文本复制检测报告单(全文标明引文)

№:ADBD2019R_2019052716103144659745215 1 检测时间:2019-05-27 16:10:31

检测文献: 7145807_刘嘉挺_法律问题关键词抽取系统
作者: 刘嘉挺
检测范围: 中国学术期刊网络出版总库
中国博士学位论文全文数据库/中国优秀硕士学位论文全文数据库
中国重要会议论文全文数据库
中国重要报纸全文数据库
中国专利全文数据库
图书资源
优先出版文献库
大学生论文联合比对库
互联网资源(包含贴吧等论坛资源)
英文数据库(涵盖期刊、博硕、会议的英文数据以及德国Springer、英国Taylor&Francis 期刊数据库等)
港澳台学术文献库
互联网文档资源
CNKI大成编客-原创作品库
个人比对库
时间范围: 1900-01-01至2019-05-27

检测结果

去除本人已发表文献复制比：0.7% 跨语言检测结果：0%

去除引用文献复制比：0.7% 总文字复制比：0.7%

单篇最大文字复制比：0.3%（基于校园网流量的舆情热词提取及分类研究）

重复字数： [319]	总字数： [43429]	单篇最大重复字数： [118]
总段落数： [4]	前部重合字数：[183]	疑似段落最大重合字数：[183]
疑似段落数：[3]	后部重合字数：[136]	疑似段落最小重合字数：[30]

指 标：☐疑似剽窃观点 ☑疑似剽窃文字表述 ☐疑似自我剽窃 ☐疑似整体剽窃 ☐过度引用

表 格：0 公 式：9 疑似文字的图片：0 脚注与尾注：0

1.9%（183） 7145807_刘嘉挺_法律问题关键词抽取系统_第1部分（总9699字）
0.3%（30） 7145807_刘嘉挺_法律问题关键词抽取系统_第2部分（总9429字）
0%（0） 7145807_刘嘉挺_法律问题关键词抽取系统_第3部分（总16207字）
1.3%（106） 7145807_刘嘉挺_法律问题关键词抽取系统_第4部分（总8094字）

（注释：无问题部分 文字复制部分 引用部分）

指导教师审查结果

指导教师: --
审阅结果:
审阅意见: 指导老师未填写审阅意见

1. 7145807_刘嘉挺_法律问题关键词抽取系统_第1部分

总字数：9699

相似文献列表

去除本人已发表文献复制比：1.9%(183) 文字复制比：1.9%(183) 疑似剽窃观点：(0)

1	基于校园网流量的舆情热词提取及分类研究 镇佳(导师：朱国胜) -《湖北大学博士论文》- 2018-04-09	1.2%（118） 是否引证：否
2	高维数据可视化方法研究 马豪(导师：吴玲达;魏迎梅) -《国防科学技术大学博士论文》- 2016-11-01	0.7%（72） 是否引证：否
3	生物网络中的模体发现算法研究 丁吕(导师：骆嘉伟) -《湖南大学博士论文》- 2018-05-09	0.4%（43） 是否引证：否

4	基于多元信息融合的药物重定位算法研究	0.4%（40）
	朱桥(导师：骆嘉伟) -《湖南大学博士论文》- 2018-05-09	是否引证：否
5	空间辐射磁屏蔽防护机理研究	0.4%（40）
	郑宏霞(导师：黄朝艳) -《南京航空航天大学博士论文》- 2018-03-01	是否引证：否
6	基于J2EE技术的车险盈利系统的设计与实现	0.4%（40）
	赖鑫亮(导师：夏侯建兵) -《厦门大学博士论文》- 2017-10-01	是否引证：否
7	基于尾字词典的逆向回溯中文分词技术研究	0.3%（32）
	梁桢(导师：李禹生) -《武汉工业学院博士论文》- 2010-05-01	是否引证：否

原文内容

法律问题关键词抽取系统

摘要

法律问题关键词抽取是当前关键词抽取技术在我国法律范畴内的应用，法律问题与人们日常生活以及社会规范制度息息相关。目前我国的法律文件及条款在我国人口众多、社会情况复杂等诸多原因下不断完善且数据量庞大，且伴随着高速网络信息和大数据时代的到来，绝大部分的人们由于专业限制通过使用各种在线法律问答系统来咨询自身所需的法律咨询或援助。如何从法律问答库的海量数据中抽取出人们需要的、准确的数据已经是当前研究的重点。关键词抽取是法律信息采掘的最首要任务，准确的关键词能够快速引导人们关于法律问题的参考方向，能够告知人们某个法律问题所属的具体法律条款范畴。由于当前关键词抽取技术大多在长文本中基于统计学的方法，其在简短的文本如法律问答方面的关键词抽取并无优势。

因此本文主要通过构建一个seq2seq模型并在其中加入强化学习，从某一简短的法律问题中生成关键词。首先从法律问答网站上利用爬虫技术收集大量法律问题以及已经人为对应标记好的关键词，并对（问题-关键词）语料库做预处理。对文本数据进行中文分词之后，利用word2vec模型将文本中每个词转换为词向量表示。然后使用已经构建好的seq2seq模型进行训练，从而能够让训练好的模型预测生成准确的关键词。

关键词：关键词抽取；神经网络；seq2seq模型；word2vec

LEGAL ISSUES KEYWORDS EXTRACTION SYSTEM

ABSTRACT

Keywords extraction of legal issues is the application of current keyword extraction technology in China's legal scope,legal issues are closely related to people's daily lives and social norms.At present, China's legal documents and terms are constantly improving and the amount of data is huge due to many reasons such as China's large population and complex social conditions.The vast majority of people seek their own legal advice or assistance through the use of various online legal question and answer systems due to professional restrictions.How to extract the accurate data that people need from the large amount of data of the legal question and answer library is the focus of current research.Since most of the current keyword extraction techniques are based on statistical methods in long texts, the keyword extraction in legal questions and answers (short text) is not good effective.

Therefore, this paper mainly generates keywords from a short legal issue by constructing a sequence-to-sequence (seq2seq) model based on the Recurrent neural network (RNN).First, from the legal Q&A website, we use Crawler technology to collect a large number of legal questions and keywords that have been artificially labeled, and pre-process the (question-keyword) corpus.After the Chinese word segmentation of the corpus, the word2vec model is used to generate the word vector. Then use the already built seq2seq model to train, so that the trained model can predicts accurate keywords.

Key words: Keyword extraction; Neural Networks; seq2seq model; Word2vec;

目录

1 绪论.....1
1.1 课题背景及研究的目的.....1
1.2 国内外研究现状及发展趋势.....1
1.2.1 国内外研究现状.....1
1.2.2 发展趋势.....3
1.3 论文组织结构.....3
2 开发工具及相关技术.....5
2.1 系统开发工具.....5
2.1.1 Python3.6.....5
2.1.2 Pycharm.....5
2.1.3 NVIDIA GEFORCE GTX 850M.....5
2.2 中文分词.....5
2.2.1 基于字符匹配的分词方法.....5
2.2.2 基于理解的分词方法.....6
2.2.3 基于统计的分词方法.....7
2.2.4 jieba分词方法.....8
2.3 词嵌入技术.....8
2.3.1 word2vec技术.....9
2.3.2 CBOW模型和Skip-gram模型.....9
2.4 Pytorch机器学习框架.....10
3 带注意力机制的seq2seq模型.....12

3.1 循环神经网络……12
3.1.1 神经网络的概念……12
3.1.2 RNN模型结构及原理……13
3.1.3 LSTM与GRU……15
3.2 seq2seq模型原理……18
3.3 Attention机制……18
4 系统分析……21
4.1 数据需求分析……21
4.2 功能需求分析……21
4.3 性能需求分析……22
5 系统实现……23
5.1 语料收集和预处理……23
5.1.1 数据爬虫……23
5.1.2 数据预处理……24
5.2 中文分词……24
5.3 词嵌入……25
5.4 基于强化学习的seq2seq模型……25
5.4.1 Encoder-Decoder……26
5.4.2 seq2seq模型……26
5.4.3 attention机制……26
5.4.4 强化学习……27
5.5 训练模型……28
6 系统评估与演示……29
6.1 系统评估……29
6.2 系统演示……30
7 总结……31
参考文献……32
致谢……34
附录:程序主要源代码……35

1 绪论

1.1 课题背景及研究的目的

在互联网大数据时代背景下，互联网使用人数以及各种终端设备如智能手机、平板、笔记本爆发式增长。信息时代为人们带来便捷的同时，网络上庞大且冗杂的数据也让人们常常需要花费许多精力去筛选得到想要的信息。百度、Google等搜索引擎在一定程度上协助人们在如此庞复的网络数据中找到需要的信息，且只能根据关键词进行检索，且在日益增长的数据量情况下对关键词精准度的要求也越来越高。关键词是简短且具有高度总结性的文本，需要用户根据自身知识来概括提取。但在法律问题方面，由于用户专业限制，无法保证依靠人工提取关键词的精准性，从而也影响了搜索引擎返回信息结果的准确性。

我国遵守以法制治国的策略，以保障社会治安稳定、社会文明进步以及国家长治久安。随着我国法律制度体系越来越完善，法律条款及文件的数量越来越大且复杂，非法律专业人士由于专业限制无法定位具体的法律范畴。法律社区问答网站（如：www.51wf.com）的出现给了人们一个很好的咨询的途径，用户在网站上提交一段较为简短、口语化的法律问题，会有法律专业人士提供非常专业的回答，并将该问题打上标签作为其关键词，能够迅速定位用户所咨询问题所属的法律专业范畴。本课题研究的目的是通过收集上述法律社区问答网站上的大量法律问题-关键词数据，运用神经网络的关键词提取等技术，得到一个可准确挖掘出法律问题中包含的关键词的模型。该模型能够根据用户输入的法律问题，自动生成关键词，帮助用户定位问题的范畴并指引用户搜索的方向。

1.2 国内外研究现状及发展趋势

1.2.1 国内外研究现状

关键词是一段简短的总结性内容，用来表达文本的主要语义，高质量的关键词可以促进对文本内容的理解，组织和访问。因此，国外许多研究都集中在从文本内容中自动提取关键短语的方法，它已广泛应用于许多应用。如Jones等提出的信息检索[1]、Hulth等提出的文本分类等应用[2]。大多数现有的关键词提取算法通过两个步骤解决了这个问题，第一步是获取关键词候选列表，研究人员试图使用具有某些词性模式的名词短语来识别潜在的候选关键词，如Liu等通过去除停用词并提取特定词性的词[3]，Medelyan从维基百科等重要语料库中提取n-gram[4]，并使用预先建立的条例抽取短语。第二步是通过有监督或无监督的机器学习方法和一系列人为定义的特征将候选关键词对文本内容的重要性进行排名。在带监督的机器学习中，关键词抽取的工作可以归为文本分类应用，如Frank使用朴素贝叶斯训练分类器[5]，Turney等提出使用决策树算法[6]，Tang等人使用SVM算法[7]。在无监督的机器学习中，Mihalcea等人提出计算候选关键短语之间的相关性[8]，EL-Beltagy使用KP-Miner算法[9]，通过频度统计得出待选词的重要程度。

上述关键词提取方法主要存在两个主要缺点：首先，这些方法只能提取源文本中出现过的关键词；它们无法预测具有略微不同顺序的有意义的关键词或使用同义词的关键词。然而，法律问答系统通常根据其语义来分配关键词[10]，而不是根据文本内容。由于法律问题是由非专业用户口语化提出的，关键词往往不包含在问题中，所以通过上述的方法提取效果不佳，这进一步促使开发更强大的关键词预测模型。其次，将待选关键词排名时，以前的方法一般采用TF-IDF和PageRank。但是，这些方法仅基于词语共现的次数统计来检测文档中每个词的权重，不能反映作为文本内容基础的完整语义。为了解决短文本中关键词难以使用统计方法抽取的问题，诸多学者已着手研究从短文本中生成关键词的技术，如Zhang等人提出了一种用于短文本提

取关键词的递归神经网络模型[11]，Bellaachia等提出的基于图形密钥的方法抽取Twitter中的关键词[12]，Meng等提出通过CopyRNN实现深度关键词抽取[13]，Ranzato等利用序列模型训练自动生成关键词[14]。

在我国，关键词的抽取技术处于刚起步研究阶段。由于中文不像英文句法那样有天然的空格作为词的界限，且中文是世界上语法最复杂困难的语言之一，这也使得抽取中文关键词变得更困难。在中文关键词抽取之前，需要对语.料样本进行分词处理，目前许多研究人员已经提出若干优秀的中文分词工具，如HanLP、jieba分词、FudanNLP等。在现有的中文分词技术上，我国研究人员贡献了许多中文关键词抽取方法，如张建娥利用改进后TF-IDF算法并结合词语的相关性实现中文关键词抽取[15]，李素建等将最大熵模型应用于关键词的标注任务[16]，王林玉等提出利用卷积神经网络和TP-ISP算法实现实体抽取方法[17]，王锦波等人利用一种改善词频特性统计的朴素贝叶斯算法[18]，提高关键词的可读性。在根据语义的关键词抽取技术方面，王立霞等提出使用《同义词词林》得到词典中每个词语的相关程度[19],将其映射到语义相关程度网络中,使用中间密度来表示每个词语的对句子的重要性，在关键词抽取的过程中能够很好的考虑到每个词语的语义。

1.2.2 发展趋势

由于我国社会情况和地方文化复杂，我国的法律文献及条款众多并且将来还会不断的修订和完善。因此整个法律体系所涉及的法律问题类型分类以及各种法律问题检索的复杂度会越来越高。当人们碰到法律问题时往往只能够做出口语化的描述，并不能准确的定位某个法律问题所属的法律专业范畴。法律社区问答网站的出现使人们咨询法律的途径越来越便捷，但是同时由于网络信息数量巨大，且大部分法律社区问答网站需要通过专业人士人为解答、人为标记关键词，网站无法立即反馈准确关键词信息给用户。当前的法律问题关键词提取技术和法律问题分类还不是特别完善，这就有可能产生用户被误导，混淆法律概念的后果，因此法律问题关键词提取方面还需进一步探索，主要体现在：

（1）准确性：法律问题与个人、公司或部门、政府等单位的利益以及社会制度密切相关，当涉及到这些内容时，法律条款和文件往往是决定性的因素。这就要求法律问题关键词抽取系统能够给出精准的关键词来定位问题所属法律范畴。

（2）易用性：法律问题一般是由非专业人士提出的，由于我国法律文件包含的专业词汇数量巨大，普通人无法准确使用专业词汇描述某一法律问题，描述内容一般偏口语化。再者中文本身语法复杂多变且不断有新的网络流行词汇诞生，所以如何从用户口语化描述的问题中提取专业法律词汇关键词将成为今后研究的重点。

（3）完整性：社会不断的发展，我国法律文献也随之不断修订和完善。法律关键词抽取模型必须参照当前最新最完整的法律文献库，这就要求研究人员以及专业法律机构共同努力收集全面的法律语料，保证模型能够涵盖所有用户需要的法律问题关键词。

1.3 论文组织结构

本文分为7章，每章内容安排分别如下：

第1章绪论：主要分析了法律问题关键词抽取系统的背景及研究目的、国内外研究现状和发展趋势，并说明了本文的总体组织结构。

第2章开发工具及相关技术：介绍本系统使用的开发工具，中文分词技术、词嵌入技术以及Pytorch机器学习框架等相关技术。

第3章带注意力机制的seq2seq模型：主要介绍了循环神经网络、seq2seq模型结构、LSTM与GRU神经元以及Attention机制等算法思路。

第4章系统分析：通过数据需求、功能需求、性能需求三个部分对系统进行整体分析。

第5章系统实现：根据前面章节的思路以及对每个功能需求分析，对设计模型、搭建模型、训练模型的具体实现方法进行论述。

第6章系统评估与演示：根据训练好的模型，编写可视化演示程序，并随机抽样一部分数据在模型上进行测试来评估模型的准确性和完整性。

第7章总结：总结论文内容，并指明系统目前存在的不足。

2 开发工具及相关技术

2.1 系统开发工具

2.1.1 Python3.6

Python语言的易学性、可移植性以及拥有丰富强大的标准库等特点，使其成为在进行机器学习、神经网络等开发时的首选编程语言。目前许多优秀的开源机器学习框架都是基于Python语言编写的。

2.1.2 Pycharm

Pycharm拥有完整的Python集成.开发环境，如图形化界面、代码编辑器、解释器、调试器，在编程时能够大大提高开发效率以及调试效率。

2.1.3 NVIDIA GEFORCE GTX 850M

GeForce GTX 850M 是伟英达（NVIDIA ）公司旗下出品的一款中端笔记本级别显卡。该款显卡提供CUDA等技术支持，CUDA是伟英达公司发行的一款并行运算模型。它使用GPU的并行运算性能，最大限度提升机器的计算能力使用率。由于本课题所训练的语料数据量较大，使用CPU训练效率低，故在机器学习过程中使用GPU来提高训练速度。

2.2 中文分词

中文分词，即把一段中文语句按照句子本身语义分割为以词语为基本单元的词序列。在英文句子语法中，每个词语之间都使用空格当作分隔界限，然而中文句子中的词、字并没有天然的分界，且基于中文语法复杂的特点，同一中文句子中的词语可能有存在歧义的不同分割法。法律问题往往包含许多专业词汇的简称，进一步加大了词语划分的难度，故清晰准确的中文分词是本系统语料处理工作的首要指标。

2.2.1 基于字符匹配的分词方法

基于字符匹配的分词方法，是将需要分词的中文语句在一个已创建好的足够大的词典中根据特定的方法进行词语匹配。若匹配了一个词语，证明词典发现了该词，就将其分割。这种匹配的算法根据扫描的方式又分成以下四种匹配法：

（1）最大正向匹配算法

最大正向匹配算法，指的是利用类似于滑动窗口概念从左到右滑动依次匹配，该方法首先给定一个词典中最大词条的长

度作为滑动窗口的长度L，根据滑动窗口的位置取出对应的L个中文字符进行匹配。若未匹配，则L-1并重复上述步骤，直到匹配成功为止。

（2）最大逆向匹配算法

最大逆向匹配算法，即从右往左对中文语句进行最大匹配。该算法是正向匹配的逆思维，根据实验结果[20]以及中文语法的特点表明最大逆向匹配算法效果要相对于最大正向匹配算法效果更好。

（3）最大双向匹配算法

大部分情况下，前向匹配和后向匹配得到的结果是相同的，但是也有不同的情况。双向最大匹配就是对一个句子的逆向和正向一起进行最大匹配，接着对比两种匹配结果。若两者匹配的结果相同，就说明两种算法对句子语义的理解没有分歧；若不相同，就需要找到出现语义分歧的部分进一步处理。该算法的优点在于使分词结果更加准确，能够部分解决句子出现分歧语义的问题。缺点在于同一个句子需要正向和逆向各匹配一次，增加了计算量和时间复杂度。

（4）最少切分算法

不同于前述三种算法，最少分切算法使用动态规划的概念，将分词任务划分成局部问题，最大限度地减少每个句子中分割出来的词语数量。该算法需要遍历中文语句所有可能的切分路径，这也导致了计算量过大的缺点。

2.2.2 基于理解的分词方法

基于理解的分词方法，其核心思想就是让机器模拟人类大脑的思维方式来理解句子、分割词语。在对中文句子的语义剖析后进行分词，达到能够处理分词过程中歧义部分的效果，故这种方法也常称为基于语义的分词法。

该方法需要建立一个足够大的词典并且词典中包含所有词语与之对应的所有语义信息，然后根据某种匹配算法分割出子串S，若S与词典中某个词W相匹配，则取出W的语义信息并结合句子对S进行语义分析。若符合该句子的语义信息，则说明分析正确，接着分割句子剩余部分。

这种方法的缺点在于，需要构建一个庞大的语言知识和语义信息库。基于汉语语法的多样性、歧义性，在法律问答等口语化情境下，难以使用标准化句法结构来分析口语化句型。

2.2.3 基于统计的分词方法

基于统计的分词方法，其核心是将每个词语分割为单个汉字。假设某段汉字序列由n个连续汉字构成，在不同的统计文本中，若这n个连续汉字出现的频率越大，则代表这n个连续汉字组成一个词语的概率就越大。一般分为两个步骤，首先通过N-Gram算法构建词语统计模型；第二步，将语句分割为词语序列，接着计算分割结果的概率，得到概率.最大的.分割方法，这一过程使用统计学方法。

设有一段长度为i的中文语句S：

(2-1)

S的所有可能的切分路径：

(2-2)

目的是找出使得最大的分割方法:

(2-3)

根据Bayes公式：

(2-4)

计算得出：

(2-5)

对于n-gram二元结构，第n个词仅与之前一个词有联系，则有：

(2-6)

该方法的优势在于不必构建一个巨大的词典，不受法律等特殊范畴的词汇约束，并且具有分析语义信息，处理歧义内容的功能。缺点在于统计过程中需要一个庞大的语料库作为依据，并且容易抽取出对句子语义影响不大但共现概率很大的一些停用词，如：“同一”，“白白”。

2.2.4 jieba分词方法

jieba是一个综合了统计的和字符匹配的算法的优点的中文分词组件，它包含一个很大很全的词典，即dict.txt文件。并且支持添加自定义词典以及调整词典的功能，能够有效解决法律专业词汇缩写无法被词典识别的问题。它支持精确、搜索引擎、全模式这三种分词。基于本课题是从简短法律问题中抽取关键词并分析文本内容，需要最大限度地将语句准确分割，故选用精确模式。

综合前述对三种分词类别各自优缺点的分析，以及jieba库强大的功能组件且比较适合法律问题具体的句法情境，本系统使用jieba库进行法律语料数据的中文分词工作。

2.3 词嵌入技术

One-Hot.编码是词语向量的.最基本的表示方法，它通过利用M个多维向量来将M个词语编码，每个词的向量值唯一的。并且每个多维词向量中仅有一位的值为1，剩余位的值都是0。假如“盗窃”和“抢劫”对应词典中的序号分别为2和4，对应的One-Hot编码为：

盗窃：[0,1,0,0,0,…,0]

抢劫：[0,0,0,1,0,…,0]

使用这种编码的优点在于简洁易懂，但是假设在有2万个词汇构成的词典中，词向量的维度将达到2万，尽管其中只有一位是有效位。这种情况下有可能造成词向量过于稀疏和维度爆炸的后果。词的分布式表示,就是使用某种特定模型训练，把每个词语投影到一个长度较短、密度较大的向量上，通过密集低维的向量有效解决维度爆炸的问题，这个一操作又称为词嵌入（Word Embedding）。

2.3.1 word2vec技术

word2vec是谷歌开源发布的计算词向量的技术，它能够对数百万级字典和数亿个数据集进行高效训练。该工具训练得到的词向量可以非常好地衡量单词和单词之间的相似度。它包括.两种模型： CBOW与Skip-Gram。

2.3.2 CBOW模型和Skip-gram模型

（1）CBOW模型

CBOW.（Continuous .Bag-of-Word），该模型的特点在于输入词序列中某个目标词X的上下文词语，输出对X的预测，其模型结构图2.1所示：

图2.1 CBOW 模型网络结构图

模型中，Input layer内容为所有词语的One-Hot编码，Hidden layer内容为输入层所有向量相加和的平均值：

(2-7)

传统模型Output layer采用softmax函数得到词语分布概率，缺点是算法的.复杂度高，耗时长。所以CBOW使用Hierarchy Softmax来提高运算效率，该方法思路是在模型输出层建立一棵哈夫曼树。根据哈夫曼树的特点，词典中每个词代表一个叶子节点，从而计算量由单词个数降到路径条数。

（2）Skip-Gram模型

与CBOW相反，Skip-Gram根据输入的某个词语X输出对其目标上下文词语，其模型结构图2.2所示：

图2.2 Skip-gram模型网络结构图

模型中，Input layer内容为目标单词X的One-Hot编码，Hidden layer内容为X对应权重矩阵W的向量：

(2-8)

基于本课题的特点，法律问题语料库结构为输入多个词构成的序列，输出一个或多个关键词。简单对比两个模型可以得知CBOW模型更加适合法律问题关键词抽取任务。

2.4 Pytorch机器学习框架

Pytorch是Facebook开源的一款机器学习框架，它具有简洁灵活、优雅易用、高效快速等优势。其结构为tensor→autograd→nn.Module 三个从下到上的抽象结构层次，即(张量→自动求导→神经网络)。其与TensorFlow主要有以下差别：

（1）动态定义和静态定义

二者都使用张量（tensor）进行计算，且都将模型定义为DAG（有向.非循环图）。但TensorFlow需要在起始阶段静态地定义DAG，无法自适应序列的长度，而Pytorch在使用动态的方式定义图，在序列长度无法确定的情况下能够方便更改、定义和执行。

（2）调试

Pytorch在执行过程中能够动态定义，所以能够直接对其调试，而TensorFlow在调试过程中仅能观察到模型内的操作和张量，无法调试python代码。

（3）数据加载

Pytorch封装了完整的数据集和取样器API接口，方便训练时加载数据、数据迭代，而TensorFlow在这方面目前并无非常实用的工具，且其API设计十分复杂冗长，难以学习。

由于Pytorch数据加载方便、序列长度自适应、开发调试体验好等优点，且在易用性方面，其代码量较少、功能更加直观，更加符合人类思维，所以Pytorch比较适合本系统的开发。

3 带注意力机制的seq2seq模型

一段法律问题文本根据人类思维可以分为法律范畴决定词和语义铺垫词，例如有这样一则法律问题：“我在工.作时不小.心发生了交.通事故，该由谁赔偿？”，以人类的注意力思维可以得出“交通事故”、“赔偿”是这此问题所属法律范畴的决定性词语，而“不小心”、“发生”等词对该问题语义影响不大。为了使机器能够模拟人类的注意力思维来提高学习效率，本文使用了一种带注.意力（Attention）.机制的seq2seq.模型。

3.1 循环神经网络

3.1.1 神经网络的概念

神经网络（Neural Network）是指使用计算机技术来模拟人类大脑神经元网络，它能够使计算机具有人脑神经结构特征，来模仿人对物理世界的理解与交互。

指　标

疑似剽窃文字表述

1. 1.3 论文组织结构
   本文分为7章，每章内容安排分别如下：
   第1章绪论：主要

2. 7145807_刘嘉挺_法律问题关键词抽取系统_第2部分　　总字数：9429

相似文献列表

去除本人已发表文献复制比：0.3%(30)　　文字复制比：0.3%(30)　　疑似剽窃观点：(0)

1 一种新的忆阻混沌系统的构造、分析与实现　　0.3%（30）

张立言 - 《大学生论文联合比对库》- 2018-06-04　　是否引证：否

原文内容

其组成主要有以下部分：

（1）神经细胞模型

在动物体的神经.网络中，每个神经细胞（cell）都连.接了多个其他神经.细胞。当某个神经细胞被激发时，它会反馈信息

给其所连接的其他神经细胞，使得这些神经细胞状态得以更新。若某个神经细胞的兴奋值超过一个特定的值，那么它将被激活，使得该神经细胞状态置为兴奋并告知大脑。将其抽象出来，如图3.1所示：

图3.1 神经细胞模型

（2）权重和阈值

权重（weight），在图3.1中，x1~x3为输入值，w1~w3为对应的权重值。输入和权重可以分别理解为现实世界中产生某个结果的决定性因素，和每个决定性因素对结果的重要性。

阈值（bias）是神经元被激活与否的分界线，它的大小更够直观的反映正（负）激励某个神经细胞的难易程度。阈值越高表明神经细胞越难被激活，反之阈值越低则神经细胞越容易被激活。

（3）激励函数

激励函数（Activation functions），如图3.2所示神经网络中，输入的序列x1~x4经过加上对应的权重w1~w4求和后，还在输出层之前还经过了一个激励函数的运算。使用激活函数的必要性体现在当一个神经网络具有多层结构时，若无激励函数的运算，输入和输出都是矩阵相乘的线性结果。

图3.2 神经网络激活函数

3.1.2 RNN模型结构及原理

（1）单层神经网络

通过前述对神经网络的结构分析，我们容易得到如图3.3所示的基本单层神经网络结构示意图，其中输入为x,箭头表示x通过激活函数f()与Wx+b变换后获得输出y。

图3.3 单层神经网络结构

（2）RNN

在上述的单层网络模型中，只能单独处理每个输入且每个输入的前后顺序和时间顺序完全无关。在自然语言处理过程中，单独地分析一句话中的每个词是行不通的，我们需要理解这些词连接起来的序列。序列数据用单层网络模型处理效果不佳，因此使用RNN.（Recurre.nt Neural Network，循环神经网络），其结构如图3.4所示：

图3.4 RNN结构示意图

其中，x1~x4为输入序列，h1~h4为对应输入的隐含状态，y1~y4为对应输出，h0为初始状态，箭头代表向量做了一次计算。隐状态h2=f(Ux2+Wh1+b)， U、W、b为计算h的共享参数，以此类推。输出y2=softmax(Vh1+c),以此类推，可以看出每个输出的结果与当前输入和上一次输入都有关系，它能够将整个序列的词语相关联。

以上为RNN最基.本的一种结构（M v.s M），它要求输入、输出序列的长度一样，这就导致了该结构的局限性。因此这里介绍RNN的另外两种结构：

① M vs 1

在某些应用领域，将序列输入模型，目标输出的是单个值，建模时可以仅仅在输入序列对应的最后一个隐状态进行变.换输.出即可，如图3.5所示：

图3.5 N vs 1结构

该结构在在处理序列分类如判断句子感情分类等应用中使用较为广泛。

② 1 vs M

与M vs 1结构恰好相反，1 vs M.结构的输入为单个值，输出则为一个序列，建模时可以将单个的输入分别当作每个时刻地输入，如图3.6所示：

图3.6 1 vs N结构

该结构能够处理的问题有如利用情感类别生成评论数据等。

3.1.3 LSTM与GRU

上述分析表明RNN能够考虑到序列中上下文词语的相关程度，但当序列比较长时，RNN不能解析出序列首尾词语的关联程度，同时也可能存在由参数值引起的梯度弥散或梯度爆炸的现象。为了处理这类长时间依赖的缺点，研究人员提出了两种改进版RNN模型，LSTM和GRU。

（1）LSTM

LSTM.（Long. Short-.Term. Memory）模型，它将原始RNN的隐藏层神经元替换为LSTM神经元，它由1个记忆细胞和3个控制门，即Input. Gate（输入门）、Output .Gate（输出门）、Forget. Gate（忘记门）共同构成，其结构如图3.7所示。

图3.7 LSTM模型结构

在该结构中，用f、i、o分别代表代表忘记门控制信息、输入门信息、输出门的衰减信息，C表示整个LSTM神经元的状态值，W代表对应的权重，σ是激活函数。那么在t时刻忘记门ft 的计算如式3-1。

（3-1）

输入门it 公式如式3-2：

（3-2）

当前神经元状态计算如式3-3：

（3-3）

然后将上一个状态Ct-1更新为Ct，如式3-4：

（3-4）

输出门的公式如式3-5：

（3-5）

由上述步骤得出，t时刻LSTM单元的最终输出如式3-6：

（3-6）

LSTM按照上述的时间序列展开，通过3个控制门对历史记忆进行保留与丢弃操作，能够很好的处理长时依存的缺点。

（2）GRU

GRU.（Gated. Recurrent .Unit）模型，其本质为LSTM的一种简化模式。与LSTM不同的是，它仅有Update Gate（更新门）和Reset Gate（重置门）作为控制门，且它没有单独的记忆单元，而是直接在隐藏状态操作线性积累结果，其结构如图3.8所示。

图3.8 GRU模型结构

GRU传播过程如式3-7：

（3-7）

其中，zt为t时刻更新门的输出，根据上一时刻的状态判断当前需要更新多少内容，rt为重置门的输出，根据上一时刻的状态判断当前状态需要重置与否。ht为当前GRU.输出，W为对应的系数，ht为当前重置门控制的GRU待选.输出。相比于LSTM，GRU同样能够使每个循环单元自适应地捕获不同时间尺度的依赖性,但GRU的模型结构更为简化，且加深了各层的理解和不同状态之间的联系。

3.2 seq2seq模型原理

前述分析了RNN常见的3种结构：M vs M、M vs 1、1 vs M。在如图3.9所示法律问题中，法律问题的词序列作为输入，关键词标签作为输出。

图3.9 法律问题-关键词结构

可以看出，输入的序列对应的输出可以对应单个关键词，也可以是一个关键词序列，所以在处理法律问题关键词方面，上述的三种结构都不能满足要求。因此本文引入一个RNN特殊的形式：N vs M结构，也称为Encoder.-Decoder（编码器.-解码器）模型或者.seq2seq模型。如图3.10所示：

图3.10 N vs M结构

首先，原始序列被Encoder压缩成一个包含上下文语义的向量 c,接下来Decoder对它进行解码，在解码的过程中当前时刻t不断以上一时刻t-1的输出作为参数，循环解码直至输出停止符。

3.3 Attention机制

在Encoder.-Decoder模型中，由于 c 中储存了原始序列的所有语义内容，若是在句子字数比较多的情况下，一些对语义影响不大的词语也被压入c中，导致精度和性能的下降。为了处理这类问题，本文使用了Atten.tion（注意力）.机制。

假设有这样一则法律问题：“盗窃自行车，判刑多久？”，我们要从中抽取出关键词“盗窃罪”和“量刑标准”，注意力在各个阶段以不同的c作为输入，其原理如图3.11所示。

图3.11 attention机制

h1~h4为原始序列对应的隐藏状态，c1~c3为Decoder在不同输入阶段的上下文向量，w11~w14为每个对应隐状态的权重。假设h1~h4对应“偷盗”、“自行车”、“判刑”、“多久”的编码，Decoder在时间t=1时刻通过4个权重w11~w14计算第一个上下文向量c1：

(3-8)

再将与原始隐含层最后一个隐含向量连接构成一个新向量输入新的隐含层：

(3-9)

当时刻以此类推得到，与共同构成一个新的隐含层，由于权重w11~w14各不相同，就形成了Decoder对Eecoder不同输入的注意力，且权重越大表示Decoder在时刻越重视该输入。

经过上述构建模型，就可以将关键词提取任务转换为权重矩阵计算任务，假设表示Encoder状.态，表示Decoder状.态，表示注意力层输出的Decoder状态，那么时刻对Encoder隐含层每一个的权重的计算如下：

(3-10)

一般使用一下方式计算：

(3-11)

其中tanh函数在pytorch中通过torch.nn.Tanh调用，参数需要模型通过学习得到。

4 系统分析

4.1 数据需求分析

本系统需要从问法网（www.51wf.com）上收集大量法律问题语料，所收集的语料中每条数据必须包含：用户提出的法律问题，网站对该问题添加的标签。问法网的“问法知道”模块提供了数据支持，该网站的“问法知道”页面目前提供了从2008/12/30至2019/5/17期间所有用户提出的问题数据，其中分为“已解决法律咨询”和“待解决法律咨询”两类数据。

“已解决法律咨询”分类下的问题表示该问题已经得到专业律师的解答，用户根据其解答已经解决了所提出的法律问题，并且问法网也对该问题定义了标签。

“待解决法律咨询”分类下的问题表示该问题暂未得到专业律师解答，用户提出的问题暂未得到解决，问法网也暂时未对其添加标签。

经过分析，只有“已解决法律咨询”分类下的数据对本系统才是有意义的，故在收集“问法知道”数据时，应该过滤掉未带标签的法律问题。通过以上数据需求分析，本次共收集16万4千余条数据，能够涵盖4722个法律专业词汇，基本满足本系统对训练数据集的要求。

4.2 功能需求分析

本系统通过使用Pytorch框架建立一个基于序列到序列模型，在其中加入强化学习，从而生成一个模型来预测法律问题这类短文本中包含的关键词。具体功能模块包括：

（1）语料收集与预处理

从法律社区问答网站（www.51wf.com）利用爬虫技术收集该网站所有的法律问题语料，并对数据进行特定格式化以及清理，提供给之后的模块训练。

（2）中文分词

采用综合了统计和字符匹配算法的jieba分词工具进行法律语料的分词工作，将分词后的结果提供给词嵌入模块进行训练生成词向量。

（3）词嵌入

将CB.OW作为本系统模型，对分词后的法律语料进行训练，生成所有词汇的向量模型，并确保每个词向量能够储存对应词语的基本语义。

（4）搭建基于强化学习的seq2seq模型

搭建带attention机制的seq2seq模型，使用GRU作为模型的基本单元，将处理完的语料作为输入输出序列，在其中加入强化.学习用来优化.训练的过程。

（5）训练模型

在NVIDIA GEFORCE GTX 850M上使用Pytorch调用其CUDA架构进行GPU加速训练，生成一个能够预测关键词的模型。

（6）系统评估与演示

采用Tkinter工具编写一个窗口程序，并根据训练好的模型，抽取一定数量的问答语料样例进行统计，将统计的测试结果按照一定规则进行模型准确性和完整性的评估。

4.3 性能需求分析

由于在.训练时中包含太多简单.的计算任务，使用CPU计算速率低.且容易造成.计算机死机导致.运行的程序奔溃。为了分析本系统实验.硬件环境中使用CPU与.GPU的差别，利用一套网上开源的.序列到序列模型项目，分别在本机CPU和GPU上运行。结果显示在4万条数据训练过程中，CPU用时6小时，而GPU仅用时11分钟。由实验结果.分析得出，本环境所使用的GPU计算速率为CPU的32倍，且训练耗时较短，基本满足对性能的要求。

5 系统实现

5.1 语料收集和预处理

5.1.1 数据爬虫

（1）链接获取

本实验数据从问法网（www.51wf.com）的“问法知道”页面抓取，首先分析该页面的链接格式，以当前最新问题为例，如图5.1所示：

图5.1 链接格式

每个问题的链接是由一个类似于-186519的id标识的静态页面，这个id为网站后台数据库中该问题的唯一标识号，所以我们可以通过id[00000 ~ 186519]获取所有问题的链接。

（2）页面解析

有了所有链接之后，下一步剖析每个链接以获取语料。以图5.2为例：

图5.2 页面解析

由分析可知，该页面的结构简单易读，问题和每个标签都有明显标识。故可以使用BeautifulSoup库和lxml快速解析，从页面中提取出（问题-标签）数据。

5.1.2 数据预处理

在抓取数据并写入本地文件的同时，还需要对数据进行一定的预处理以便能够提供规范的语聊供模型训练，共有以下几步：

（1）去除标点符号

使用re库的sub()函数能够去除数据中的所有标点符号。

（2）标识问题和标签分隔符

在获取每条数据时，通过在“问题”和“标签”之间添加‘\t’字符加以分割。

（3）脱敏处理

语料中存在许多的阿拉伯数字，有可能对训练精度造成一定影响，要对数字进行一定的脱敏操作，使用notepad等文本编辑器，通过正则表达式将所有数字替.换为<num>。

5.2 中文分词

中文分词是本系统的首要任务，准确的分词结果能够使模型训练结果更加精确。本系统以jieba库作.为语料的中文分词工具，主要分为以下两步：

（1）添加法律词条扩展词典

在遇到某个比较长的法律词汇，在口语化时人们通常会使用缩写，比如“律师事务所”经常被缩写为“律所”。假设现在有一则法律问题中包含“违规律所”一词（即违规律师事务所），分词器由于字典限制有可能会将其分割为“违/规律/所”，这并不是句子真正的语义。为了解决这一问题，使用自定义法律词条词典，以扩展词典里的词汇。

问法网目前提供了共213944个专业的法律词条数据，抓取这些词条并写入法律扩展词典中。图5.3展示了添加法律词条扩展词典前后的分词结果：

图5.3添加法律词条扩展词典前后的分词结果

（2）精确模式分词

使用jieba的精确分词模式可以将法律问题语句精准分词，分词结果样例见图5.4。

图5.4 分词结果样例

5.3 词嵌入

语料数据分词结束后，需要将所有词汇转换为词向量。使用gensim库封装的模型方法训练语料，维度定义为256。本次实验生成的.vector文件中包含的词向量共21465个，且每个词向量存储了其对应的语义内容，使用gensim库的most_similar()方法加以测试，测试结果表明通过词向量能够找到与某个词关联最大的词或者近义词。使用Matplotlib展示部分词向量结果如图5.5所示。

图5.5 部分词向量结果

5.4 基于强化学习的seq2seq模型

搭建seq2seq模型是本系统最重要的模块，本系统使用IBM在2018年开源的pytorch-seq2seq框架，它提供了更加灵活的模型选项以及强大的可扩展元素，应用在seq2seq模型的学习、推测和检查点等。

5.4.1 Encoder-Decoder

（1）Encoder（编码器）

使用EncoderRNN()函数创建一个编码器RNN，将其应用于输入序列。其思路是定义词典大小、最大序列长度、隐含层特征个数等参数，同时传入之前已经训练好的词向量。该编码器的输入是语料库的原始序列的列表，列表长度为自定义的batch_size（批量大小）。输出包括原始输入序列的特征张量，和包含隐状态h特征的张量。

（2）Decoder（解码器）

使用Decoder.RNN()函数构建一个解码器RNN，将其应用在seq2.seq模型解码功能上。思路是将Encoder编码的结果作为输入，定义序列开始和结束标志符(<SOS>,<EOS>)作为解码的标志位进行解码。其输入包括Encoder中原始输入序列的特征张量和隐状态h特征张量，以及用于从RNN隐藏状态生成标记的激活函数（一般使用softmax）。

5.4.2 seq2seq模型

在定义好Encoder-.Decoder后，将其共同构建为一个标准化的seq.2seq模型。在编码过程中，编码器每一步输入的词构成输入序列 ,RNN将其转换为上下文向量c，编码过程如公式5-1：

false (5-1)

其中，ht为在第t步输入时的隐.状态，将编码后的结果输入解码器进行解码过程，解码器将上下文向量c重新扩展成序列Y=[y1,y2,…,ys]，解码过程如公式5-2：

(5-2)

由式5-2可以看出，解码器在第t步时的隐含状态st与st-1、yt-1、c有关，输出yt与st、yt-1、c有关。

5.4.3 attention机制

在Decoder的输出功能上应用attention（注意力）机制，输入为包含解码器输出特征的张量作为上一次的输出状态（output），以及包含编码器输入序列特征的张量作为上下文语义（context）。输出为包含解码器已注意到的输出特征的张量（output*），以及注意力权重的张量（attn），计算公式如式5-3。

(5-3)

通过定义一个nn.Module的子类并重写其forward ()方法实现式5-3算法，如图5.6。

图5.6 实现attention算法部分代码

5.4.4 强化学习

强化学习，是指在机器训练的过程中，根据设定的奖惩规则不断尝试并从中找到规律的过程。当机器做出错误的动作时，给予一定的惩罚使之获得错误经验；当机器做出正确的动作时，给予一定的奖励使之获得激励。

它主要包括四个因素：Agent（智体）、Environment（环境）、action（动作）、reward（奖励）。其内容就是Agent根据当前的环境状态做相应的action，得到奖励后，将本身置为下一状态的过程，这个过程会一直循环下去直至终止状态。根据Markov判断过程，下一状态是由即将做出的action和当前状态共同决定的。本系统的Agent就是seq.2seq模型，模型生成的关键词作为action，生成的关键词用k表示。本课题属于文本处理问题，故选取基于策略梯度的PolicyGradient算法[21]，该算法思路如图5.7。

图5.7 PolicyGradient算法

设S为所有环境状态的集合，表示在t-1时刻的环境状况；设A为Agent能够做出的所有action的组集，表示在t-1时刻Agent即将做出的动作；设R为决策过程中每一步奖励值的集合，表示在t时刻根据t-1时刻的s和a做出的奖励（reward），

故视为一个学习步骤。本系统为每一步学习步骤设置的奖励策略如式5-4：

（5-4）

其中K表示目标关键词集合，表示动作a获得的奖励，为输出序列的长度。当Agent生成的关键词不在K中时（即生成了错误的关键词）或者输出了重复的关键词，我们给予0的奖励作为惩罚，这样reward（奖励）机制能够增加下一次好的action出现的概率，减少下一次不好的action出现的概率，达到强化学习的效果。

5.5 训练模型

本次实验通过使用pytorch的CUDA语义将模型以及变量放在NVIDIA GEFORCE GTX 850M GPU上训练，训练数据共16万条。以GRU模型作为基本单元，设置BATCH_SIZE(批数据大小)= 64，EPOCHS=8，训练过程共耗时1小时24分钟。

6 系统评估与演示

6.1 系统评估

本系统以生成关键词的准确率、召回率分别衡量模型的准确度、完整性。和分别表示目标关键词和模型生成的关键词，代表生成的关键词中符合要求的部分，定义如式6-1：

(6-1)

分别从训练样本和非训练样本中随机抽取500条数据按照上述方法进行计算，得出的结果如图6.1所示：

图6.1数据抽样测试结果

由抽样的1000条数据和结果得出，500条训练样本中，生成的关键词至少包含一个目标关键词的条数是481条（96.2%），准确率为76.91%，召回率为79.63%；500条非训练样本中，生成的关键词至少包含一个目标关键词的条数是452条（90.4%），准确率为73.26%，召回率为77.31%。经过样本分析，得出训练样本与非训练样本抽取结果的差异原因主要是由于目前网上的训练数据仅有16万条左右，不能够涵盖所有可能出现的法律关键词，导致非训练样本生成的关键词有一定的偏差。部分关键词生成样例见图6.2。

图6.2 关键词生成样例

6.2 系统演示

本系统通过Tkinter编写图形化窗口程序，通过用户输入的法律问题，调用已经训练好的模型抽取该问题对应的关键词，并显示在界面上。运行结果如图6.1。

图6.1 系统运行结果

7 总结

本系统基于目前关键词研究技术提出了一种应用于法律问题范畴的从短文本中生成关键词技术，目的是通过精确的关键词帮助人们提高法律检索的效率，帮助人们迅速定位自身法律问题所属的具体法律范围。本系统主要通过搭建一个带注意力的seq2seq模型，并在其编码器输出部分引入强化学习来优化模型训练，从而让模型能够准确抽取法律关键词。在开发前，先对目前我国各大法律社区问答网站进行调研，确定了问法网（www.51wf.com）提供的数据最适合作为本系统的训练样本。在系统分析过程中确定了课题的主要的功能有：法律问题语料抓取和预处理、中文分词、词嵌入、带注意力机制的seq2seq模型的搭建，强化学习。在系统环境方面，为本机配置了使用GPU加速训练的CUDA环境，大大提高训练速度，最后通过Tkinter编写图形化窗口程序。

通过从训练样本和非训练样本中分别抽样500条进行测试统计，结果表明本系统能够较为准确的从某一简短的法律问题中抽取关键词，但准确率并未达到目前互联网对数据准确性的要求，主要有以下两个原因：其一，由于自身时间和技术水平的限制以及大学期间没有太多机会接触机器学习等相关知识，导致在系统开发过程中往往更加注重系统的功能性，没能详细研究出一种适合本系统的最优算法。其二，问法网目前提供的数据量仅为16万条左右，并不能涵盖大多数的法律问题，从而约束了关键词抽取的范围。其三，通过分析本次从问法网上抓取的数据发现，样本中的关键词描述是用户和法律专家共同标注的，有许多关键词对于问题的描述也并不是很准确。

总体来说，本系统实现了课题的基本功能要求，但有些模块还需要做的更加完善，更加具有实用性。需要完善的要点主要有：（1）需要继续研究一种适合本课题场景的神经网络优化算法，在提高学习效率的同时也能让模型更加精确。（2）需要收集更多法律关键词语料数据，这也要求各个法律社区问答网站、我国法律专业人士以及问法人的共同努力，建立一个强大的、权威的法律问题关键词语料库来提供数据支持。

参考文献

Jones S , Staveley M S . Phrasier: a system for interactive document retrieval using keyphrases[J]. 1999:160-167.

Argaw A A , Hulth A , Megyesi B B . General-Purpose text categorization applied to the medical domain[J]. Stp.lingfil.uu.se, 2008:3-10.

Liu Z, Li P, Zheng Y, et al. Clustering to Find Exemplar Terms for Keyphrase Extraction[C]. empirical methods in natural language processing, 2009: 257-266.

Medelyan O, Frank E, Witten I H, et al. Human-competitive tagging using automatic keyphrase extraction[C]. empirical methods in natural language processing, 2009: 1318-1327.

Frank E, Paynter G W, Witten I H, et al. Domain-Specific Keyphrase Extraction[C]. international joint conference on artificial intelligence, 1999: 668-673.

Turney P D. Learning Algorithms for Keyphrase Extraction[J]. Information Retrieval, 2000, 2(4): 303-336.

Zhang K , Xu H , Tang J , et al. Keyword Extraction Using Support Vector Machine.[C]// International Conference on Advances in Web-age Information Management. Springer-Verlag, 2006:85-96.

Mihalcea R, Tarau P. TextRank: Bringing Order into Texts[C]. empirical methods in natural language processing, 2004: 404-411.

El-Beltagy S R . KP-Miner: A Simple System for Effective Keyphrase Extraction[C]// Innovations in Information Technology, 2006. IEEE, 2006:2-15.

廉鑫. 社区问答系统中若干关键问题研究[D]. 南开大学, 2014:11-15.

Zhang K , Xu H , Tang J , et al. Keyword Extraction Using Support Vector Machine.[C]// International Conference on Advances in Web-age Information Management. Springer-Verlag, 2006:85-96.

Bellaachia A , Al-Dhelaan M . NE-Rank:A Novel Graph-Based Keyphrase Extraction in Twitter[C]// IEEE/WIC/ACM International Conferences on Web Intelligence & Intelligent Agent Technology. IEEE, 2012:372-378.

Meng R, Zhao S, Han S, et al. Deep keyphrase generation[J].arXiv preprint arXiv:1704.06879, 2017:582-590.

Ranzato M, Chopra S, Auli M, et al. Sequence Level Training with Recurrent Neural Networks[J]. international conference on learning representations, 2016:1-10.

张建娥.基于TFIDF和词语关联度的中文关键词提取方法[J].情报科学,2012(10):110-112+123.

李素建, 王厚峰, 俞士汶, et al. 关键词自动标引的最大熵模型应用研究[J]. 计算机学报, 2004, 27(9):1192-1197.

王林玉,王莉,郑婷一.基于卷积神经网络和关键词策略的实体关系抽取方法[J].模式识别与人工智能,2017(05):83-90.

王锦波,王莲芝,高万林.一种改进的朴素贝叶斯关键词提取算法研究[J].计算机应用与软件,2014(2):174-176.

王立霞, 淮晓永. 基于语义的中文文本关键词提取算法[J]. 计算机工程, 2012, 38(1):1-4.

Sun M , T'Sou B K . Ambiguity Resolution in Chinese Word Segmentation[C]// 10 Th Pacific Asia Conference on Language. 1995:384-389.

Sutton R S, Mcallester D A, Singh S P, et al. Policy Gradient Methods for Reinforcement Learning with Function Approximation[C]. neural information processing systems, 1999: 1057-1063.

致谢

在xxxx大学四年的学习和生活即将结束，在这四年的时间里，我从一个稚嫩的男孩蜕变为一个成熟并且有社会经验的人。在长理的校园里我认识了许多来自各省的同学朋友，接触了许多优秀的学长学姐和老师，这些人的出现，让我的大学生活能够如此精彩。

在将近4个月的毕业设计过程中，我的指导老师曾道建老师在系统开发和论文上都给了我很大的帮助，我在公司实习期间老师任然不忘时刻督促我、指导我，给我提供了许多对课题有帮助的资料以及开发环境。

在此，我首先要感谢曾道建老师，陪伴我走过大学期间最后阶段。

感谢我的室友这四年来一起在寝室生活的中能够包容我的缺点和坏习惯，在我需要帮助的时候他们总是最先站出来，感谢实习期间公司的导师和同事们对我的照顾，让我从工作中获得了许多经验。

最后，感谢所有同学和学院老师，没有你们我就无法完成本次系统的开发。在进入社会后，我会继续发扬长理人的精神，为我国互联网事业付诸自己的努力！

附录:程序主要源代码

```
#fields.py
import logging
import torchtext
class SourceField(torchtext.data.Field):
def __init__(self, **kwargs):
logger = logging.getLogger(__name__)
if kwargs.get('batch_first') is False:
logger.warning("Option batch_first has to be set to use pytorch-seq2seq. Changed to True.")
kwargs['batch_first'] = True
if kwargs.get('include_lengths') is False:
logger.warning("Option include_lengths has to be set to use pytorch-seq2seq. Changed to True.")
kwargs['include_lengths'] = True
super(SourceField, self).__init__(**kwargs)
class TargetField(torchtext.data.Field):
""" Wrapper class of torchtext.data.Field that forces batch_first to be True and prepend <sos> and append <eos> to sequences in preprocessing step.
Attributes:
sos_id: index of the start of sentence symbol
eos_id: index of the end of sentence symbol
"""
SYM_SOS = '<sos>'
SYM_EOS = '<eos>'
def __init__(self, **kwargs):
logger = logging.getLogger(__name__)
if kwargs.get('batch_first') == False:
logger.warning("Option batch_first has to be set to use pytorch-seq2seq. Changed to True.")
kwargs['batch_first'] = True
if kwargs.get('preprocessing') is None:
kwargs['preprocessing'] = lambda seq: [self.SYM_SOS] + seq + [self.SYM_EOS]
else:
func = kwargs['preprocessing']
kwargs['preprocessing'] = lambda seq: [self.SYM_SOS] + func(seq) + [self.SYM_EOS]
self.sos_id = None
self.eos_id = None
super(TargetField, self).__init__(**kwargs)
def build_vocab(self, *args, **kwargs):
super(TargetField, self).build_vocab(*args, **kwargs)
self.sos_id = self.vocab.stoi[self.SYM_SOS]
self.eos_id = self.vocab.stoi[self.SYM_EOS]
# evaluator.py
from __future__ import print_function, division
import torch
import torchtext
import seq2seq
from seq2seq.loss import NLLLoss
class Evaluator(object):
def __init__(self, loss=NLLLoss(), batch_size=64):
self.loss = loss
self.batch_size = batch_size
```

```
def evaluate(self, model, data):
model (seq2seq.models): model to evaluate
data (seq2seq.dataset.dataset.Dataset): dataset to evaluate against
model.eval()
loss = self.loss
loss.reset()
match = 0
total = 0
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
batch_iterator = torchtext.data.BucketIterator(
dataset=data, batch_size=self.batch_size,
sort=True, sort_key=lambda x: len(x.src),
device=device, train=False)
tgt_vocab = data.fields[seq2seq.tgt_field_name].vocab
pad = tgt_vocab.stoi[data.fields[seq2seq.tgt_field_name].pad_token]
with torch.no_grad():
for batch in batch_iterator:
input_variables, input_lengths = getattr(batch, seq2seq.src_field_name)
target_variables = getattr(batch, seq2seq.tgt_field_name)
decoder_outputs, decoder_hidden, other = model(input_variables, input_lengths.tolist(), target_variables)
seqlist = other['sequence']
for step, step_output in enumerate(decoder_outputs):
target = target_variables[:, step + 1]
loss.eval_batch(step_output.view(target_variables.size(0), -1), target)
non_padding = target.ne(pad)
correct=seqlist[step].view(-1).eq(target).masked_select(non_padding).sum().item()
match += correct
total += non_padding.sum().item()
if total == 0:
accuracy = float('nan')
else:
accuracy = match / total
return loss.get_loss(), accuracy
import math
import torch.nn as nn
import numpy as np
class Loss(object):
""" Base class for encapsulation of the loss functions.
This class defines interfaces that are commonly used with loss functions
in training and inferencing. For information regarding individual loss
functions, please refer to http://pytorch.org/docs/master/nn.html#loss-functions
Note:
Do not use this class directly, use one of the sub classes.
Args:
name (str): name of the loss function used by logging messages.
criterion (torch.nn._Loss): one of PyTorch's loss function. Refer
to http://pytorch.org/docs/master/nn.html#loss-functions for
a list of them.
Attributes:
name (str): name of the loss function used by logging messages.
criterion (torch.nn._Loss): one of PyTorch's loss function. Refer
to http://pytorch.org/docs/master/nn.html#loss-functions for
a list of them. Implementation depends on individual
sub-classes.
acc_loss (int or torcn.nn.Tensor): variable that stores accumulated loss.
norm_term (float): normalization term that can be used to calculate
the loss of multiple batches. Implementation depends on individual
sub-classes.
def __init__(self, name, criterion):
self.name = name
self.criterion = criterion
```

```
if not issubclass(type(self.criterion), nn.modules.loss._Loss):
raise ValueError("Criterion has to be a subclass of torch.nn._Loss")
self.acc_loss = 0
self.norm_term = 0
def reset(self):
""" Reset the accumulated loss. """
self.acc_loss = 0
self.norm_term = 0
def get_loss(self):
raise NotImplementedError
def eval_batch(self, outputs, target):
raise NotImplementedError
def cuda(self):
self.criterion.cuda()
def backward(self):
if type(self.acc_loss) is int:
raise ValueError("No loss to back propagate.")
self.acc_loss.backward()
class NLLLoss(Loss):
_NAME = "Avg NLLLoss"
def __init__(self, weight=None, mask=None, size_average=True):
self.mask = mask
self.size_average = size_average
if mask is not None:
if weight is None:
raise ValueError("Must provide weight with a mask.")
weight[mask] = 0
super(NLLLoss, self).__init__(
self._NAME,
nn.NLLLoss(weight=weight, size_average=size_average))
def get_loss(self):
if isinstance(self.acc_loss, int):
return 0
# total loss for all batches
loss = self.acc_loss.data.item()
if self.size_average:
# average loss per batch
loss /= self.norm_term
return loss
def eval_batch(self, outputs, target):
self.acc_loss += self.criterion(outputs, target)
self.norm_term += 1
class Perplexity(NLLLoss):
_NAME = "Perplexity"
_MAX_EXP = 100
def __init__(self, weight=None, mask=None):
super(Perplexity, self).__init__(weight=weight, mask=mask, size_average=False)
def eval_batch(self, outputs, target):
self.acc_loss += self.criterion(outputs, target)
if self.mask is None:
self.norm_term += np.prod(target.size())
else:
self.norm_term += target.data.ne(self.mask).sum()
def get_loss(self):
nll = super(Perplexity, self).get_loss()
nll /= self.norm_term.item()
if nll > Perplexity._MAX_EXP:
print("WARNING: Loss exceeded maximum value, capping to e^100")
return math.exp(Perplexity._MAX_EXP)
return math.exp(nll)
#EncoderRNN
```

```
import torch.nn as nn
from .baseRNN import BaseRNN
class EncoderRNN(BaseRNN):
def __init__(self, vocab_size, max_len, hidden_size,
input_dropout_p=0, dropout_p=0,
n_layers=1, bidirectional=False, rnn_cell='gru', variable_lengths=False,
embedding=None, update_embedding=True):
super(EncoderRNN, self).__init__(vocab_size, max_len, hidden_size,
input_dropout_p, dropout_p, n_layers, rnn_cell)
self.variable_lengths = variable_lengths
self.embedding = nn.Embedding(vocab_size, hidden_size)
if embedding is not None:
self.embedding.weight = nn.Parameter(embedding)
self.embedding.weight.requires_grad = update_embedding
self.rnn = self.rnn_cell(hidden_size, hidden_size, n_layers,
batch_first=True, bidirectional=bidirectional, dropout=dropout_p)
def forward(self, input_var, input_lengths=None):
embedded = self.embedding(input_var)
embedded = self.input_dropout(embedded)
if self.variable_lengths:
embedded = nn.utils.rnn.pack_padded_sequence(embedded, input_lengths, batch_first=True)
output, hidden = self.rnn(embedded)
if self.variable_lengths:
output, _ = nn.utils.rnn.pad_packed_sequence(output, batch_first=True)
return output, hidden
#DecoderRNN
import random
import numpy as np
import torch
import torch.nn as nn
from torch.autograd import Variable
import torch.nn.functional as F
from .attention import Attention
from .baseRNN import BaseRNN
if torch.cuda.is_available():
import torch.cuda as device
else:
import torch as device
class DecoderRNN(BaseRNN):
KEY_ATTN_SCORE = 'attention_score'
KEY_LENGTH = 'length'
KEY_SEQUENCE = 'sequence'
def __init__(self, vocab_size, max_len, hidden_size,
sos_id, eos_id,
n_layers=1, rnn_cell='gru', bidirectional=False,
input_dropout_p=0, dropout_p=0, use_attention=False):
super(DecoderRNN, self).__init__(vocab_size, max_len, hidden_size,
input_dropout_p, dropout_p,
n_layers, rnn_cell)
self.bidirectional_encoder = bidirectional
self.rnn = self.rnn_cell(hidden_size, hidden_size, n_layers, batch_first=True, dropout=dropout_p)
self.output_size = vocab_size
self.max_length = max_len
self.use_attention = use_attention
self.eos_id = eos_id
self.sos_id = sos_id
self.init_input = None
self.embedding = nn.Embedding(self.output_size, self.hidden_size)
if use_attention:
self.attention = Attention(self.hidden_size)
self.out = nn.Linear(self.hidden_size, self.output_size)
```

```
def forward_step(self, input_var, hidden, encoder_outputs, function):
batch_size = input_var.size(0)
output_size = input_var.size(1)
embedded = self.embedding(input_var)
embedded = self.input_dropout(embedded)
output, hidden = self.rnn(embedded, hidden)
attn = None
if self.use_attention:
output, attn = self.attention(output, encoder_outputs)
predicted_softmax=function(self.out(output.contiguous().view(-1, self.hidden_size)), dim=1).view(batch_size,
output_size, -1)
return predicted_softmax, hidden, attn
def forward(self, inputs=None, encoder_hidden=None, encoder_outputs=None,
function=F.log_softmax, teacher_forcing_ratio=0):
ret_dict = dict()
if self.use_attention:
ret_dict[DecoderRNN.KEY_ATTN_SCORE] = list()
inputs, batch_size, max_length = self._validate_args(inputs, encoder_hidden, encoder_outputs,
function, teacher_forcing_ratio)
decoder_hidden = self._init_state(encoder_hidden)
use_teacher_forcing = True if random.random() < teacher_forcing_ratio else False
decoder_outputs = []
sequence_symbols = []
lengths = np.array([max_length] * batch_size)
def decode(step, step_output, step_attn):
decoder_outputs.append(step_output)
if self.use_attention:
ret_dict[DecoderRNN.KEY_ATTN_SCORE].append(step_attn)
symbols = decoder_outputs[-1].topk(1)[1]
sequence_symbols.append(symbols)
eos_batches = symbols.data.eq(self.eos_id)
if eos_batches.dim() > 0:
eos_batches = eos_batches.cpu().view(-1).numpy()
update_idx = ((lengths > step) & eos_batches) != 0
lengths[update_idx] = len(sequence_symbols)
return symbols
if use_teacher_forcing:
decoder_input = inputs[:, :-1]
decoder_output, decoder_hidden, attn = self.forward_step(decoder_input, decoder_hidden,encoder_outputs,
function=function)
for di in range(decoder_output.size(1)):
step_output = decoder_output[:, di, :]
if attn is not None:
step_attn = attn[:, di, :]
else:
step_attn = None
decode(di, step_output, step_attn)
else:
decoder_input = inputs[:, 0].unsqueeze(1)
for di in range(max_length):
decoder_output, decoder_hidden, step_attn = self.forward_step(decoder_input, decoder_hidden, encoder_outputs,
function=function)
step_output = decoder_output.squeeze(1)
symbols = decode(di, step_output, step_attn)
decoder_input = symbols
ret_dict[DecoderRNN.KEY_SEQUENCE] = sequence_symbols
ret_dict[DecoderRNN.KEY_LENGTH] = lengths.tolist()
return decoder_outputs, decoder_hidden, ret_dict
def _init_state(self, encoder_hidden):
""" Initialize the encoder hidden state. """
if encoder_hidden is None:
```

```
return None
if isinstance(encoder_hidden, tuple):
encoder_hidden = tuple([self._cat_directions(h) for h in encoder_hidden])
else:
encoder_hidden = self._cat_directions(encoder_hidden)
return encoder_hidden
def _cat_directions(self, h):
if self.bidirectional_encoder:
h = torch.cat([h[0:h.size(0):2], h[1:h.size(0):2]], 2)
return h
def _validate_args(self, inputs, encoder_hidden, encoder_outputs, function, teacher_forcing_ratio):
if self.use_attention:
if encoder_outputs is None:
raise ValueError("Argument encoder_outputs cannot be None when attention is used.")
# inference batch size
if inputs is None and encoder_hidden is None:
batch_size = 1
else:
if inputs is not None:
batch_size = inputs.size(0)
else:
if self.rnn_cell is nn.LSTM:
batch_size = encoder_hidden[0].size(1)
elif self.rnn_cell is nn.GRU:
batch_size = encoder_hidden.size(1)
# set default input and max decoding length
if inputs is None:
if teacher_forcing_ratio > 0:
raise ValueError("Teacher forcing has to be disabled (set 0) when no inputs is provided.")
inputs = torch.LongTensor([self.sos_id] * batch_size).view(batch_size, 1)
if torch.cuda.is_available():
inputs = inputs.cuda()
max_length = self.max_length
else:
max_length = inputs.size(1) - 1 # minus the start of sequence symbol
return inputs, batch_size, max_length
#attention.py
import torch
import torch.nn as nn
import torch.nn.functional as F
class Attention(nn.Module):
r"""
Applies an attention mechanism on the output features from the decoder.
.. math::
\begin{array}{ll}
x = context*output \\
attn = exp(x_i) / sum_j exp(x_j) \\
output = \tanh(w * (attn * context) + b * output)
\end{array}
Args:
dim(int): The number of expected features in the output
Inputs: output, context
- **output** (batch, output_len, dimensions): tensor containing the output features from the decoder.
- **context** (batch, input_len, dimensions): tensor containing features of the encoded input sequence.
Outputs: output, attn
- **output** (batch, output_len, dimensions): tensor containing the attended output features from the decoder.
- **attn** (batch, output_len, input_len): tensor containing attention weights.
Attributes:
linear_out (torch.nn.Linear): applies a linear transformation to the incoming data: :math:`y = Ax + b`.
mask (torch.Tensor, optional): applies a :math:`-inf` to the indices specified in the `Tensor`.
Examples::
```

```
>>> attention = seq2seq.models.Attention(256)
>>> context = Variable(torch.randn(5, 3, 256))
>>> output = Variable(torch.randn(5, 5, 256))
>>> output, attn = attention(output, context)
"""
def __init__(self, dim):
super(Attention, self).__init__()
self.linear_out = nn.Linear(dim*2, dim)
self.mask = None
def set_mask(self, mask):
"""
Sets indices to be masked
Args:
mask (torch.Tensor): tensor containing indices to be masked
"""
self.mask = mask
def forward(self, output, context):
batch_size = output.size(0)
hidden_size = output.size(2)
input_size = context.size(1)
# (batch, out_len, dim) * (batch, in_len, dim) -> (batch, out_len, in_len)
attn = torch.bmm(output, context.transpose(1, 2))
if self.mask is not None:
attn.data.masked_fill_(self.mask, -float('inf'))
attn = F.softmax(attn.view(-1, input_size), dim=1).view(batch_size, -1, input_size)
# (batch, out_len, in_len) * (batch, in_len, dim) -> (batch, out_len, dim)
mix = torch.bmm(attn, context)
# concat -> (batch, out_len, 2*dim)
combined = torch.cat((mix, output), dim=2)
# output -> (batch, out_len, dim)
output = F.tanh(self.linear_out(combined.view(-1, 2 * hidden_size))).view(batch_size, -1, hidden_size)
return output, attn
#seq2seq.py
import torch.nn as nn
import torch.nn.functional as F
class Seq2seq(nn.Module):
""" Standard sequence-to-sequence architecture with configurable encoder
and decoder.
Args:
encoder (EncoderRNN): object of EncoderRNN
decoder (DecoderRNN): object of DecoderRNN
decode_function (func, optional): function to generate symbols from output hidden states (default: F.log_softmax)
Inputs: input_variable, input_lengths, target_variable, teacher_forcing_ratio
- **input_variable** (list, option): list of sequences, whose length is the batch size and within which
each sequence is a list of token IDs. This information is forwarded to the encoder.
- **input_lengths** (list of int, optional): A list that contains the lengths of sequences
in the mini-batch, it must be provided when using variable length RNN (default: `None`)
- **target_variable** (list, optional): list of sequences, whose length is the batch size and within which
each sequence is a list of token IDs. This information is forwarded to the decoder.
- **teacher_forcing_ratio** (int,
```


optional): The probability that teacher forcing will be used. A random number
is drawn uniformly from 0-1 for every decoding token, and if the sample is smaller than the given value,

teacher forcing would be used (default is 0)
Outputs: decoder_outputs, decoder_hidden, ret_dict
- **decoder_outputs** (batch): batch-length list of tensors with size (max_length, hidden_size) containing the outputs of the decoder.
- **decoder_hidden** (num_layers * num_directions, batch, hidden_size): tensor containing the last hidden state of the decoder.
- **ret_dict**: dictionary containing additional information as follows {*KEY_LENGTH* : list of integers representing lengths of output sequences, *KEY_SEQUENCE* : list of sequences, where each sequence is a list of predicted token IDs, *KEY_INPUT* : target outputs if provided for decoding, *KEY_ATTN_SCORE* : list of sequences, where each list is of attention weights }.
"""
def __init__(self, encoder, decoder, decode_function=F.log_softmax):
super(Seq2seq, self).__init__()
self.encoder = encoder
self.decoder = decoder
self.decode_function = decode_function
def flatten_parameters(self):
self.encoder.rnn.flatten_parameters()
self.decoder.rnn.flatten_parameters()
def forward(self, input_variable, input_lengths=None, target_variable=None,
teacher_forcing_ratio=0):
encoder_outputs, encoder_hidden = self.encoder(input_variable, input_lengths)
result = self.decoder(inputs=target_variable,
encoder_hidden=encoder_hidden,
encoder_outputs=encoder_outputs,
function=self.decode_function,
teacher_forcing_ratio=teacher_forcing_ratio)
return result
from __future__ import division
import logging
import os
import random
import time
import torch
import torchtext
from torch import optim
import seq2seq
from seq2seq.evaluator import Evaluator
from seq2seq.loss import NLLLoss
from seq2seq.optim import Optimizer
from seq2seq.util.checkpoint import Checkpoint
class SupervisedTrainer(object):
""" The SupervisedTrainer class helps in setting up a training framework in a
supervised setting.
Args:
expt_dir (optional, str): experiment Directory to store details of the experiment,
by default it makes a folder in the current directory to store the details (default: `experiment`).
loss (seq2seq.loss.loss.Loss, optional): loss for training, (default: seq2seq.loss.NLLLoss)
batch_size (int, optional): batch size for experiment, (default: 64)
checkpoint_every (int, optional): number of batches to checkpoint after, (default: 100)
"""
def __init__(self, expt_dir='experiment', loss=NLLLoss(), batch_size=64,
random_seed=None,
checkpoint_every=100, print_every=100):
self._trainer = "Simple Trainer"
self.random_seed = random_seed
if random_seed is not None:
random.seed(random_seed)
torch.manual_seed(random_seed)
self.loss = loss
self.evaluator = Evaluator(loss=self.loss, batch_size=batch_size)

```
self.optimizer = None
self.checkpoint_every = checkpoint_every
self.print_every = print_every
if not os.path.isabs(expt_dir):
expt_dir = os.path.join(os.getcwd(), expt_dir)
self.expt_dir = expt_dir
if not os.path.exists(self.expt_dir):
os.makedirs(self.expt_dir)
self.batch_size = batch_size
self.logger = logging.getLogger(__name__)
def _train_batch(self, input_variable, input_lengths, target_variable, model, teacher_forcing_ratio):
loss = self.loss
# Forward propagation
decoder_outputs, decoder_hidden, other = model(input_variable, input_lengths, target_variable,
teacher_forcing_ratio=teacher_forcing_ratio)
# Get loss
loss.reset()
for step, step_output in enumerate(decoder_outputs):
batch_size = target_variable.size(0)
loss.eval_batch(step_output.contiguous().view(batch_size, -1), target_variable[:, step + 1])
# Backward propagation
model.zero_grad()
loss.backward()
self.optimizer.step()
return loss.get_loss()
def _train_epoches(self, data, model, n_epochs, start_epoch, start_step,
dev_data=None, teacher_forcing_ratio=0):
log = self.logger
print_loss_total = 0 # Reset every print_every
epoch_loss_total = 0 # Reset every epoch
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
batch_iterator = torchtext.data.BucketIterator(
dataset=data, batch_size=self.batch_size,
sort=False, sort_within_batch=True,
sort_key=lambda x: len(x.src),
device=device, repeat=False)
steps_per_epoch = len(batch_iterator)
total_steps = steps_per_epoch * n_epochs
step = start_step
step_elapsed = 0
for epoch in range(start_epoch, n_epochs + 1):
log.debug("Epoch: %d, Step: %d" % (epoch, step))
batch_generator = batch_iterator.__iter__()
# consuming seen batches from previous training
for _ in range((epoch - 1) * steps_per_epoch, step):
next(batch_generator)
model.train(True)
for batch in batch_generator:
step += 1
step_elapsed += 1
input_variables, input_lengths = getattr(batch, seq2seq.src_field_name)
target_variables = getattr(batch, seq2seq.tgt_field_name)
loss = self._train_batch(input_variables, input_lengths.tolist(), target_variables, model, teacher_forcing_ratio)
# Record average loss
print_loss_total += loss
epoch_loss_total += loss
if step % self.print_every == 0 and step_elapsed > self.print_every:
print_loss_avg = print_loss_total / self.print_every
print_loss_total = 0
log_msg = 'Progress: %d%%, Train %s: %.4f' % (
step / total_steps * 100,
```

```
self.loss.name,
print_loss_avg)
log.info(log_msg)
# Checkpoint
if step % self.checkpoint_every == 0 or step == total_steps:
Checkpoint(model=model,
optimizer=self.optimizer,
epoch=epoch, step=step,
input_vocab=data.fields[seq2seq.src_field_name].vocab,
output_vocab=data.fields[seq2seq.tgt_field_name].vocab).save(self.expt_dir)
if step_elapsed == 0: continue
epoch_loss_avg = epoch_loss_total / min(steps_per_epoch, step - start_step)
epoch_loss_total = 0
log_msg = "Finished epoch %d: Train %s: %.4f" % (epoch, self.loss.name, epoch_loss_avg)
if dev_data is not None:
dev_loss, accuracy = self.evaluator.evaluate(model, dev_data)
self.optimizer.update(dev_loss, epoch)
log_msg += ", Dev %s: %.4f, Accuracy: %.4f" % (self.loss.name, dev_loss, accuracy)
model.train(mode=True)
else:
self.optimizer.update(epoch_loss_avg, epoch)
log.info(log_msg)
def train(self, model, data, num_epochs=5,
resume=False, dev_data=None,
optimizer=None, teacher_forcing_ratio=0):
""" Run training for a given model.
Args:
model (seq2seq.models): model to run training on, if `resume=True`, it would be
overwritten by the model loaded from the latest checkpoint.
data (seq2seq.dataset.dataset.Dataset): dataset object to train on
num_epochs (int, optional): number of epochs to run (default 5)
resume(bool, optional): resume training with the latest checkpoint, (default False)
dev_data (seq2seq.dataset.dataset.Dataset, optional): dev Dataset (default None)
optimizer (seq2seq.optim.Optimizer, optional): optimizer for training
(default: Optimizer(pytorch.optim.Adam, max_grad_norm=5))
teacher_forcing_ratio (float, optional): teaching forcing ratio (default 0)
Returns:
model (seq2seq.models): trained model.
"""
# If training is set to resume
if resume:
latest_checkpoint_path = Checkpoint.get_latest_checkpoint(self.expt_dir)
resume_checkpoint = Checkpoint.load(latest_checkpoint_path)
model = resume_checkpoint.model
self.optimizer = resume_checkpoint.optimizer
# A walk around to set optimizing parameters properly
resume_optim = self.optimizer.optimizer
defaults = resume_optim.param_groups[0]
defaults.pop('params', None)
defaults.pop('initial_lr', None)
self.optimizer.optimizer = resume_optim.__class__(model.parameters(), **defaults)
start_epoch = resume_checkpoint.epoch
step = resume_checkpoint.step
else:
start_epoch = 1
step = 0
if optimizer is None:
optimizer = Optimizer(optim.Adam(model.parameters()), max_grad_norm=5)
self.optimizer = optimizer
self.logger.info("Optimizer: %s, Scheduler: %s" % (self.optimizer.optimizer, self.optimizer.scheduler))
self._train_epoches(data, model, num_epochs,
```

```
start_epoch, step, dev_data=dev_data,
teacher_forcing_ratio=teacher_forcing_ratio)
return model
```

说明：1.总文字复制比：被检测论文总重合字数在总字数中所占的比例

2.去除引用文献复制比：去除系统识别为引用的文献后，计算出来的重合字数在总字数中所占的比例

3.去除本人已发表文献复制比：去除作者本人已发表文献后，计算出来的重合字数在总字数中所占的比例

4.单篇最大文字复制比：被检测文献与所有相似文献比对后，重合字数占总字数的比例最大的那一篇文献的文字复制比

5.指标是由系统根据《学术论文不端行为的界定标准》自动生成的

6.红色文字表示文字复制部分;绿色文字表示引用部分;棕灰色文字表示作者本人已发表文献部分

7.本报告单仅对您所选择比对资源范围内检测结果负责

amlc@cnki.net

http://check.cnki.net/

http://e.weibo.com/u/3194559873/