声明：我已知悉学校对于考试纪律的严肃规定，将秉持诚实守信宗旨，严守考试纪律，不作弊，不剽窃；若有违反学校考试纪律的行为，自愿接受学校严肃处理。

2018-2019 学年第二学期 COMP130137.01 《模式识别与机器学习》课程项目 机器阅读理解 (项目名称)

学号：16307130260 , 姓名：熊浩然，贡献: 33%，签名：
学号：16307130138 , 姓名：常朝坤，贡献: 33%，签名：
学号：16307130086 , 姓名：何畅扬，贡献: 33%，签名：

Abstract

本课程项目由三人分工协作完成了两个任务，分别为百度 MRC 比赛和 fastnlp MRC 代码实现。百度 MRC 比赛过程中，我们选择了 BiDAF 和 bert 两种模型同步进行探索的方式。BiDAF 模型在经过添加 dropout、self attention 以及数据清洗选择、模型输出后处理等工作之后获得了比赛使用的主要模型。bert 模型因为模型本身特性以及数据过长等因素没有获得整个过程的模型，但是总结一些关于 bert 训练的经验，并训练获得了 yes_or_no 部分的分类模型。

fastnlp MRC 部分主要实现 BiDAF 在 SQuAD1.1 数据集上的训练和测试。主要工作有 SQuAD 数据集的 vocabulary 和 dataset 的建立、BiDAF 在 pytorch 和 fastnlp 框架下模型的实现以及 SQuAD metric 的重新实现。

1 问题描述

机器阅读理解 (Machine Reading Comprehension) 是指让机器阅读文本，然后回答和阅读内容相关的问题。阅读理解是自然语言处理和人工智能领域的重要前沿课题，对于提升机器智能水平、使机器具有持续知识获取能力具有重要价值，近年来受到学术界和工业界的广泛关注。本次机器阅读理解竞赛是中国计算机学会、中国中文信息学会和百度公司联合举办的 2019 语言与智能技术竞赛的一部分，提供了面向真实应用场景的大规模高质量中文阅读理解数据集，重点关注当前优秀的阅读理解系统尚不能正确回答的问题，全面评测机器进行深度语言理解以回答复杂问题的能力。

本次竞赛的任务为对于给定问题 *q* 及其对应的文本形式的候选文档集合 *D=d1, d2, ..., dn, * 要求参评阅读理解系统自动对问题及候选文档进行分析，输出能够满足问题的文本答案 a，使得 a 能够正确、完整、简洁地回答问题 *q*。任务给出了包含约 28 万来自百度搜索的真实问题的数据集，划分为包含 27 万个问题的训练集、约 3000 个问题的开发集和约 7000 个问题的测试集，每个问题对应 5 个候选文档文本及人工整理的优质答案。在评判方法上，竞赛采用 ROUGE-L 和 BLUE4 作为评价指标，前者为主评价指标。针对是非及实体类型问题，对 ROUGE-L 和 BLEU4 评价指标进行了微调，适当增加了正确识别是非答案类型及匹配实体的得分奖励，一定程度上弥补传统 ROUGE-L 和 BLEU4 指标对是非和实体类型问题评价不敏感的问题。共有一百多支队伍参加了本次机器阅读理解竞赛。

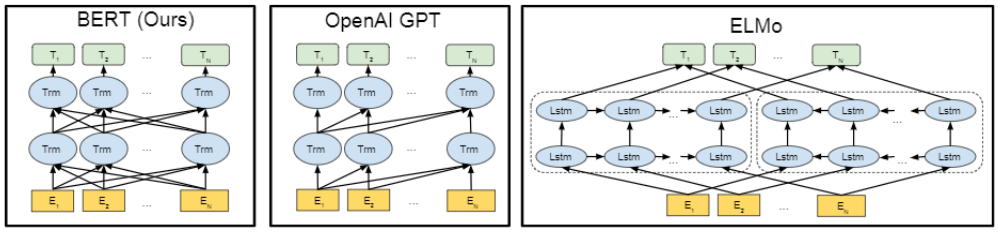

Figure 1: BERT 与 openAI GPT 以及 ELMo 的结构对比图

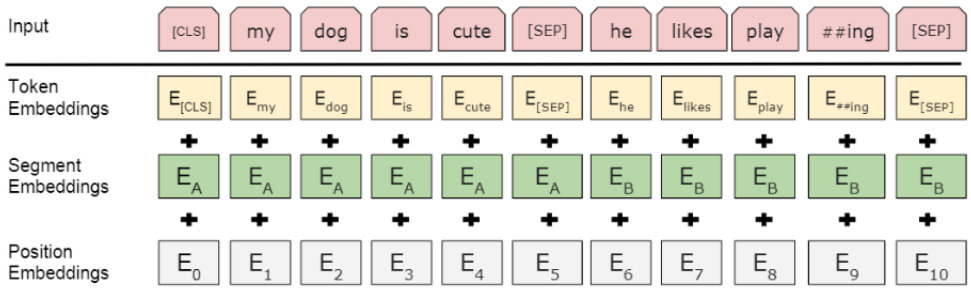

Figure 2: Transformer 网络结构图

2 方法与技术

本部分主要介绍实验中主要使用的两种模型：BERT 与 BIDAF。预处理、输出处理等将在实验设计部分具体介绍。

2.1 BERT

bert 是自 2018 年 10 月以来最火的模型，在包括机器阅读理解等多个 NLP 领域都有很好的表现，不仅具有 state-of-the-art 的高准确率，而且相对 rnn 能捕捉更长距离的依赖、更加高效。因此，我们也在实验中尝试了 BERT 模型。

bert 与 openAI GPT 和 ELMo 的对比如 figure1 所示。简单来说，bert 相当于双向 transformer block 连接的 OpenAI GPT。

其中，transformer 的结构图如 Figure2 所示。我们先从 transformer 看起。Transformer 是一个 encoder-decoder 的结构。上图中，左侧部分为 encoder，由 Multi-Head Attention 和一个全连接组成，用于将输入语料转化成特征向量；右侧部分是 decoder，其输入为编码器的输出以及已经预测的结果，由 Masked Multi-Head Attention, Multi-Head Attention 以及一个全连接组成，用于输出最后结果的条件概率。bert 的输入很有讲究 (见 figure3)，需要三个部分组成，如上图所示，bert 需要对 input 进行三个部分的处理。第一个是最基础的，将 word 转换为 token embedding，即将 word 转换为 index 并在其中填入 [CLS] 和 [SEP] 两种参数，将第一段和第二段文字进行分离。为了获得区分，bert 还为 tokenize 之后的句子进行了 Segment Embedding，即用不同的 mask 来标注第一句话的 token 和第二句话的 token。第三部分为位置信息，相当于一个 valid mask，对于有用的位置 mask 为 1，其余 mask 为 0，能够体现出 tokenize 之后的句子的位置和长度。

在数据经过 bert 之后，需要对其输出进行后处理，针对不同问题实现不同的上层结构，用于得到需要的结果。本次比赛中主要使用了 QA task 结构和 Sentence Pair Classification Task 结构。如 figure4，Sentence Pair Classification Task 需要将第一个句子和第二个句子连接之后输入到 bert 中，上层利用 CLS 中的结果连接到一个 FC 层，最后通过 softmax 来得到最终结果，loss 使用交叉熵来进行训练。

对于 QA 问题，利用相似的处理，将问题作为第一句话，将文章作为第二句话输入模型。最终将输出连接到 FC 层，获得每个位置对应的 logits，再将其转换为每个位置作为开始以及结束的概率。loss 为目标开始位置和目标结束位置交叉熵的和，进行训练。

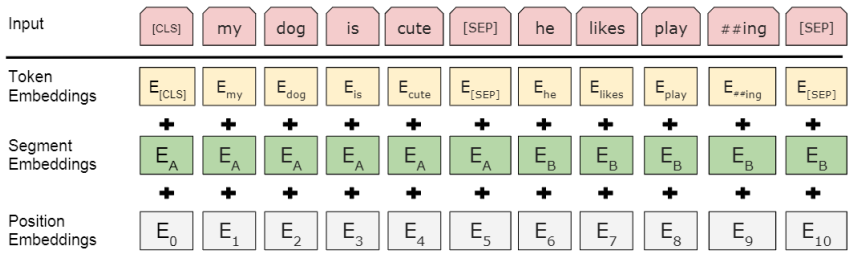

Figure 2: BERT input representation. The input embeddings is the sum of the token embeddings, the segmentation embeddings and the position embeddings.

Figure 3: BERT 网络输入的组成

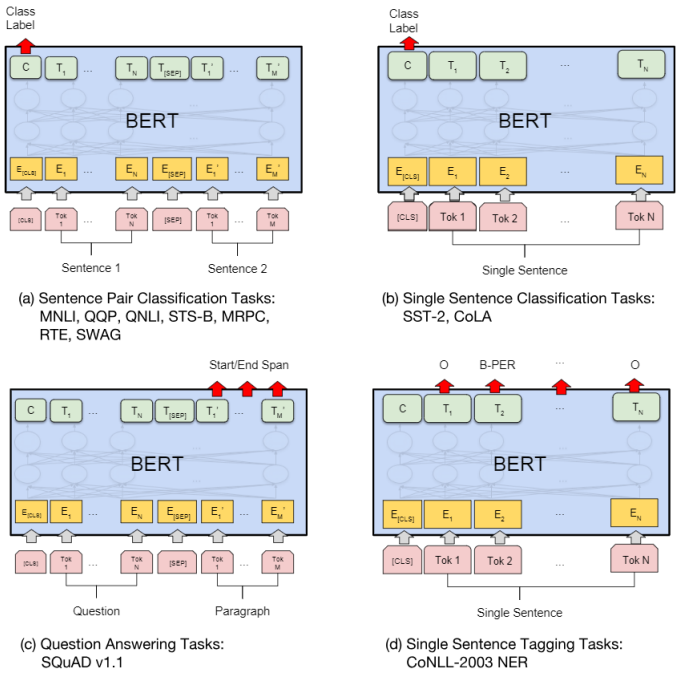

Figure 4: BERT 模型所应用的四种 NLP 任务

因为 transformer 结构的影响，bert 对输入的数据有长度限制，要求长度不能超过 512（因为 bert 代码只能支持多 tpu 训练而不能支持多 gpu 训练，时间不够充裕无法修改成支持多 gpu 的代码，而且调研时发现 batch size 过小会导致 bert 无法 fine tune，所以选择了 base 版本），所以在输入进入时需要进行一些处理。

对于 Sentence Pair Classification，bert 提供的方式为直接截断。因为大部分输入都比较短。

对于 QA，bert 提供了更加复杂的机制，对于一个问题和一篇文章，将问题根据设置的阈值进行截断，截断后如果剩余空位置能够放入文章全部 token，则直接将其拼接后输入 bert。但是如果文章过长，则考虑一个剩余长度大小的 window，设置合适的 stride，在文章上进行滑动，从而获得多段截断的文章内容，将问题和该片段进行拼接，得到一个 feature。一个 example 可以得到一个或者多个 feature。

将 features 输入到 bert 后获得对应的输出，将输出进行选择，得到文章每个位置合适的表示，连接 FC 层后获得该位置对应的 logits，再将其转换为每个位置作为开始以及结束的概率。

System	Seq Length	Max Batch Size
`BERT-Base`	64	64
...	128	32
...	256	16
...	320	14
...	384	12
...	512	6
`BERT-Large`	64	12
...	128	6
...	256	2
...	320	1
...	384	0
...	512	0

Figure 5: bert 输入的细节

2.2 bert 模型探究始末

因为刚开始对 bert 的了解仅限于对模型内部算法和构造以及输入输出格式的了解，对 bert 的使用以及 bert 的特性没有太多的概念，再加上比赛时间比较紧，所以很遗憾最终没有探索获得针对改数据集效果较好的、用于整个阅读理解 QA 过程的 bert 模型，但在比赛接近提交截止日期时训练出一份效果较佳的预测 yes_or_no 分类的 bert 模型。

下面给出探究的主要工作

1. google 提供了一份结构非常清晰的 QA task 代码，为了尽可能小的降低产生 bug 的可能性，决定以原代码为主要逻辑，获得了一份预处理和训练分开的 bert for QA 代码，而且写了一份脚本，将百度原始数据转换为 bert 代码能够接受的输入格式。
2. 为了直观感受 bert 在本问题以及此数据集上的表现能力，于是将数据直接输入到 QA stride 模型中进行处理，并对 bert 进行了 fine tune，两个 epoch 历时将近 10 个小时。但是获得的模型预测结果非常差，远不如预想的水平。
3. 将预测结果进行 print，发现预测结果中很多莫名其妙的符号，因此猜测效果不好是因为训练数据集中有各式各样的无用符号，影响了模型的训练。根据上述猜测，我们对数据集进行了清洗，去除了许多不需要的符号，并去掉很多无用的 whitespace。再次将清洗过的语料输入到模型中进行训练，得到结果无明显提升。
4. 开始考虑是否是因为参数设置的问题，同步利用多张卡训练了多个模型，使用不同的 lr、问题长度阈值、stride 以及 window size 阈值，但是训练出的结果仍然不理想。
5. 探究 tokenization 部分的逻辑，发现文章长度过长，一个 example 可能会得到数十个 feature，于是考虑在训练的时候选择相关程度最高的文字作为文章段进行输入，输入长度得到了大幅缩减。训练之后结果有所提升，但是依然没能达到百度给的 base line 的水平，这个时候时间已经接近比赛截止日期。
6. 为了不浪费 bert 这条路，我们将问题和获得的最终答案拼接，进行三分类，训练了一个 yes_or_no 分类模型，因为输入数据非常理想、较为规整，所以获得预测的准确率在 dev 集中能够达到 90%，最后这一部分代码被融入到最终的结果。
7. 比赛过后，反观为何不能训练出合适的 bert for QA 代码，我们的猜测是虽然已经选择了相关程度最高的文字作为文章段进行输入，但是这部分段落依然长度较大，滑动窗口平均需要滑动 4-5 次。而 bert 在 squad 数据集中滑动窗口使用次数并不是很多，说明 bert 对于较长文字的前后文关联的记忆能力依然不是很强。但是该数据集

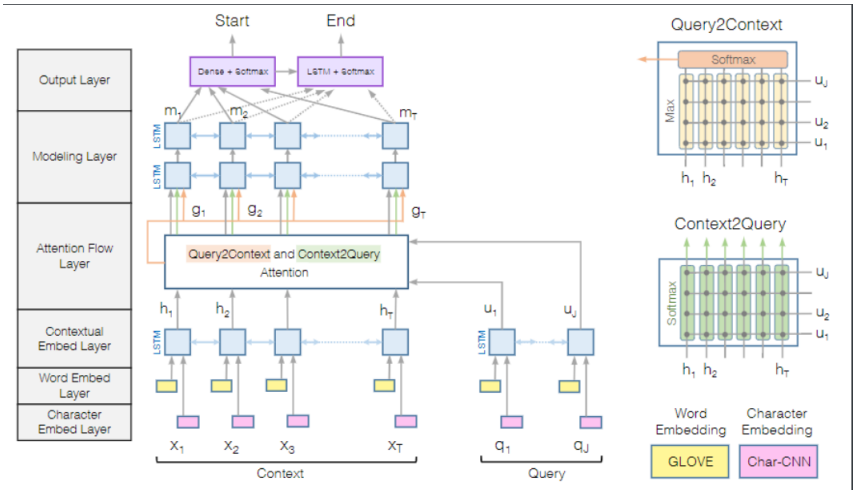

Figure 6: BiDAF 网络结构图

的文章以及最终答案都非常的的长（相对于 512 宽度而言），所以如果没有足够的数据筛选工作，很难训练出好的 bert 模型。

2.3 BIDAF

BIDAF 的结构如 figure6 所示。自底向上逐层分析：

1. Character Embedding Layer：利用 character 级别的 CNN 转换 word 到向量空间
2. Word Embedding Layer: 利用预训练的 word embedding 模型转换 word 到向量空间
3. Contextual Embedding Layer: 对上一部 context 和 query 的结果分别用 Bi-LSTM 编码，获取局部关系，得到隐层输出 H 和 U
4. Attention Flow Layer: 耦合 query 和 context 向量，并为 context 中的每个单词生成一组 query-aware 的特征向量。具体的说，利用 attention 机制，分别计算对于每一个 context word，哪些 query words 相关性最高（C2Q）；以及对于每一个 query word，哪些 context words 相关性最高 (Q2C)；再将结果拼接。
5. Modelling Layer: 再经过一个 Bi-LSTM 模型得到 M
6. Output Layer: 预测 start 与 end

3 实验设计

3.1 预处理

在预处理之前，我们先简单分析 Dureader 给出的数据。在 Dureader 数据中，每个问题有至多 5 个相关文档，平均文档长度为 394，这也意味着若对一个 question，将所有 document 作为输入不是一个好的选择（长度过长）。为了实验方便，我们没有自己依据 raw data 做分词等操作，而是直接根据 preprocessed 数据进行模型训练。

preprocessed 数据的一个样例如下：其中一些重要的参数属性在这里给出解释:

1. is_selected: 在选择答案时是否参考了此文档。若为 False, 则此文档未被参考
2. most_related_para: 可能包含每个文档答案的最相关的段落，根据 answer token 对每个文档的最高 recall 选出
3. fake_answers: 使用正确答案匹配文档，找到 F1-score 最大的子串，作为 fake_answers
4. answer_docs: fake_answers 所在 document 的 id
5. answer_spans: fake_answers 于所在 document 中的 span

在训练时，找到 answer_docs 对应的 document，再依据 most_related_para 及 answer_spans 找到输入数据。

显然，这样的数据输入存在一个很大的问题：很多有用的信息在数据输入阶段就被隐藏了（我们只使用了 answer_docs 中 most_related_para 的部分信息）。因此，在 preprocessed 的基础上，我们对数据输入做出了如下修改：

1. 若标题与各段落长度之和不超过设定的最大长度值（最终设定为 500），则将它们拼接并作为输入；
2. 否则，将标题及 most_related_para 先放入输入中；
3. 将去掉 most_related_para 的所有段落按照与问题的 bleu-4 分值排序（若段落长度本身小于 4，不满足 bleu-4 要求，则将其舍弃）
4. 按 bleu-4 分值从高到底不断将段落加入输入中，直到输入长度达到设定的最大长度值为止

3.2 模型训练

3.2.1 SIF$_E mbedding$

阅读理解问题要求机器学习到语句的语义信息，而简单的随机 embedding 是无法做到这一点的，所以实验中加入了 SIF Embeddding 机制，其具体实现依赖于 gensmi 工具的 word2vec 类。由于 baseline 提供了加载预训练 embedding 的接口（但是没有提供训练模型与方法），所以 embedding 的训练在本实验中是单独进行的，具体的训练逻辑可参见 SIF.py 下的 SIFModel。

3.2.2 Self-attention

self-attention（自注意力机制）在 NLP 任务中具有捕获句子内部结构的功能，且相对 RNN 结构有更快的速度和距离更远的依赖捕获能力。本实验中，在 Bidaf 模型的 AttentionFlow Layer 添加一个 context2context 的 attention 层，并整合原来的结果形成新的输出送往 Modeling Layer。（question 的语句短而简单，为其增加 self-attention 层意义不大）

3.2.3 Dropout

Dropout 通过随机失活部分梯度而提升模型的泛化能力，在神经网络训练中是一种常用的方法，从另一个角度看，dropout 也像是一种模型级别的 ensemble。在不使用 dropout 进行训练时，发现模型很快便发生了过拟合现象，所以本实验中加入了 rate=0.3 的 dropout，以提升模型的泛化能力。结果证明，dropout 对模型的改进约有 2 个百分点。

preprocessed 数据的一个样例如下：其中一些重要的参数属性在这里给出解释:

1. is_selected: 在选择答案时是否参考了此文档。若为 False, 则此文档未被参考
2. most_related_para: 可能包含每个文档答案的最相关的段落，根据 answer token 对每个文档的最高 recall 选出
3. fake_answers: 使用正确答案匹配文档，找到 F1-score 最大的子串，作为 fake_answers
4. answer_docs: fake_answers 所在 document 的 id
5. answer_spans: fake_answers 于所在 document 中的 span

在训练时，找到 answer_docs 对应的 document，再依据 most_related_para 及 answer_spans 找到输入数据。

显然，这样的数据输入存在一个很大的问题：很多有用的信息在数据输入阶段就被隐藏了（我们只使用了 answer_docs 中 most_related_para 的部分信息）。因此，在 preprocessed 的基础上，我们对数据输入做出了如下修改：

1. 若标题与各段落长度之和不超过设定的最大长度值（最终设定为 500），则将它们拼接并作为输入；
2. 否则，将标题及 most_related_para 先放入输入中；
3. 将去掉 most_related_para 的所有段落按照与问题的 bleu-4 分值排序（若段落长度本身小于 4，不满足 bleu-4 要求，则将其舍弃）

31	baseline(ensemble) FDNLP 复旦大学	44.82	42.26	2019/5/19

Figure 7: 比赛提交结果

Table 1: progress

Method	ROUGE-L	BLUE-4 (μm)
baseline	37.97	31.04
preprocess	42.85 (+4.88)	39.14 (+8.10)
model optimize	47.50 (+4.65)	41.60 (+2.46)
postprocess	48.04 (+0.54)	41.76 (+0.16)
bert yes-or-no	49.84 (+1.80)	45.64 (+3.88)

4. 按 bleu-4 分值从高到底不断将段落加入输入中，直到输入长度达到设定的最大长度值为止

3.3 输出处理

后处理涉及一些简单的操作，但能有效地进一步提高模型准确率。我们通过观察，发现很多预测结果末尾都没有标点，但给出的标准输出中均以句号结尾。故我们将尾字符不是句号的结果均加上句号，将结果提高约 0.5%。

4 结果与分析

在竞赛官网上，提交的最终结果如 figure7 所示。其中，ROUGE-L 为 44.82，BLEU-4 为 42.26。

由于官网每日一次的提交限制，我们大多数测试在 https://ai.baidu.com/broad/submission 上完成。由于数据集不同，在此网站上测试的最优结果为 ROUGE-L：49.84，BLEU-4：45.64。各个操作在数据集上的效果为：可以看到，预处理与模型调优是最重要的两个部分，但后处理等操作也能为模型增色。

5 相关工作对比分析

目前，2019 年语言与智能高峰论坛尚未召开，因此我们在实验总结与相关工作上以 2018 年竞赛的表现优秀者作为参照。

5.1 Naturali(ROUGE-L: 63.38 BLEU-4: 59.23)

Naturali 工作如 figure8 所示。其中三个方面令我印象深刻：第一，它在开发集中仅使用前三篇文档进行预测的结果，这比使用全部文档进行预测得到的结果有明显提高；第二，它 ensemble 了 BiDAF，MatchLSTM 与 DCA，这在 single model 的基础上上升了 2.28%；第三，它在答案片段抽取时利用了多个参考答案，这也是我们模型训练中未考虑到的。

5.2 台達電子 (ROUGE-L:56.67 BLEU-4: 48.03)

台達電子的工作如 figure9 所示。其在改良预测代码中采用的方法是全文串接，将文章段落以句号串接起来，以整篇来预测答案，大大提高了性能。另外，它工作的另一个亮点是数据扩展：它利用所有资源，扩增多个参考答案做为训练数据。另外，为了保持训练数据的质量，工作中它也滤除了相似度分数低于 0.7 的训练数据

模型各部分的影响

模型	Search		Zhidao	
	ROUGE-L	Δ	ROUGE-L	Δ
基线系统	30.77	-	45.90	-
+ 篇章预处理	42.07	+ 11.30	50.52	+ 4.62
+ 预训练词向量	46.98	+ 4.51	55.02	+ 4.50
+ 其它特征	47.15	+ 0.17	55.67	+ 0.65
+ 多个答案	48.75	+ 1.60	56.97	+ 1.30
+ 联合训练	48.76	+ 0.01	57.20	+ 0.23
+ 最小风险训练	49.61	+ 0.85	58.11	+ 0.91

Figure 8: Naturali 组比赛报告结果

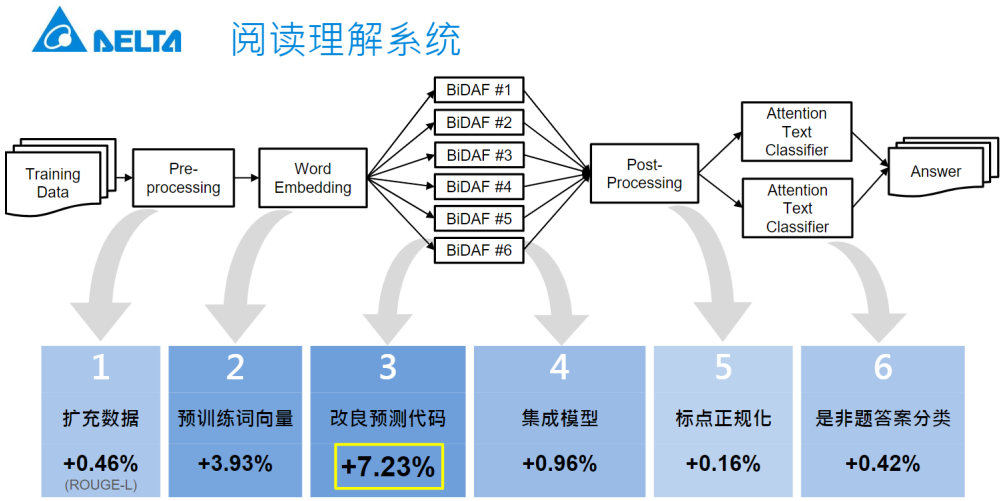

Figure 9: 台達電子组比赛报告结果

6 为 FASTNLP 添加 MRC 模块

6.1 主要任务

在 fastnlp 框架下实现 bidaf 模型，并编写运行样例，体现 fastnlp 提供的工具的便捷性，最后将模型最终结果与原论文结果（squad 1.1）进行对比。

6.2 开发环境

为了使用最新的 fastnlp，本部分的代码在 fastnlp dev0.5 的代码环境下进行开发。dev 代码中有一些小错误没有解决，在使用时，将无用的错误部分注释，以达到能够正常开发的状态。

6.3 主要实现

1. squadDataset
   该模块的任务为加载数据并构建词表。加载数据的逻辑参考 SougouMRCToolkit 中的 SquadReader，构建词表的过程使用 fastnlp 提供的 Vocabulary 类进行快速构建。
2. bidaf BiDAF 模型按照论文中的 bidaf 结构进行实现，在 forward 部分包括了 Character Embedding Layer，Word Embedding Layer，Highway network，Contextual Embedding Layer，Attention Flow Layer，Modeling Layer 和 Output Layer。

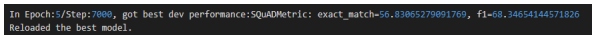

Figure 10: 模型运行最终结果

fastnlp 提供了 predict 功能，能够非常方便地实现 start 和 end 的 masked 概率，具体代码见 predict 函数。

模型当中也使用了 fastnlp dev 版本中的 pretrained embedding。

3. loss

   loss 的实现较为简单，将 forward 部分获得的 start logits 和 end logits，利用 target 计算得到交叉熵，求和后作为 loss。

4. metric

   fastnlp 中已经提供了一个关于 squad 的 metric，但是代码中对于无问题回答的 index 设置和我们数据处理方式有所出入，而且代码中获得最优结果的方式过于简单（直接选择概率最大的 start 和 end），代码中甚至出现少许错误。所以我们重写了 metric，提供了更好的 index 选择方式，纠正了错误。具体代码见 metric_best_pred.py(index 获取逻辑见 _get_best_answer 方法)。

   但是由于 f1 计算方式与原论文代码的计算方式不太一样，会导致结果降低较多，所以我们决定再重写一份 metric。

   重写的 metric 利用了 SougouMRCToolkit 的 SquadEvaluator 类，在其基础上构建。主要逻辑为利用模型的预测结果（answer span）生成文本型的 answer，并与相对应的 ground truth 计算 f1 score.

5. trainer

   利用 trainer 将之前的实现组织起来，训练获得最优的结果。

详细逻辑可参见 machine_reading_comprehension 文件夹中对应源码。

6.4 最终结果

最终结果如 figure10 所示。

6.5 TODO

因为从领会老师布置任务的主要意图，到 pj 提交只有四天时间，所以本部分代码仅做到了在 fastnlp 框架下实现可运行且效果较为接近原论文的 bidaf，为了让这份代码能够 merge 进入 fastnlp 我们还需要在后续完成以下任务。

1. 模型效果问题: 因为原模型是老版本 tensorflow 的代码，所以在复现为 pytorch 代码的过程中可能会有些许行为不太一致，造成模型效果稍微不如原模型，需要对细节进行修改，从而让结果尽可能接近原论文水平。
2. 提高 fastnlp 使用率:fastnlp 中还提供了很多能够代替本部分代码的功能，但是因为时间不足没有完全“fastnlp 化”，后续需要利用更多的 fastnlp 提供的工具来实现这个模型。
3. 评测标准问题:SQuAD 官网提供了一份评测的代码，SougouToolkit 的评测逻辑实际上也是在该份代码上的包装，由于本实验中的 metric 又是对 SougouToolkit 的评测代码的包装，所以逻辑有些冗余，可以进一步精简。

7 dev 版本 bug

7.1 No module named 'fastNLP.io.data_loader'

7.1.1 错误描述

如 figure10 所示。

7.1.2 错误修正

在 data_loader 文件夹下创建 __init__.py 即可

```
Traceback (most recent call last):
  File "train_bidaf.py", line 6, in <module>
    from model.model import BiDAF
  File "/Users/jarvis/Desktop/fastNLP/reproduction/machine_reading_comprehension/model/model.py", line 5, in <module>
    from fastNLP.modules.aggregator.attention import BiAttention
  File "/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/fastNLP/__init__.py", line 61, in <module>
    from .core import *
  File "/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/fastNLP/core/__init__.py", line 18, in <module>
    from .callback import Callback, GradientClipCallback, EarlyStopCallback, TensorboardCallback, LRScheduler, ControlC
  File "/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/fastNLP/core/callback.py", line 77, in <module>
    from ..io.model_io import ModelSaver, ModelLoader
  File "/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/fastNLP/io/__init__.py", line 29, in <module>
    from .dataset_loader import DataSetLoader, CSVLoader, JsonLoader, ConllLoader, \
  File "/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/fastNLP/io/dataset_loader.py", line 33, in <module>
    from .data_loader.sst import SSTLoader
ModuleNotFoundError: No module named 'fastNLP.io.data_loader'
```

Figure 11: 错误 1

fastNLP/core/metrics.py

Hunk 1 : Lines 848-854

```
848 848         """get_metric函数将根据evaluate函数累计的评价指标统计量来计算最终的评价结果."""
849 849         evaluate_result = {}
850 850
851     -       if self.no_ans_correct + self.no_ans_wrong + self.has_ans_correct + self.no_ans_wrong <= 0:
    851 +       if self.no_ans_correct + self.no_ans_wrong + self.has_ans_correct + self.has_ans_wrong <= 0:
852 852             return evaluate_result
853 853
854 854         evaluate_result['EM'] = 0
```

Figure 12: 修正 2

7.2 metric error

7.2.1 错误描述

fastnlp dev 版中提供了一个用于阅读理解的 metric 模块，但是其中 get_metric 方法中有一个手抖型错误尚未更正。

7.2.2 错误修正

如 figure11 所示。

8 参考文献

1. Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. 2016. Bidirectional attention flow for machine comprehension. arXiv preprint arXiv:1611.01603 .
2. Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.
3. W. He, K. Liu, Y. Lyu, S. Zhao, X. Xiao, Y. Liu, Y. Wang,H. Wu, Q. She, X. Liu et al., "Dureader: a chinese machine reading comprehension dataset from real-world applications," arXiv preprint arXiv:1711.05073, 2017.
4. Vaswani, Ashish, et al. Attention is all you need. Advances in Neural Information Processing Systems. 2017.
5. Srivastava N, Hinton G, Krizhevsky A, et al. Dropout: A simple way to prevent neural networks from overfitting[J]. The Journal of Machine Learning Research, 2014, 15(1): 1929-1958.
6. FastNLP 中文文档
   https://fastnlp.readthedocs.io/zh/latest/index.html
7. Sougou SMRCToolkit
   https://github.com/sogou/SMRCToolkit