对所有数据均加上了维度注释。其中在编写程序的过程中遇到一个问题，如图 3-1 所示。顺着报错开始寻找问题所在，一直以为是环境中有第三方库不兼容造成的问题，但检查并更新一遍后仍然报错，后来开始对照 transformer_wikitext2.py 中的数据维数逐个检查，最终发现是修改后传入模型中的数据直接替代了原始数据，导致第二次迭代的维数不匹配，从而出现了问题。

另外，再分享三个小 tip，都是我在实验中关于维度控制的心得：在拆维度时千万不要破坏维度原来本身的意义，后果很可怕；还有就是，虽然 reshape 函数可以用，但仍要记得 transpose 后如果接 permute 或者 view 必须要加 contiguous，这是数据真实存储连续与否的问题；一般都会把 batch_size 放在第 0 维，因为基本上不对 batch 维做操作，放在最前面可以用来防止影响后面总需要使用 transpose 而造成维度混乱。

```
    return forward_call(*input, **kwargs)
  File "D:\ProgramData\Anaconda3\envs\pytorch-1.7.1\lib\site-packages\torch\nn\modules\transformer.py", line 238, in forward
    output = mod(output, src_mask=mask, src_key_padding_mask=src_key_padding_mask)
  File "D:\ProgramData\Anaconda3\envs\pytorch-1.7.1\lib\site-packages\torch\nn\modules\module.py", line 1130, in _call_impl
    return forward_call(*input, **kwargs)
  File "D:\ProgramData\Anaconda3\envs\pytorch-1.7.1\lib\site-packages\torch\nn\modules\transformer.py", line 463, in forward
    x = self.norm1(x + self._sa_block(x, src_mask, src_key_padding_mask))
  File "D:\ProgramData\Anaconda3\envs\pytorch-1.7.1\lib\site-packages\torch\nn\modules\transformer.py", line 474, in _sa_block
    need_weights=False)[0]
  File "D:\ProgramData\Anaconda3\envs\pytorch-1.7.1\lib\site-packages\torch\nn\modules\module.py", line 1130, in _call_impl
    return forward_call(*input, **kwargs)
  File "D:\ProgramData\Anaconda3\envs\pytorch-1.7.1\lib\site-packages\torch\nn\modules\activation.py", line 1160, in forward
    attn_mask=attn_mask, average_attn_weights=average_attn_weights)
  File "D:\ProgramData\Anaconda3\envs\pytorch-1.7.1\lib\site-packages\torch\nn\functional.py", line 5089, in multi_head_attention_forward
    raise RuntimeError(f"The shape of the 2D attn_mask is {attn_mask.shape}, but should be {correct_2d_size}.")
RuntimeError: The shape of the 2D attn_mask is torch.Size([100, 100]), but should be (512, 512).
```

图 3-1 维度控制报错图

四、实验结果分析

使用相同的训练参数，通过 Huggingface 在 SQuAD 数据集上实现的机器阅读理解的结果与 transformer 实现版本进行对比，可以明显看到 transformer 逊色于 BERT 之处。其中，transformer 实现的 MRC，在训练初期，模型的准确率几

乎为 0，随着迭代次数的增加，模型性能会逐渐提升，但仍然明显低于 BERT 模型；而 BERT 版本的实现在刚开始训练时，准确率就很高。其中，两个版本的实现在训练初期的训练结果如图 4-1、4-2 所示。这足以看出 BERT 相较于 transformer 不仅在输入输出上的调整更加人性化，而且 BERT 中还包含很多语义信息，这使得模型最初训练就会有很不错的效果，这也是 transformer 无法比拟的。另外，对实验进行分析后，反观机器阅读理解任务，我还对任务本身有一些简单的思考和想法。

```
| epoch   1 |   100/ 6139 batches | lr 0.10 | ms/batch 144.92 | loss  5.16 | accuracy  0.01 |
| epoch   1 |   200/ 6139 batches | lr 0.10 | ms/batch 142.66 | loss  5.00 | accuracy  0.02 |
| epoch   1 |   300/ 6139 batches | lr 0.10 | ms/batch 144.35 | loss  4.99 | accuracy  0.02 |
| epoch   1 |   400/ 6139 batches | lr 0.10 | ms/batch 145.03 | loss  4.92 | accuracy  0.02 |
| epoch   1 |   500/ 6139 batches | lr 0.10 | ms/batch 146.30 | loss  4.92 | accuracy  0.01 |
| epoch   1 |   600/ 6139 batches | lr 0.10 | ms/batch 145.65 | loss  4.92 | accuracy  0.02 |
| epoch   1 |   700/ 6139 batches | lr 0.10 | ms/batch 144.93 | loss  4.84 | accuracy  0.02 |
| epoch   1 |   800/ 6139 batches | lr 0.10 | ms/batch 145.14 | loss  4.83 | accuracy  0.02 |
| epoch   1 |   900/ 6139 batches | lr 0.10 | ms/batch 146.33 | loss  4.83 | accuracy  0.02 |
| epoch   1 |  1000/ 6139 batches | lr 0.10 | ms/batch 145.08 | loss  4.78 | accuracy  0.02 |
| epoch   1 |  1100/ 6139 batches | lr 0.10 | ms/batch 146.22 | loss  4.82 | accuracy  0.02 |
| epoch   1 |  1200/ 6139 batches | lr 0.10 | ms/batch 147.78 | loss  4.82 | accuracy  0.02 |
| epoch   1 |  1300/ 6139 batches | lr 0.10 | ms/batch 146.25 | loss  4.81 | accuracy  0.02 |
| epoch   1 |  1400/ 6139 batches | lr 0.10 | ms/batch 146.87 | loss  4.80 | accuracy  0.02 |
| epoch   1 |  1500/ 6139 batches | lr 0.10 | ms/batch 148.67 | loss  4.84 | accuracy  0.02 |
| epoch   1 |  1600/ 6139 batches | lr 0.10 | ms/batch 147.70 | loss  4.76 | accuracy  0.02 |
| epoch   1 |  1700/ 6139 batches | lr 0.10 | ms/batch 147.78 | loss  4.72 | accuracy  0.03 |
| epoch   1 |  1800/ 6139 batches | lr 0.10 | ms/batch 146.24 | loss  4.74 | accuracy  0.03 |
```

图 4-1 transformer 实现版本在训练初期 accuracy 值

```
| epoch   0 |   100/ 6139 batches | loss  3.59 | accuracy  0.19 |
| epoch   0 |   200/ 6139 batches | loss  2.10 | accuracy  0.46 |
| epoch   0 |   300/ 6139 batches | loss  1.93 | accuracy  0.49 |
| epoch   0 |   400/ 6139 batches | loss  1.76 | accuracy  0.54 |
| epoch   0 |   500/ 6139 batches | loss  1.71 | accuracy  0.54 |
| epoch   0 |   600/ 6139 batches | loss  1.51 | accuracy  0.59 |
| epoch   0 |   700/ 6139 batches | loss  1.54 | accuracy  0.57 |
| epoch   0 |   800/ 6139 batches | loss  1.45 | accuracy  0.61 |
| epoch   0 |   900/ 6139 batches | loss  1.47 | accuracy  0.59 |
| epoch   0 |  1000/ 6139 batches | loss  1.40 | accuracy  0.63 |
| epoch   0 |  1100/ 6139 batches | loss  1.43 | accuracy  0.60 |
| epoch   0 |  1200/ 6139 batches | loss  1.48 | accuracy  0.60 |
| epoch   0 |  1300/ 6139 batches | loss  1.40 | accuracy  0.63 |
| epoch   0 |  1400/ 6139 batches | loss  1.37 | accuracy  0.62 |
| epoch   0 |  1500/ 6139 batches | loss  1.30 | accuracy  0.63 |
| epoch   0 |  1600/ 6139 batches | loss  1.36 | accuracy  0.61 |
| epoch   0 |  1700/ 6139 batches | loss  1.27 | accuracy  0.65 |
| epoch   0 |  1800/ 6139 batches | loss  1.37 | accuracy  0.61 |
| epoch   0 |  1900/ 6139 batches | loss  1.32 | accuracy  0.64 |
| epoch   0 |  2000/ 6139 batches | loss  1.28 | accuracy  0.65 |
| epoch   0 |  2100/ 6139 batches | loss  1.38 | accuracy  0.63 |
```

图 4-2 BERT 实现版本在训练初期 accuracy 值

通过这几周在完成实践任务的过程中对文献和程序的理解和思考，个人认为机器阅读理解任务的要点在于让模型真正地去理解人类语言，使其“知其然”的同时也要“知其所以然”，在理解能力上，如果类比人类，模型在自然语言理解中最大的局限在于缺乏“常识”，而真实环境下，人类的理解却又离不开常识知识。例如，当人说这杯水真“烫”时，我们能够理解烫是什么概念，因为我们可能被烫伤过，但是机器却无法理解，因为它没有温度传感器，无法感受到这个世界，它能感受到的只是用来训练它的文本。再比如，一个刚出生的婴儿如果失去所有感官，只剩听觉和语言能力，并且每天只能躺在床上不能与社会接触，光凭一个人每天和他对话，他能否像人类一样感受世界？能否像人类一样正常聊天？显然不能。因此，缺乏常识知识的机器阅读理解离人类的理解水平仍有较大差距。

五、总结

在前八周的课程学习和后面完成报告的过程中，这是我第一次主动地去接触英文文献和这么高强度的代码，刚开始读起来确实有些吃力，要花很长时间才能读懂论文，同时很多内容看不懂、推理过程不完善等等。写代码更是困难重重，一个 bug 改一晚上都是常有的事儿，甚至还会遇到一些疑难杂症，挂梯子去外网寻找各种解决办法，起初用最原始的方法进行数据处理，后来发现新的而且非常好用的 Tokenizer。其中，我逐渐学会了根据参考文献去选读和要精读论文相关的内容，逐步熟悉根据复杂的公式去理解代码的每一行的过程。通过这段时间的学习，我对自己的研究领域有了更深入地了解，对自然语言处理也有了更进一步的认识，甚至还能有一些自己的想法。很感谢老师和学长们能够给我们提供这样一个机会自主学习，在该门课程中收获颇丰！

参考文献

[1] 徐士亮. 新闻领域非结构化文本中文机器阅读理解研究[D]. 黑龙江: 哈尔滨工业大学, 2020.

[2] Ashish V, Noam S, Niki P, et al. Attention Is All You Need.[J], ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS 30 (NIPS 2017), 2017, 30(): 5998-6008.