

申请上海交通大学硕士学位论文

SQL 查询语句的自动生成技术研究

论文作者 熊云翔

学 号 116037910048

导 师 沈备军

专 业 软件工程

答辩日期 2018 年 12 月 12 日

Submitted in total fulfillment of the requirements for the degree of Master
in Engineering

Research on Automatic Generation Technology of SQL Query Statement

XIONG YUNXIANG

Advisor

Prof. BEIJUN SHEN

SOFTWARE ENGINEERING

SHANGHAI JIAO TONG UNIVERSITY

SHANGHAI, P.R.CHINA

Dec. 12th, 2018

上海交通大学 学位论文原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的作品成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

学位论文作者签名：_____

日 期：_____年 ____月 ____日

上海交通大学 学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权上海交通大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

本学位论文属于

保 密 ☐，在 _____ 年解密后适用本授权书。

不保密 ☐。

(请在以上方框内打 ☒)

学位论文作者签名： _____

指导教师签名： _____

日 期： _____ 年 ____ 月 ____ 日

日 期： _____ 年 ____ 月 ____ 日

SQL 查询语句的自动生成技术研究

摘 要

待写 关键词：上海交大, 饮水思源, 爱国荣校

RESEARCH ON AUTOMATIC GENERATION TECHNOLOGY OF SQL QUERY STATEMENT

ABSTRACT

write **KEY WORDS:** SJTU, master thesis, XeTeX/LaTeX template

目 录

插图索引	IX
表格索引	XI
算法索引	XIII
主要符号对照表	XV
第一章 绪论	1
1.1 研究背景	1
1.2 研究目标和研究内容	2
1.3 论文结构	3
第二章 基于映射的 INL2SQL 生成	5
2.1 研究问题	5
2.2 相关技术	6
2.3 解决方案	7
2.3.1 依赖解析树生成	8
2.3.2 解析树节点映射	8
2.3.3 解析树优化重构	9
2.3.4 查询树翻译	12
2.4 实验与分析	13
2.4.1 数据集	13
2.4.2 实验结果	14
2.4.3 实验分析	14
2.5 本章小结	16
第三章 基于深度强化学习的 NL2SQL 生成	17
3.1 研究问题	17
3.2 相关技术	18
3.2.1 NL2SQL 研究现状	18

3.2.2	深度学习	18
3.2.3	强化学习	18
3.2.4	语义解析	18
3.3	解决方案	18
3.3.1	增强解析器模型	18
3.3.2	动作序列	19
3.3.3	编码器	20
3.3.4	解码器	21
3.3.5	过滤条件顺序问题和隐式列名问题	22
3.4	实验与分析	24
3.4.1	数据集及评价指标	24
3.4.2	实验设置	25
3.4.3	实验结果	25
3.5	本章小结	30
第四章	基于多任务学习的 NL2SQL 生成	31
4.1	研究问题	31
4.2	相关技术	32
4.2.1	迁移学习与多任务学习	32
4.2.2	元学习	32
4.3	解决方案	32
4.3.1	TCR 模板	32
4.3.2	多任务网络	34
4.3.3	编码器	35
4.3.4	解码器	37
4.4	实验与分析	39
4.4.1	多任务网络的实验结果	39
4.4.2	不同优化策略下的实验结果	41
4.4.3	实验分析	42
4.5	本章小结	43
第五章	总结与展望	45
5.1	本文工作小结	45
5.2	展望	45

参考文献	47
致 谢	49
攻读学位期间发表的学术论文	51
攻读学位期间参与的项目	53
简 历	55

插图索引

2-1 这里将出现在插图索引中	5
2-2 这里将出现在插图索引中	7
2-3 这里将出现在插图索引中	13
3-1 这里将出现在插图索引中	17
3-2 这里将出现在插图索引中	21
3-3 这里将出现在插图索引中	22
3-4 这里将出现在插图索引中	28
3-5 这里将出现在插图索引中	29
4-1 这里将出现在插图索引中	31
4-2 这里将出现在插图索引中	33
4-3 这里将出现在插图索引中	33
4-4 这里将出现在插图索引中	35

表格索引

2-1 指向一个表格的表目录索引	9
2-2 指向一个表格的表目录索引	10
2-3 指向一个表格的表目录索引	14
2-4 指向一个表格的表目录索引	14
2-5 指向一个表格的表目录索引	15
3-1 指向一个表格的表目录索引	19
3-2 指向一个表格的表目录索引	25
3-3 出现在表目录的标题	26
3-4 指向一个表格的表目录索引	27
3-5 出现在表目录的标题	28
3-6 出现在表目录的标题	30
4-1 出现在表目录的标题	39
4-2 出现在表目录的标题	42

算法索引

2-1 解析树结构调整算法	11
-------------------------	----

主要符号对照表

ϵ	介电常数
μ	磁导率
ϵ	介电常数
μ	磁导率
ϵ	介电常数
μ	磁导率
ϵ	介电常数
μ	磁导率
ϵ	介电常数
μ	磁导率
ϵ	介电常数
μ	磁导率
ϵ	介电常数
μ	磁导率
ϵ	介电常数
μ	磁导率
ϵ	介电常数
μ	磁导率
ϵ	介电常数
μ	磁导率
ϵ	介电常数
μ	磁导率
ϵ	介电常数
μ	磁导率
ϵ	介电常数
μ	磁导率
ϵ	介电常数

μ	磁导率
ϵ	介电常数
μ	磁导率
ϵ	介电常数
μ	磁导率
ϵ	介电常数
μ	磁导率
ϵ	介电常数
μ	磁导率
ϵ	介电常数
μ	磁导率
ϵ	介电常数
μ	磁导率
ϵ	介电常数
μ	磁导率
ϵ	介电常数
μ	磁导率
ϵ	介电常数
μ	磁导率
ϵ	介电常数
μ	磁导率
ϵ	介电常数
μ	磁导率

第一章 绪论

1.1 研究背景

随着社会的不断发展，以 IT 和互联网技术为标志的信息产业不断地改变着人类的工作和生活方式。在此背景之下，数据库技术应运而生。它是一种建立在计算机存储设备上的仓库，可以将大量数据按照数据结构来组织、存储和管理。关系数据库中存储了大量的数据和信息。医疗、教育、金融等各个行业都在使用关系型数据库作为数据存储以及应用程序的基础。在软件开发过程中，软件的开发和技术人员会频繁地进行 SQL 语句的创建与查询以及相关数据库的操作。在软件运行时，业务人员也会使用 SQL 语句进行报表与在线分析（OLAP）的定制。

目前，各种计算机系统，尤其是商业领域，应用关系型数据库系统和 SQL 语言来进行数据管理，仍然是最主流和最成熟的方案。需要注意的是，SQL 作为一种数据库操作语言，本质上仍然是一种编程语言，需要操作人员具有一定的专业知识，经过数据库和 SQL 相关知识的培训，才能比较熟练的进行数据库的管理。此外，除了要具备 SQL 和数据库技术的相关知识，具体到真实数据库的操作，数据管理人员还需要对于所使用的关系型数据库的关系模式有所了解，才能将各种操作需求转化为 SQL 语句，来对数据库系统进行管理。然而，随着数据库系统的应用场景越发广泛和复杂，以及数据库数据处理量的不断提升，数据管理人员对数据库的操作逻辑也越来越复杂，数据库查询需求所涉及的数据量也越来越大，相关的关系模式也越来越复杂和多样化。当数据库管理需求达到这样的复杂度，非专业的数据管理人员就越来越无法满足需求。

1.1.0.1 自然语言接口自动生成 SQL 查询语句

自然语言接口（NLI, NaturalLanguageInterfaces），是自然语言处理和人机交互的交叉领域，旨在为人类提供通过自然语言与计算机交互的手段。同时，自然语言接口也是人机交互领域研究的终极目标，从各种对话机器人，到今天各种智能穿戴设备装载的语音助手，人机交互领域和自然语言理解领域的专家们一直在朝建立真正智能的自然语言接口这个目标不断前进。

自然语言接口生成 SQL 也是人们关注的一个领域^[1]，自上个世纪提出以来，人们不断研究从自然语言生成 SQL 语句的可能性，并且的确在研究过程中取得了一些令人振奋的成果。通过自然语言接口生成 SQL 的数据库管理系统，原型已经出现在六十年代和七十年代初期，那时候最著名的自然语言接口数据库是 Lunar，正如其名字，包含月

球岩石和化学数据库的自然语言界面。这个原型的实现，是基于特定数据库的，因此无法很容易地修改为和不同的数据库一起使用。之后出现了其他的自然语言接口数据库，用户可以通过对话系统来定制查询，并且这些系统可以配置不同的接口，供不同的底层数据库调用。这时候的自然语言数据库系统使用语义语法，是一种句法和语义处理的综合技术。之后，还有关注于将自然语言输入转化为逻辑语言的技术，以此技术作为自然语言接口数据库的核心技术。

一直以来，自然语言接口是人机交互领域的终极追求，也是人机交互、机器学习领域专家孜孜不倦钻研学习的热门问题。对于本文所指出的业务层面与技术层面之间的矛盾，如果能对最终用户提供一个自然语言接口，使得我们的系统能直接从最终用户的自然语言中理解到用户的查询意图，并结合数据库，直接生成符合查询意图的 SQL 语句返回给用户，那么这一矛盾就可以较好的得到解决。

1.1.0.2 自然语言自动生成 SQL 查询语句

XXXXXXXXXX

XXXXXXXXXX

XXXXXXX

因此，本文设计和实现一个 SQL 查询语句自动生成工具，为数据库使用者提供简单、便捷的接口，将数据库信息映射到业务需求。用户无需了解 SQL 语句的使用方式，只需关注数据操作需求对应的业务需求，从而弥合业务人员与数据操作之间的矛盾。同时对从自然语言自动生成 SQL 查询语句技术（以下简称 NL2SQL）进行了研究，提出了针对英文自然语言生成 SQL 查询语句的解决方案以及针对中文自然语言生成 SQL 查询语句的解决方案，使得用户可以通过自然语言的表述方式生成 SQL 查询语句并从关系型数据库中找到所需信息，从而缩短业务与技术之间的鸿沟，提高报表与 OLAP 分析的开发效率。

1.2 研究目标和研究内容

本文的研究目标是研究 SQL 查询语句的自动化生成技术，采用解析树映射、语义解析、编码-解码器、注意力机制、深度强化学习和多任务学习等技术，提出基于自然语言接口、英文自然语言和中文自然语言自动生成 SQL 查询语句的技术方案。

本文的研究思路是，先对自然语言进行初步解析和理解，并在其中插入人机交互机制，让用户来引导生成的过程，指导自然语言理解，提出自然语言接口自动生成 SQL 查询语句的方法。然后结合编码-解码器和深度强化学习等技术，对更具难度的纯英文自然语言自动生成 SQL 查询语句技术进行研究。最后使用多任务学习技术将中文-英文翻

译任务和英文自然语言生成 SQL 查询语句技术有机结合，从而提出难度更高的中文自然语言生成 SQL 查询语句方法。

具体研究内容包括：

1. SQL 查询语句自动生成现状。xxxxxxx
2. 基于映射的 NLI2SQL 生成。xxxxxx
3. 基于深度强化学习的 NL2SQL 生成。xxxxx
4. 基于多任务学习的 NL2SQL 生成。xxxx
5. 实验。xxxx

1.3 论文结构

第一章 绪论

从自然语言接口和自然语言自动生成 SQL 查询语句两个方面介绍了课题的研究背景、研究目标和研究内容，对全文做出总览。最后说明了论文的组织结构

第二章 基于映射的 INL2SQL 生成

第三章 基于深度强化学习的 NL2SQL 生成

第四章 基于多任务学习的 NL2SQL 生成

第五章 总结与展望

第二章 基于映射的 INL2SQL 生成

2.1 研究问题

一直以来,自然语言接口是人机交互领域的终极追求,也是人机交互、机器学习领域专家孜孜不倦钻研学习的热门问题。交互式自然语言接口生成 SQL 查询语句(INL2SQL)的关键问题就是怎样使得我们的系统能在与用户的多轮交互之下理解到用户的查询意图,并结合数据库直接生成符合查询意图的 SQL 语句返回给用户。其主要过程为:用户首先输入自然语言查询语句以及给出对应的数据库表模式,系统接收到输入后判断输入的正确性并和用户产生多轮交互,在交互完成并确认输出有效后将对应的 SQL 查询语句及执行结果输出。

输入: 'Return the altitude of Willis Tower in Chicago'

Rank	Name	Location	Height(ft)	Floor	Year
1	One World Trade Center	New York City	1,776	104	2014
2	Willis Tower	Chicago	1,451	108	1974

输出: {'Name'='Willis Tower'; 'Location'='Chicago'}, {'select'=['Height(ft)','Floor','Year']}

输入: {'select'='Height(ft)'}

输出: {'sql'='SELECT 'Height(ft)' WHERE Name='Willis Tower' AND Location='Chicago';
'result'='1,451'}

图 2-1 基于映射的 INL2SQL 生成示例

Figure 2-1 English caption

图2-1中给出了基于映射的 INL2SQL 生成中的典型的例子。用户首先输入自然语言查询语句“Return the altitude of Willis Tower in Chicago”以及给出对应的数据库表模式(包含表、列名等)。系统判断出两个有效过滤条件即‘Name’=‘Willis Tower’和‘Location’=‘Chicago’但不确认聚合函数是否为‘SELECT’及其对象是否为‘Height(ft)’。

系统返回一个包含 ‘Name’ = ‘Willis Tower’ ; ‘Location’ = ‘Chicago’ 和 ‘select’ =[‘Height(ft)’ ,’ Floor’ ,’ Year’] 的交互项。其中 ‘Name’ = ‘Willis Tower’ ; ‘Location’ = ‘Chicago’ 为 json 格式并表示为已经确认的信息, ‘select’ =[‘Height(ft)’ ,’ Floor’ ,’ Year’] 为 json 格式表示需要用户在 ‘Height(ft)’ ,’ Floor’ ,’ Year’ 中做出选择。用户接受到交互信息后进行选择并发送包含 ‘select’ = ‘Height(ft)’ 的 json 数据。最后, 系统在多轮交互并确认输出有效后将对应的 SQL 查询语句 ‘SELECT ‘Height(ft)’ WHERE Name= ‘Willis Tower’ AND Location= ‘Chicago’ 及执行结果 ‘1,451’ 输出。

2.2 相关技术

在软件产品的开发与使用过程中, 非专家用户常常会在指定的关系型数据库上使用关键字搜索 [!! 引用!!] 来进行实时的查询。所以近段时间以来, 有很多的研究人员对关键字搜索领域进行研究 [!! 引用!!]。关键字搜索的目的不是通过输入的关键字的集合来找到与这些关键字相关联的数据, 而是通过这些关键字来推测并解析出其背后想要表达的查询意图及对应的自然语言。在这些研究中, 有的方法通过聚合函数 [!! 引用!!]、布尔运算符 [!! 引用!!]、查询语句片段 [!! 引用!!] 等方式对关键字进行拓展。现在看来, 这些研究内容是解决自然语言查询这一挑战的奠基者。本章所提出的解决方案能够支持更加丰富的查询, 使得用户可以使用语义更加复杂的语句进行查询。

数据库的自然语言接口 (NLIDB) 已经被研究人员研究了很长的时间 [!! 引用!!]。早期的 NLIDB 需要依靠为每个数据库单独定制的手工语法来进行查询, 这些手工语法的拓展性很差, 很难在其他数据库中进行使用。相比之下, 本章所提出的解决方案以生成 SQL 查询语句作为目的, 具有很好的通用性, 可以被移植到其他的数据库上使用。

如 [!! 引用!!] 所指出, NLIDB 在使用上最大的问题在于其所依赖的信息很有限, 正常的自然语言往往是没有一个固定的格式和语义。这也就导致了 NLIDB 无法准确地推断用户输入的查询意图。为了解决可靠性的问题, [!! 引用!!] 等人采用了 PRECISE 方法, 将输入的自然语言查询划分为语义更加容易被解析的查询片段, 并将这些查询片段更为精确地转换为 SQL 查询语句。但是, PRECISE 无法处理那些语义不确切或者难以处理的自然语言查询。相比而言, 本章提出的解决方案通过与用户的多轮交互, 可以较好地解决语义不明确的问题。

也有很多文献 [!! 引用!!] 提出过交互式 NLIDB 的想法, 早期交互式 NLIDB [!! 引用!!] 主要目标在于从查询生成对应的结果。而 NaLIX [!! 引用!!] 则更进一步, 当用户的查询超出模型的语义理解能力的范围之外时会把相应的重构查询的建议返回给用户, 这种策略会大大减少用户在使用时对其查询进行重构的负担。然而, NaLIX 对输入的查询超过模型的语义理解能力的处理方式不太正确。而在本章提出的解决方案中,

对于给定的自然语言查询语句，我们会与用户多轮交互并向用户解释我们解析的过程以及生成 SQL 的过程，从而对查询进行重构，解决歧义问题。所以，在与用户的多轮反馈与交互下，用户在使用我们的系统时可以放心地输入较为复杂的查询，其生成的 SQL 查询语句可以包含多级聚合操作、嵌套以及各种类型的连接操作等。

总而言之，本章提出的解决方案具有较好的通用性，能够解决语义不明确的问题，可以通过与用户的多轮交互向用户解释我们解析的过程以及生成 SQL 的过程从而对查询进行重构。

2.3 解决方案

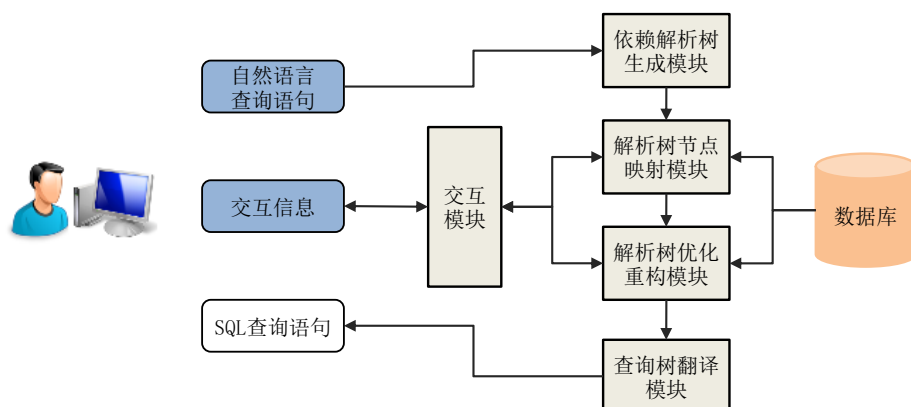


图 2-2 基于映射的 NLI2SQL 生成的总体方案

Figure 2-2 English caption

图2-2是本文从自然语言生成 SQL 语句模型的总体方案，它包含依赖解析树生成、解析树节点映射、解析树优化重构、查询树翻译、交互式对话器、用户接口六个模块：

1. 用户接口：用户与系统进行交互的接口，包括输入自然语言、返回 SQL 语句、解析过程中的交互等等。
2. 交互式对话器：管理解析过程与用户的交互，在适当的时候与用户进行交互，让用户对解析过程进行指导。
3. 依赖解析树生成：将用户输入的，以自然语言表示的查询意图，应用自然语言理

- 解技术，转化为依赖解析树，即将词语进行词性标注，以及识别出词语之间的关系，并将其组织成一个树状结构。详见2.3.1节。
4. 解析树节点映射：根据解析树节点对应的词语和数据库元数据、数据、SQL 语法等信息，将解析树节点映射为 SQL 语法组件。在这个过程中，如果节点的映射有多个候选答案，交互式对话器会发起与用户的交互，将多个候选映射展示给用户，让用户来进行选择。详见2.3.2节。
 5. 解析树优化重构：节点映射完毕后，解析树节点经过系统匹配和用户指导后，得到了比较准确的结果，但解析树的结构仍是最初由依赖解析树生成得到的结构，这一原始结构的准确度并不高，受限于自然语言的复杂性和省略性，可能会有错误关系、缺失关系、缺失节点等等。为了使解析树能得到比较准确的结构，将设计算法和规则，修正错误关系，补全缺失节点和缺失关系，得到较为准确的解析树，称为查询树。详见2.3.3节。
 6. 查询树翻译：解析树的结构已经符合 SQL 语法，节点映射结果也对应于真实的 SQL 语法、数据库 schema 或数据，可以很自然地将树状结构的翻译为 SQL 语句，最后将 SQL 语句通过用户接口返回给用户。详见2.3.4节。

接下来，将详细阐述依赖解析树模块、解析树节点映射模块、解析树优化重构模块、查询树翻译器的设计思路和实现细节。

2.3.1 依赖解析树生成

本模块将用户输入的自然语言查询意图解析为解析树，包含各词语的词性标注、关系提取等等信息。在具体实现过程中，这一模块基于 StanfordCoreNLP[!!!!!!! 此处引用] 实现。StanfordCoreNLP 是斯坦福大学推出的自然语言处理工具集，支持多种语言，还提供了 C++、Python、Java 等多种程序语言的编程接口，提供依赖解析、命名实体识别、词性标注、情感分析、机器翻译等多种功能。本模块调用该工具，将自然语言解析为依赖解析树。

2.3.2 解析树节点映射

本模块将依赖解析树的节点对应的词语，映射到 SQL 组件上。解析树节点的类型定义如表2-1所示。

其中名字节点和值节点是与当前应用的业务数据库有关，其余五种节点都与业务数据库无关，仅与 SQL 语法规则相关。所以，本系统建立了一个五种与业务数据库无关的节点类型与自然语言单词的词典映射。映射过程的实现如下：对每一个解析树节点对应的单词 n ，分别计算其与业务数据库元数据、存储数据、词典映射中词语 v 的相似度

$Sim_{wup}(n, v)$, $Sim_{gram}(n, v)$ 的定义如公式2-1所示:

$$Sim(n, v) = \max(Sim_{wup}(n, v), Sim_{gram}(n, v)) \quad (2-1)$$

其中 $Sim_{wup}(n, v)$ 为 n 和 v 的 WUP 相似度 [!! 引用!!], $Sim_{gram}(n, v)$ 为 n 和 v 的 q-gram 的 Jaccard 相似度的平方根 [!! 引用!!]。经公式计算, 可以得到节点单词 n 与所有 SQL 组件的相似度; 对相似度进行排序, 可以得到前五相似的 SQL 组件, 如果前五相似的 SQL 组件的相似度 $Sim(n, v)$ 的值差别较大, 则直接以相似度最高的 SQL 组件作为当前单词 n 的映射, 并赋予该组件对应的节点类型; 若前五相似的 SQL 组件的相似度差别较小, 则视作歧义, 将候选的 SQL 组件返回给用户, 让用户来进行选择, 最后用户选择的结果会作为当前节点的映射。

2.3.3 解析树优化重构

节点映射完成后, 需要对解析树的结构进行重构, 保证解析树的结构能有较高的准确性。由于解析树可能会存在关系解析错误和节点关系缺失, 所以这一模块对于解析树的优化重构会分为两个步骤进行, 分别为结构调整和隐藏节点插入。

2.3.3.1 结构调整

在进行结构调整之前, 首先定义什么样的树结构是好的、合法的。我们从两个角度来考虑这个问题: 第一点是树结构与经依赖解析器解析后的原始结构的差别有多大; 第

表 2-1 解析树节点的类型

Table 2-1 A Table

节点类型	对应的 SQL 组件
选择节点 (SN)	SELECT
操作符节点 (ON)	一个操作符, 如等于 (“=”)、小于 (“<”)
聚合函数节点 (FN)	一个聚合函数, 如 AVG、MAX
名字节点 (NN)	业务数据库中的一个数据表的名字, 或数据表的一个字段的名字
值节点 (VN)	业务数据库中某字段的一个值
度量节点 (QN)	ALL, ANY, EACH
逻辑节点 (LN)	AND, OR, NOT

二是树结构是否符合 SQL 的语法，这一点的评估可以根据表2-2的定义来确定 [!!yinyong!!]。

表 2-2 合法解析树结构规则

Table 2-2 A Table

1	$Q \rightarrow (SClause) (ComplexCondition) *$
2	$SClause \rightarrow SELECT + GNP$
3	$ComplexCondition \rightarrow ON + (leftSubtree * rightSubtree)$
4	$leftSubtree \rightarrow GNP$
5	$rightSubtree \rightarrow GNP \mid VN \mid MIN \mid MAX$
6	$GNP \rightarrow (FN + GNP) \mid NP$
7	$NP \rightarrow NN + (NN) * (Condition) *$
8	$condition \rightarrow VN \mid (ON + VN)$

表2-2根据 SQL 语句的语法，结合了树状结构，定义了能合理的转化为 SQL 语句的语法树应该满足什么样的规则，这样的语法树我们称之为查询树。在表2-2中，“+”代表父子节点的关系，“*”代表兄弟节点的关系，上标“*”代表可重复出现的兄弟节点，“|”代表“或者关系”，表示当前节点可能存在的情况。算法2-1展示了结构调整的算法，算法的基本思想是，建立一个优先级队列，对于当前的解析树，调用 adjust() 函数（第 8 行），通过一次移动子树操作，生成在这一次操作后所有可能的结构；然后记录当前树的哈希值（第 12 行），防止之后出现重复的树结构；若当前树结构没有出现过（第 10 行），且 edit 值小于一个阈值，则对此树进行下一步操作；由于移动了一次子树，返回的树的 edit 属性加一（第 11 行），这一属性将用来评估生成的树结构与原始结构的差异；调用 evaluate() 函数，记录当前结构有多少节点不满足表2-2设定的规则，不满足规则的节点数将用来评估树结构在语法上的合法性；综合这两方面评估标准，为树打分，如果分数比之前的树结构要高，则加入优先级队列；若该树完全符合语法规则，则视为一颗查询树，加入 result 集合；之后对优先级队列内的树结构重复以上操作，直到优先级队列为空；最后根据评估分数对 result 集合排序，将结果返回。

结构调整之后的解析树结果集，将会通过交互式对话器，与用户进行交互，因为结果集中的解析树虽然都符合 SQL 语法规则，但仍有可能存在与用户意图不同的情况，如 SELECT 子句中的名字节点和 WHERE 子句中的名字节点，位置可能会互换，虽然仍然符合 SQL 语法规则，但与原始查询意图已经有比较大的差别了。所以，在这里需要再一次应用人机交互机制，让用户来选择结果集中与自己的查询意图比较相似的解析树。

算法 2-1 解析树结构调整算法

输出: 结构调整之后的解析树结果集

```

1: results  $\leftarrow$  empty_set()
2: priorityQueue  $\leftarrow$  empty_priority_queue()
3: priorityQueue  $\leftarrow$  priorityQueue + parseTree
4: hash_table  $\leftarrow$  empty_set()
5: hash_table  $\leftarrow$  hash_table + hash(parseTree)
6: while priorityQueue  $\neq$  empty_priority_queue() do
7:   tree  $\leftarrow$  priorityQueue.pop()
8:   treeList  $\leftarrow$  adjust(tree)
9:   for adjustTree in treeList do
10:    if hash(adjustTree) notin hash_table and adjustTree.edit < t then
11:      adjustTree.edit  $\leftarrow$  tree.edit + 1
12:      hash_table  $\leftarrow$  hash_table + hash(adjustTree)
13:      if evaluate(adjustTree)  $\geq$  evaluate(tree) then
14:        priorityQueue.add(tree)
15:      end if
16:      if adjustTree is valid then
17:        results.add(adjustTree)
18:      end if
19:    end if
20:  end for
21: end while
22: results  $\leftarrow$  rank(results)
23: return results

```

交互完成后，筛选出的结果集，会进入下一步——隐藏节点插入。

2.3.3.2 隐藏节点插入

结构调整完成后，对经过排序和用户交互筛选后的结果集合，进行隐藏节点插入。在给出隐藏节点插入的方法之前，先给出需要用到的概念定义，即“核心节点”。核心节点指的是在节点类型为 `leftSubtree` 和 `rightSubtree` 的情况下（表2-2），`leftSubtree`（`rightSubtree`）的所有子节点中，高度最高的名称节点被称为核心节点。

经过研究，需要进行隐藏节点插入的情况有以下几种：

1. 左子树（`leftSubtree`）与右子树（`rightSubtree`）的核心节点对应了不同的 SQL 组件，即认为右子树真正的核心节点在自然语言表达时被省略了 [14]；这是十分常见的现象，因为人在进行自然语言表达时，对两个值进行比较时，会很自然的省略掉后者的一部分，如 “Ihavemorebooksthan yours” 这句话，就将最后的 “yourbooks” 给省略成了 “yours”；
2. 左右子树的约束条件应该一致，如果不一致，则认为右子树一部分约束条件被省略了，如 “returnauthorswhosepaperspublishedin2018morethanJack’s” 这句话，过滤条件的左子树有 “in2018” 这一约束，而右子树在解析之后没有这一约束，事实上右子树的这一约束被隐藏了，需要作为隐藏节点插入进去；
3. 某些函数会被省略，如聚合函数 “COUNT”，在自然语言表达中经常会被省略，如 “Ihavemorebooksthan yours” 这句话，“thenumberof” 就被省略了。在树结构中，如果过滤条件的操作符为 “more”、“less” 等词语，而左右子树的核心节点对应的是非数字类型的 SQL 组件，那么就认为 “COUNT” 被省略了，需要作为隐藏节点插入解析树。

进行隐藏节点插入之后，查询树的结构就比较完整了，将会输入下一模块进行翻译。

2.3.4 查询树翻译

这一阶段的查询树已经在节点映射、树结构、完整性方面都比较可信、合法了，翻译步骤如下：

1. 根据表2-2中的树结构，在 `SClause` 子树下的结构为 `SELECT` 子句，读取 `SClause` 下的名称节点，根据对应的 SQL 组件（如果 SQL 组件对应某数据表，该数据表会预定义一个核心字段，如用户的名字、城市的名称，该数据表的核心字段将作为结果返回），填充入 `SELECT` 子句，并记录 SQL 组件对应的数据表，以备 `FROM` 子句的生成。

2. 在 **ComplexCondition** 子树下的结构为 **WHERE** 子句，分别读取左子树和右子树核心节点对应的 **SQL** 组件，记录下 **SQL** 组件对应的数据表，并查看左右子树中所有的节点，根据其节点类型，将其翻译为对应的 **SQL** 组件，并应用于核心节点；左右子树解析完后，以操作符连接左右子树，将其填充入 **WHERE** 子句。
3. 根据之前记录的相关数据表，生成 **SQL** 语句的 **FROM** 子句。
4. 将三部分按照语法连接起来，作为合法的 **SQL** 语句返回给用户。

2.4 实验与分析

2.4.1 数据集

本次实验所使用的业务数据库为 **MySQL** 的样例数据库 **Classicmodels**，图 3-2 为 **Classicmodels** 数据库的数据库模式图

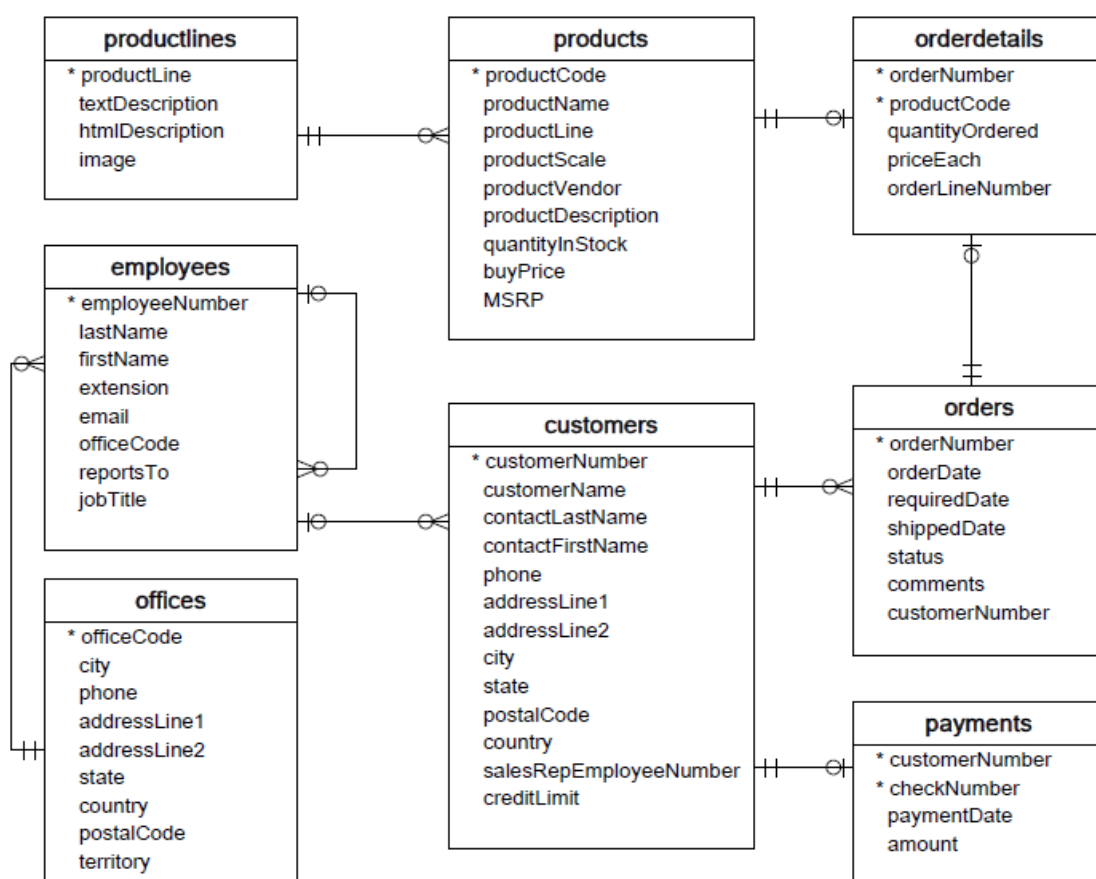


图 2-3 Classicmodels 数据库的数据库模式图

Figure 2-3 English caption

实验所使用的自然语言查询数据集由作者根据 Classicmodels 数据库的模式建立, 根据查询意图的复杂度, 分为简单、中等、困难三类。每种类别, 提出了 20 条自然语言查询, 共计 60 条, 用来测试模型的准确性。表2-3给出了三种类别的自然语言查询示例。

表 2-3 自然语言查询实例

Table 2-3 A Table

查询语句复杂度	示例
简单	return all the customers
中等	return all the offices in London
困难	return the city of office that has an employee named Diane

2.4.2 实验结果

首先将所有自然语言查询语句, 输入系统, 经过各组件的处理, 得到系统生成的 SQL 语句, 再判断其是否正确表达了自然语言查询语句的意图。实验结果如表2-4所示。

表 2-4 模型生成 SQL 语句的准确性

Table 2-4 A Table

查询语句复杂度	准确率
简单	100%
中等	80%
困难	35%

实验还检验了在没有用户交互指定节点映射、选择合适树结构的情况下, 模型生成 SQL 语句的准确性, 如表2-5所示; 由于没有选择合适树结构, 导致解析树结构调整之后有多个候选结果, 最后返回结果也会有多个, 故只要返回结果中包含正确 SQL 语句, 就视做正确生成:

2.4.3 实验分析

1. 经过两次实验, 分别检验完整模型和无交互机制模型生成 SQL 语句的准确性。可以看出, 交互机制对于模型的准确性有很大的提升; 而加入了交互机制后, 模型可以比较准确的处理简单和中等复杂度的查询意图, 即便是复杂度较高的查询

- 意图, 本文提出的模型仍然可以正确生成一部分困难复杂度的 SQL 语句。在实验过程中, 经常出现节点歧义需要映射, 如 “price” 一词可以映射到 “products” 表中的 “buyPrice”, 也有可能映射到 “orderdetails” 表中的 “priceEach”。那么对于测试语句 “return order details whose price is higher than 50”, 如果没有交互机制, 对于 “price” 这个节点的映射就会优先映射到 “products” 中的 “buyPrice”, 而实际上应该映射到 “orderdetails” 表中的 “priceEach”。可以看出, 交互机制在节点映射这一部分准确率的提升, 大大影响了整体模型的准确率。
2. 在作者构造的自然语句查询意图数据集中, 简单复杂度的查询意图大体上是返回某数据表中某一字段, 中等复杂度的查询意图会添加一些聚合函数和简单过滤条件, 困难复杂度的意图会增加更多的聚合函数和跨表查询, 总体上来说结构都比较简单。在表 3-5 所示的结果中, 困难复杂度意图的错误生成情况, 一部分来源于查询意图对应的 SQL 语句需要子查询, 如 “return the customer who has the most orders”。目前的模型还不能很好的处理这种情况, 在解析树语法和翻译过程中都还没考虑子查询的情况, 这可能是接下来需要进一步进行的工作, 即提升模型的处理能力, 使其能处理更复杂的查询意图。
 3. 目前模型基于相似度的节点映射机制仍然有不足之处, 如 “return the mobile number of customer whose name is Australian Gift Network” 这一查询意图, 对于顾客名称 “Australian Gift Network”, 系统会将其映射为三个不同的节点, 导致生成出错。尽管模型对于数据库模式中的一些连接词或短语, 如 “thenumberof”、“customername”, 进行了特殊处理, 但对于上文所示的这些特殊短语或词语, 目前没有较好的方法来进行处理。
 4. 本文自行建立了一个解析树节点类型与自然语言单词的映射词典 (表2-1), 能处理一些常见的单词与 SQL 语法的对应关系, 如 “return”、“in”、“have”、“thenumberof”, 但映射关系仍然不足, 所以本实验所使用的测试查询意图都需要使用这些比较固定的词语来构建。事实上, 自然语言的表达非常多样, 如果要增加模型的处理

表 2-5 模型在无交互情况下生成 SQL 语句的准确性

Table 2-5 A Table

查询语句复杂度	准确率
简单	65%
中等	50%
困难	15%

能力，扩充这个映射词典也是非常必要的。

5. 尽管人机交互机制对于节点映射的准确率有较大提升，但根据观察，立足于目前数据量较小、数据库模式简单的前提下，这一机制能保证映射相似度排名前五的 SQL 组件包含正确的映射关系；但随着数据库数据量的增加、数据库模式的复杂化，目前节点映射的相似度计算机制，可能无法确保正确的 SQL 组件能有较高的相似度。

2.5 本章小结

本章提出了从自然语言到 SQL 语句的模型的设计和具体实现，基本思想是结合自然语言解析技术和人机交互思想，用人机交互的方式指导解析过程，减轻自然语言解析的歧义问题和错误情况，相比于纯自然语言解析技术，大大提升了准确性。由于准确性得到了提升，该模型的可用性也就得到了提升。

本章针对自然语言生成 SQL 模型的准确性进行了实验，实验数据库为 MySQL 样例数据库 classicmodels，测试用自然语言查询意图为作者自行根据实验数据库建立，分别包含简单、中等、困难三种复杂度的查询意图。以此为测试数据集，对完整模型和无交互机制的模型进行了实验，并给出了实验结果。结果发现，完整模型能较好的处理简单和中等复杂度的查询意图，一定程度上能处理困难复杂度的查询意图，而且经过与无交互机制的模型进行对比，发现交互机制对于模型准确度的提升十分显著。但对实验结果进行细致分析，也发现模型存在处理能力仍然不足、无法处理子查询、自然语言意图形式比较固定、无法处理特殊词组等问题，需要进一步进行提升和改进。

第三章 基于深度强化学习的 NL2SQL 生成

3.1 研究问题

基于深度强化学习的 NL2SQL 生成的模型的总输入表示为 x ，其包含由单词 w_i 组成的自然语言问题 w 以及由列名 c_j 组成的数据库单张表的模式 c （其中，列名 c_j 可由单个或多个单词组成）。模型的输出为一条可执行的 SQL 查询语句 y 以及其在数据库中执行的结果 r 。我们通过给出 NL2SQL 任务中一个典型的例子（如图3-1所示）来进一步说明。

输入： What is the height of Willis Tower in Chicago?

Rank	Name	Location	Height(ft)	Floor	Year
1	One World Trade Center	New York City	1,776	104	2014
2	Willis Tower	Chicago	1,451	108	1974

输出： SELECT 'Height(ft)' WHERE Name='Willis Tower' AND Location='Chicago'

执行结果： 1,451

图 3-1 NL2SQL 的一个典型示例

Figure 3-1 English caption

在图3-1中， w 代表自然语言问题 “What is the height of Willis Tower in Chicago?”，其中 w_0, w_1, w_2, \dots 分别代表单词 “What”、“is”、“the” 等。 c 代表数据库表的模式 “Name Location Height(ft) Floor Year”，其中 c_0, c_1, c_2, \dots 分别代表单词 “Name”、“Location”、“Height(ft)” 等。总输入 x 代表由 w 和 c 组成的集合。模型的输出 y 在该示例中对应的 SQL 查询语句为 “SELECT ‘Height(ft)’ WHERE Name= ‘Willis Tower’ AND Location= ‘Chicago’”，其在数据库中执行的结果 r 为 1451。

基于深度强化学习的 NL2SQL 生成的关键问题在于如何去理解自然语言语句的意图并在一次交互的状态下将意图映射到 SQL 查询语句上，即在知道数据库表模式 c 的状态下，将用户输入的自然语言问题 w 转换为一条 SQL 查询语句 y 并得到数据库执行结果 r 。

3.2 相关技术

3.2.1 NL2SQL 研究现状

3.2.2 深度学习

3.2.3 强化学习

3.2.4 语义解析

3.3 解决方案

3.3.1 增强解析器模型

针对每个输入 x 来生成结构化的输出 y 的过程可以被分解成为一系列的语义解析决策的过程。所以，增强解析器模型的基本思路为：解析器从初始状态启动并不断根据学习的策略采取不同的操作。每个动作 (action) 都会将解析器从一个状态 (state) 转移为另一个状态，知道解析器到达它的最终状态并停止。在解析器的最终状态下，我们可以获取一个完整的输出 y 。我们采取一种概率的方法来对整个的策略 (policy) 建模。它可以对由输入 x 产生的有效的动作 (action) 的集合以及解码器产生的历史行为进行概率分布进行预测。所以，整个增量语义解析器的训练目标就转换为了如何优化一个参数化的策略的问题。

$$P_{\theta}(y|x) = P_{\theta}(\mathbf{a}|x), \quad \theta \text{ 为模型参数} \quad (3-1)$$

根据公式3-1，通过执行动作 (action) 序列 $\mathbf{a} = \{a_1, a_2, \dots, a_k\}$ ，解析器将被不断引导并从初始状态转换为包含输出 y 的最终状态。在此，我们需要假设每个输出 y 有且仅有一个对应的动作序列 \mathbf{a} (详见3.3.5节内容)。行动序列的概率 $P_{\theta}(\mathbf{a}|x)$ 可展开为增量策略概率的乘积 (公式3-2)：

$$P_{\theta}(\mathbf{a}|x) = \prod_{i=1}^k P_{\theta}(a_i|x, a_{<i}), \quad |\mathbf{a}| = k \quad (3-2)$$

在推断 (inference) 期间，我们的模型并非尝试枚举整个输出空间并找到最高的得分 $\mathbf{a}^* = \arg \max_{\mathbf{a}} P_{\theta}(\mathbf{a}|x)$ ，而是在解码器中采用了一种贪心的方法：在每一步都根据策略 (policy) 来选择得分最高的行动，即 $a_i^* = \arg \max_{a_i} P_{\theta}(a_i|x, a_{<i}^*)$ 。

在后面的几节中，我们将给出解析器状态 (state) 的定义以及动作 (action) 清单，还会详细介绍整个基于编码器-解码器神经网络体系结构的增强解析器模型。

3.3.2 动作序列

首先，我们给出对应于图3-1中示例的完全解析之后的结构化表示：

$$\begin{bmatrix} AGG & NONE \\ SELCOL & Height(ft) \\ COND & \left\langle \begin{bmatrix} COL & Name \\ OP & = \\ VAL & WillisTower \end{bmatrix}, \begin{bmatrix} COL & Location \\ OP & = \\ VAL & Chicago \end{bmatrix} \right\rangle \end{bmatrix}$$

因此，解析器的中间状态就被这样分部表示，其中一些尚未填充的特征值表示为 ϵ 。

解析器的初始状态 p_0 的值为空和空列表，表示为 $\begin{bmatrix} AGG & \epsilon \\ SELCOL & \epsilon \\ COND & \langle \rangle \end{bmatrix}$ 。

除此之外，我们还需要定义动作 (action) 清单。每个动作可以将解析器的状态从一个状态转换为另一个状态，即 $p \rightarrow p'$ 。我们设 $p = \begin{bmatrix} AGG & agg \\ SELCOL & selcol \\ COND & cond \end{bmatrix}$ ，并且在表3-1中给出每个动作执行后的转移状态 p' 。

表 3-1 动作 (action) 清单

Table 3-1 A Table

动作 (action)	由状态 p 执行该动作之后的状态	参数表示
AGG(agg)	$p[AGG \mapsto agg]$	-
SELCOL(c_i)	$p[SELCOL \mapsto c_i]$	r_i^C
CONDCOL(c_i)	$p[COND \mapsto p.COND \begin{bmatrix} COL & c_i \\ OP & \epsilon \\ VAL & \epsilon \end{bmatrix}]$	r_i^C
CONDOP(op)	$p[COND_{-1} \mapsto p.COND_{-1}[OP \mapsto op]]$	-
CONDVAL($w_{i:j}$)	$p[COND_{-1} \mapsto p.COND_{-1}[VAL \mapsto w_{i:j}]]$	r_i^W and r_j^W
END	$p(\text{最终状态})$	-

需要说明的是，在表3-1中，在解码器中所使用的参数表示将在3.3.4节中进行解释； $p[AGG \mapsto agg]$ 表示与状态 p 处于同一状态，不过其特征值 AGG 已经被赋值为 agg ； $||$ 表示列表展开； $COND_{-1}$ 表示在列表中的上一个元素；

值得注意的是，动作 $CONDVAL$ 会从输入问句 w 中选择所需的文字段 $w_{i:j}$ 。但这

样做会导致一个问题，它将产生大量的动作，其数量级为问题长度的二次方。因此，我们将动作 *CONDVAL* 分解为两个连续的子动作，一个去选择起始位置 w_i ，另一个则选择终止位置 w_j 。在动作序列的最后，我们需要增加一个 *END* 动作来执行解析过程并使解析器进入结束状态。举例来说，图3-1中的例子可以看作是如下的一个动作序列：

1. *AGG(NONE)*
2. *SELCOL(c_3)*
3. *CONDCOL(c_1)*
4. *CONDOP(=)*
5. *CONDVAL($w_{5:6}$)*
6. *CONDCOL(c_2)*
7. *CONDOP(=)*
8. *CONDVAL($w_{8:8}$)*

该节中的定义是基于所有有效序列都具有 *AGG SELCOL (CONDCOL CONDOP CONDVAL)* END* 形式的假设之上。也就保证了我们可以从所有的最终状态中提取出完整的逻辑形式出来。对于其他不同结构的 SQL 语句来说，我们还需要重新设计动作的清单以及解析器的状态。

3.3.3 编码器

基于深度强化学习的 NL2SQL 生成的技术方案如图3-2所示。包括 xxxxx（此处有一大段）。

其中基于深度强化学习的 NL2SQL 生成网络包含编码器和解码器两个部分，如图3-3所示。

编码器的具体步骤如下：

1. 对于输入的句子 w 中的每个单词 w_i ，先将其使用词向量进行向量编码 (word embedding[!! 引用!!])。
2. 将其送入一个双向的长短期记忆网络 (bi-directional Long Short-Term Memory, 简称 bi-LSTM)，其中每个细胞 (cell) 会有一个隐状态 h_i^W 。
3. 由于一个列名可能由一个或多个单词构成，我们对每个列名先进行词向量编码并输入一个 bi-LSTM 中，再使用从 bi-LSTM 中得到的最终的隐状态作为列名初始表示 (initial representation)。
4. 使用自注意力机制 (self-attention[!! 引用!!]) 将该初始表示转换为 h_j^C 。
5. 在得到基于内容的表示 h_i^W 和 h_j^C 之后，使用 cross-serial dot-product attention[!! 引用!!] 得到 h_j^C 和 h_i^W 的加权平均值并作为单词 w_i 和 c_j 的上下文向量。

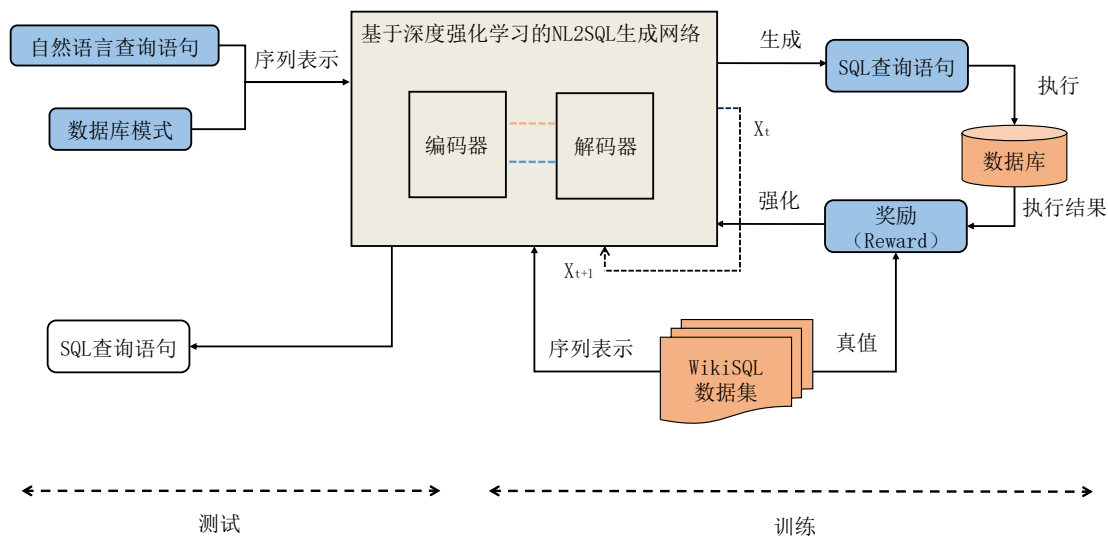


图 3-2 基于深度强化学习的 NL2SQL 生成的技术方案

Figure 3-2 English caption

6. 将这两个上下文向量进行拼接并分别送入使用自然语言问题和列名作为输入的 bi-LSTM 中。这两个 LSTM 网络的隐状态就是我们所期望的和上下文相关的表达 r_i^W 和 r_j^C 。

3.3.4 解码器

在3.3.3节中，我们已经获得了对于单词 w_i 上下文相关的表示 r_i^W 和以及列名 c_j 的上下文相关的表示 r_j^C 。接下来是解码器部分的设计与细节。

首先，解码器的目标是为了对由输入 x 和历史活动 $a_{<i}$ 构成的概率分布 $P_\theta(a|x, a_{<i})$ 进行建模。其主要包括以下两点挑战：

1. 活动 (action) 的不确定性：所有活动均需要取决于输入以及当前解析器的状态，不存在固定的活动。
2. 解析器的决策完全依赖于上下文信息：解析器依赖于解码历史信息以及输入的问题和列名信息。

为了解决第一个问题，我们使用了基于 LSTM 的解码器架构并使用活动的独立打分机制。模型给每个候选的活动 a 打上分数 s_a 并且使用 *softmax* 函数将分数正则化 (normalize) 到一个概率分布上。对于时刻 i 来说，我们用 h_i^{DEC} 来表示当前解码器的隐状态并且使用双线性函数 $s_a = (h_i^{DEC})^T U^A r_a^A$ 来给分数 a 建模。双线性函数中 r_i^A 就是活动

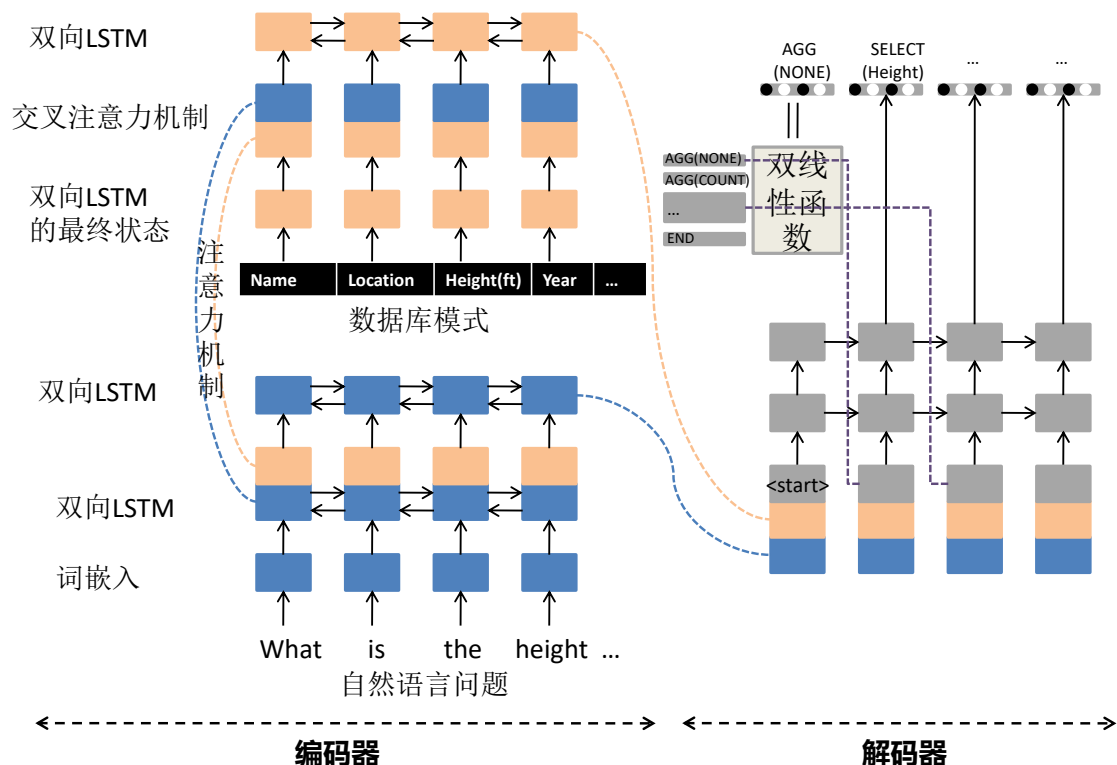


图 3-3 基于深度强化学习的 NL2SQL 生成网络

Figure 3-3 English caption

a 的向量表示并且是由活动嵌入 (action embedding) 和参数表示 (parameter representation) 建模得到，其中参数表示已经在表3-1中给出。

我们使用 dot-product attention mechanism[!! 引用!!] 来捕获解析器的决策和输入的问题以及列名之间的依赖关系。之后，将前一时刻 i 的输出活动表示 $r_{a_i}^A$ 、自然语言问句的注意力向量 e_i^W 以及列名的注意力向量 e_i^C 拼接在一起，作为 $i+1$ 时刻的 LSTM 解码器的第一层的输入。其中，向量 $e_i^C = \sum_j \alpha_{i,j} r_j^C$, $\alpha \propto h_i^{DEC} \cdot r_j^C$ 。向量 e_i^W 的定义相似可得。

3.3.5 过滤条件顺序问题和隐式列名问题

在前文中，我们做出过一个假设，即每个自然语言问题都只会对应一条 SQL 查询语句，并且每个查询语句都只会对应一个正确的动作序列。实际上，这个假设并不能完全成立。一个简单的例子就是出现在 WHERE 子句里面的过滤条件的顺序并不确定，不同

的过滤条件的顺序将会导致不同的动作序列。除此之外，我们在3.3.5.2节中还介绍了另一个会产生歧义的情况，它会产生一个自然语言问题可以转换为多种 SQL 查询语句且其执行结果仍保持一致的问题。对于最终用户而言，这些查询语句其实是完全等效的。

针对上述的两种情况，本文会将每个训练样本转换为多个序列以供参考并且在训练过程中使用多种目标策略。得益于语法分析领域的灵感，本文会采用一个非确定的预言 (oracle) 从而使得解析器可以在多个动作序列中进行探索。特别注意的是，在前文中所述的训练机制都是采用基于静态预言而非非确定预言的。

我们假定预言 (oracle) O 可在时刻 t 返回一组正确且连续的动作 $O(x, a_{<t})$ 。其中，将 $O(x, a_{<t})$ 集合中的任何一个对象都可以执行解析并得到正确的结果。对于每一个训练样本 L_x 来说，其训练目标将被如公式3-3定义。

$$L_x = \sum_{i=0}^k \log \sum_{a \in O(x, a_{<i})} P_{\theta}(a|x, a_{<i}) \quad (3-3)$$

其中 $a_{<i}$ 表示动作序列 a_1, \dots, a_{i-1} ， $a_i = \arg \max_{a \in O(x, a_{<i})} s_a$ 代表在训练时解析器判定的置信度最高的动作。当 O 是静态预言时，其只会包含一个单独且正确的动作，公式3-3可被化简为一个简单的交叉熵的损失函数。而当 O 是非确定的预言时，在训练期间，解析器会对多个不同的但是同样是正确的动作序列进行解析而不再是一个唯一的动作序列。

3.3.5.1 解决过滤条件顺序问题

训练一个形如 text-to-SQL 的解析器往往会碰到过滤条件顺序的问题。在 SQL 查询语句中，过滤条件的先后顺序不会对语句的意图和执行结果产生影响。但是在增强解析器模型中，我们需要将查询语句固定下来然后一句预定义的格式进行执行。预测出的与 groundtruth 的标签不同但是完全正确的语句可能没有办法得到适当的汇报 (reward)。

有一些前人的研究使用了增强学习 [!! 引用 seq2sql!!] 或模块化的序列集成 [!! 引用 Xu et al., 2017] 的方案来解决过滤条件顺序的问题。其中，前者在一定程度上解决了这个问题但明显降低了模型的优化稳定性并导致训练时间大大增加。而后者采用了一种较为复杂的模型设计来捕获子句之间的相互依赖关系，其预测出的过滤条件的信息将被用于预测下一个过滤条件。

在本章的方案中，本文利用一个非确定性的预言来解决过滤条件的顺序问题。我们的模型结合了增量式方法的优势，利用子句之间的相互依赖性和模块化的方法来获得更高质量的训练信号。具体来说，在预测下一个过滤条件的几个步骤中，模型会接受所有可能的后续条件（即尚未预测出的过滤条件）而忽视其位置所在。对于图3-1中给出的示例，除了本文在3.3.2节中给出的转换序列之外，我们提出的非确定性的预言也接收

$CONDCOL(c_2)$ 作为第二个动作的正确的延续。如果模型即将预测出第一个动作，那么它将在预测第一个动作之前继续去预测第二个过滤条件是什么。

3.3.5.2 解决隐式列名问题

在模型的初步验证实验过程中，我们观察到解析器模型在开发集上会产生一些错误的预测，其主要的出错原因是隐式列名的问题。在很多场景下，自然语言的问题中并没有明确提到过滤条件中所需要的列名。例如：在图3-1中的问题中就没有直接提到“Name”这个列名。同样地，“What is the area of Canada?”这个典型的自然语言问题中也没有出现关键词“country”。对于我们人类而言，这种隐式的表达会使得我们的自然语言的问题更加简洁而且我们可以很快地从上下文的相关内容中轻易地推断出缺失的信息。但是对于机器学习的模型来说，这些都会是巨大的挑战。

本文中，我们使用非确定性的预言中 $ANYCOL$ 来学习上诉所提及的隐式列名的问题。在模型运行过程中，我们将预测根据过滤条件进行拓展从而模拟人类如何轻松找到未在问题中出现的列名。在图3-1中的例子中，除了动作 $CONDCOL(c_1)$ 外，我们还允许模型进行另一种的预测 $CONDCOL(ANYCOL)$ 。当后者出现时，比如 $ANYCOL = 'WillisTower'$ ，我们会将它拓展成为一种符合的子句如 $Rank = WillisTowerORName = WillisTowerOR\dots$ 。当列名没法被明确地和过滤值解析出来时，非确定预言可以同时接收 $ANYCOL$ 或者列名并且让我们的模型区预测哪种形式更容易被学习。

3.4 实验与分析

3.4.1 数据集及评价指标

WikiSQL 是由 [!! 引用!!]Zhong et al. (2017) 提出的第一个由自然语言问题、对应的 SQL 查询语句及数据表 schema 构成的大型数据集。WikiSQL 虽然在 SQL 语法上的覆盖范围比之前的数据集（例如，ATIS[!! 引用!!] 和 GeoQuery[!! 引用!!]）要弱，但是在问题、数据表模式和内容上具有很高的多样性。这也使其在近期吸引了大量的研究人员的关注并使用神经网络进行建模和评估 [!! 引用!!]。

在本章的实验中，我们采用了 WikiSQL 数据集中的默认划分，将数据集划分为训练集、验证集和测试集。我们在验证集和测试集上对训练后的分别使用静态预言和非确定性预言的模型进行了评价。在最后，我们会给出逻辑形式准确性（即 SQL 查询的精准匹配度）和执行准确性（即在数据库执行后可以产生相同的正确结果的比例），其中执行准确性是我们优化的关键指标。

3.4.2 实验设置

在数据预处理过程中,我们大量借鉴了 [!! 引用 !!] 的方法对数据进行处理。在词嵌入层 (embedding layer) 在训练集中出现至少两次的单词才会被保留在词汇表 (vocabulary) 中,剩下的将被赋予特殊的输入单元 “UNK”。我们使用已经训练好的 Glove 词嵌入 [!! 引用 !!] 并且可以在我们的模型训练时对词嵌入进行继续训练,该词嵌入的维度为 16 维。我们还使用 [!! 引用 !!] 的方法对自然语言问题和列名进行了类型嵌入 (type embedding): 对于每一个单词 w_i 和 c_j , 我们会判断其是否出现在列名中并给出一个离散的表示,而这些特征会被嵌入为 4 维的向量表示。类型嵌入将会和词嵌入连接在一起并作为双向 LSTM 网络的输入。

另外,还需要说明的是,编码阶段的双向 LSTM 的单隐藏层的大小为 256 维,其中单向为 128 维。解码阶段的 LSTM 由大小各为 256 的两个隐藏层组成。所有的注意力连接都采用 3.3.4 节中描述的点积形式。

在训练阶段,我们设置批尺寸 (batch size) 为 64 并使用 0.3 的丢失率 (dropout rate) 进行正则化处理。同时,我们使用了 ADAM 优化器 [!! 引用 !!]、设置初始学习率为默认值 0.001 用于参数的更新迭代以及将梯度修剪到 5.0 用于保证训练的稳定。

3.4.3 实验结果

3.4.3.1 WIKISQL 实验结果

图 3-1 为 WikiSQL 数据集 [!! 引用 !!] 中的一个样例。WikiSQL 数据集是纯自然语言生成 SQL 查询语句的第一个数据集。其中包含 80654 组自然语言问句及其对应的 SQL 查询语句,涵盖 24241 张从 Wikipedia 中获取的数据表。从图中可以看到, NL2SQL 任务的输入实际包括两部分: 自然语言问题以及一个简单的数据表 schema (schema 代表数据表及表中的列)。

WikiSQL 数据集中 SQL 查询语句具有一定的约束条件,必须符合如下模板:

表 3-2 SQL 查询语句模板

Table 3-2 A Table

*SELECT agg selcol WHERE col op val (AND col op val)**

在 3-2 中, *selcol* 代表表中的列名, *agg* 代表聚合操作 (例如: COUNT, SUM 或空), *WHERE* 后面为由一系列过滤调教构成的子句, 每个 *op* 代表一个过滤操作 (例如: “=”), *val* 代表出现在自然语言问句中的过滤值。值得注意的是, 尽管模板中的过滤条件服从标准的线性顺序, 但由于存在 *AND* 符号, 所以过滤条件的先后顺序是无关紧要的。

表 3-3 WikiSQL 中验证集和测试集的准确度

Table 3-3 A Table with footnotes

模型	验证集		测试集	
	Acc_{lf}	Acc_{ex}	Acc_{lf}	Acc_{ex}
TypeSQL 模型 (引用)	72.5	79.0	71.7	78.5
SQLNet 模型 (引用)	76.1	82.0	75.4	81.4
本文提出的模型				
增强解析器 (静态预言)	76.1	82.5	75.5	81.6
增强解析器 (非确定性预言, 不解决过滤条件顺序问题)	75.4	82.2	75.1	81.8
增强解析器 (非确定性预言)	49.9	84.0	49.9	83.7
增强解析器 (非确定性预言) + EG (5)	51.3	87.2	51.1	87.1

实验的主要结果如表3-3所示。其中 Acc_{lf} 表示逻辑形式准确度, Acc_{ex} 表示执行准确度, “+ EG (5)” 表示使用执行引导策略且集束大小为 5。

可以看到我们的增强解析器模型在使用静态预言的情况下就可以达到目前该领域的最高水平 ([!! 引用!!]) 的范围。在此基础之上, 使用了非确定性预言的增强解析器在执行准确度上还可以大幅度提升 2.1%, 但是其逻辑形式准确度反而会有所下降, 主要原因是使用了 *ANYCOL* 进行列的选择时, 生成的 SQL 查询语句和真值 (groundtruth) 不能完全匹配。我们还对过滤条件顺序问题和非确定性预言中的 *ANYCOL* 的贡献度进行了实验对比。当我们的非确定性预言只用来解决3.3.5.1节中的过滤条件顺序问题时, 模型的性能和使用静态预言的模型大致相同。我们认为这是因为在训练集中所出现的过滤条件的序列变化已经足够丰富, 序列到动作的模型已经可以学习的很好。除此之外, 在预言中增加 *ANYCOL* 可以更好地捕捉到隐式的列明并提升性能, 使得执行准确度从 81.3% 增加至 83.7%。

我们的增强解析器模型在解码阶段使用的是贪心策略, 即选择通过策略预测出的获得最高得分的动作。同样, 我们也可以使用集束搜索 (beam search) 来进行搜索。其中, 集束搜索是一种启发式的图搜索算法, 为了减小搜索的空间与时间, 在向图进行深度扩展时, 去裁剪掉一些质量较差的节点, 保留一些质量较高的节点。使用集束搜索意味着我们可以在更大的搜索空间中进行搜索。另外, 我们还将执行引导策略和集束搜索策略结合在一起, 进行执行引导的解码器可以避免生成语义错误的 SQL 查询语句, 例如会产生运行时错误 (runtime error) 和空结果 (empty result) 错误的 SQL 查询语句。除此之外

有一点很关键，那就是将生成出的部分的输出放到数据库中的执行，得到的执行结果可以用来指导解码过程。解码器可以维持一种部分输出的状态，该状态由聚合操作、选择列操作和完整的过滤条件构成。在每个动作执行之后，由执行引导策略引导的解码器将保留前 K 个得分最高的且没有运行错误的 SQL 查询语句，在结束状态时从这 K 个语句中选择出最有可能的那个查询语句。在 $K=5$ 时，我们的模型在执行引导策略下载测试机上实现了 87.1% 的执行准确度。

我们在表3-4中给出了具有执行引导过程的解码器在静态预言下的模型性能。其中“+ EG (k)”的含义同表3-3， k 为集束大小。其性能与非确定性预言下模型的性能非常接近，但需要更大的集束，也就意味着其解码的时间会大大增加。

表 3-4 使用不同集束大小的模型性能

Table 3-4 A Table

预言类型	w/o EG	+ EG (1)	+ EG (3)	+ EG (5)
静态	81.6	83.5	86.4	86.7
非确定	83.7	86.0	87.1	87.1
运行速度 (每秒运行的样本数)	48.3	30.1	8.2	4.4

3.4.3.2 其他数据集实验结果

除了在 WikiSQL 进行实验以为，本文还在 ATIS[!! 引用 X!!] 和 Spider 数据集 [!! 引用 X!!] 上进行了补充实验。

ATIS 数据集实验

为了测试我们的模型是否可以推广到其他的数据集上，本文还是用了 ATIS 数据集进行了实验，数据在表3-5中给出。ATIS 数据集包含更多样的 SQL 结构，包括对多个表的查询以及嵌套查询。为了和本文的任务设置相兼容，我们仅保留了 ATIS 数据集中没有嵌套查询的样本，即仅包含 AND 操作不包含 INNER JOIN 操作符。我们会把多个相关的表进行 JOIN 连接并构造成一个数据表以便我们的模型进行输入。经过简化之后数据集共包含 933 个样本并分布在训练集、验证集和测试集中，并分别有 714、93 和 126 个样本。

我们使用静态预言和不使用 ANYCOL 的非确定预言的模型进行训练，在测试集上分别达到了 67.5% 和 69.1% 的准确性。使用非确定预言的模型在性能上有大幅增强，这也验证了我们之前的假设：与 WikiSQL 相比而言，ATIS 是一个小得多的数据集，因此

表 3-5 ATIS 数据集上的实验结果

Table 3-5 A Table with footnotes

模型	验证集		测试集	
	Acc_{lf}	Acc_{ex}	Acc_{lf}	Acc_{ex}
增强解析器（静态预言）	87.1	88.2	65.9	67.5
增强解析器（非确定性预言, 不解决过滤条件顺序问题）	88.1	89.2	68.3	69.1

可以解决过滤条件顺序问题的模型的效果会有不小提升。另外，需要说明的是，由于 ATIS 数据的性质，没有对 *ANYCOL* 进行测试。

Spider 数据集实验

Easy

What is the number of cars with more than 4 cylinders?

```
SELECT COUNT(*)
FROM cars_data
WHERE cylinders > 4
```

Hard

Which countries in Europe have at least 3 car manufacturers?

```
SELECT T1.country_name
FROM countries AS T1 JOIN continents
AS T2 ON T1.continent = T2.cont_id
JOIN car_makers AS T3 ON
T1.country_id = T3.country
WHERE T2.continent = 'Europe'
GROUP BY T1.country_name
HAVING COUNT(*) >= 3
```

Meidum

For each stadium, how many concerts are there?

```
SELECT T2.name, COUNT(*)
FROM concert AS T1 JOIN stadium AS T2
ON T1.stadium_id = T2.stadium_id
GROUP BY T1.stadium_id
```

Extra Hard

What is the average life expectancy in the countries where English is not the official language?

```
SELECT AVG(life_expectancy)
FROM country
WHERE name NOT IN
(SELECT T1.name
FROM country AS T1 JOIN
country_language AS T2
ON T1.code = T2.country_code
WHERE T2.language = "English"
AND T2.is_official = "T")
```

图 3-4 spider 数据集样例

Figure 3-4 English caption

spider 数据集是一个由 11 名耶鲁大学学生标注的大型的跨域的包含语义解析和文

本到 SQL 语句的数据集。其挑战的目标为开发在跨域的数据集上开发自然语言接口。spider 数据集由 200 个数据库中的 10181 个问题和 5693 个 SQL 查询语句组成，并且其中有很多表覆盖了 138 个不同的域。不同复杂程度的 SQL 查询会出现在训练集和测试集中，如图3-4中所示。这也就要求模型需要具有较好的拓展性，从而适应新的 SQL 查询语句以及新的数据库模式。

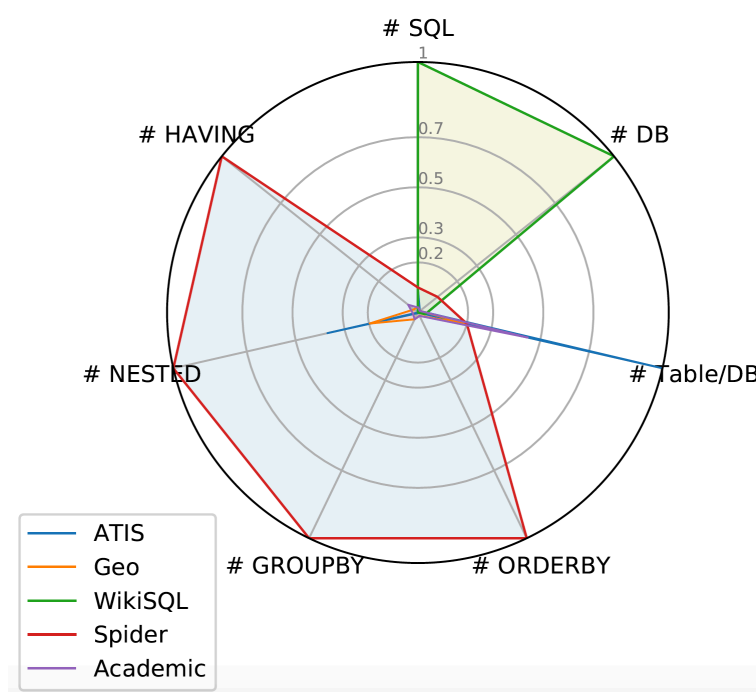


图 3-5 spider 数据集的组成

Figure 3-5 English caption

如图3-5所示，spider 数据集的 1.0 版本有别于其他之前的所有 SQL 语义解析的数据集：

- 对于 ATIS, Geo 和 Academic 数据集来说，每个数据集都只包含在一个数据库中执行的若干个 SQL 查询，并且 SQL 查询在训练集和测试集的分割时完全相同的。
- WikiSQL 数据集中的 SQL 查询和表的数量都相对较大，但是所有的 SQL 查询语句都较为简单，并且每个数据库都只有一个没有任何外键的表。

所以，在 spider 数据集上的实验能够说明模型从简单的 SQL 查询语句生成到复杂的 SQL 查询语句生成的能力，说明模型的通用性与泛化性。

从表3-6可以看出，我们使用 ANYCOL 的非确定预言的模型进行测试，在测试集上分别达到了 23.2% 和 24.1% 的准确性，超过了之前提出的所有方法。使用非确定预言的模型可以很好地解决过滤条件顺序问题，所以在性能上表现优异。同时，得益于

表 3-6 Spider 数据集上的实验结果

Table 3-6 A Table with footnotes

模型	验证集	测试集
Seq2Seq + attention	1.8	4.8
TypeSQL	8.0	8.2
SQLNet	10.9	12.4
SyntaxSQLNet	18.9	19.7
增强解析器（非确定性预言）+ EG (5)	23.2	24.1

ANYCOL 机制对隐式列名问题的处理，在 spider 数据集上的表现证明了模型具有很好的泛化性能。

3.5 本章小结

第四章 基于多任务学习的 NL2SQL 生成

4.1 研究问题

在第三章中，我们结合了深度学习与深度强化学习来解决 NL2SQL 的问题。但是，目前的所有研究都只是将英文的自然语言问题转换为机器可执行的 SQL 查询语句，而没有涉及到其他语种的自然语言。在本章中，我们主要的研究问题就是研究如何进一步提升 NL2SQL 任务中英文自然语言生成 SQL 查询语句的准确性以及如何同时解决中文自然语言问题转换为 SQL 查询语句的问题。其中最具挑战的地方就是怎样将中文自然语言转换为英文自然语言的任务与英文自然语言生成 SQL 查询语句有机地结合起来。

基于多任务学习的 NL2SQL 生成的模型的总输入表示为 x ，其包含由单词 w_i 组成的自然语言问题 w 以及由列名 c_j 组成的数据库单张表的模式 c （其中，列名 c_j 可由单个或多个单词组成）。模型的输出为一条可执行的 SQL 查询语句 y 以及其在数据库中执行的结果 r 。我们通过给出 NL2SQL 任务中一个典型的例子（如图3-1所示）来进一步说明。

输入：芝加哥的威利斯大厦有多少层？

Rank	Name	Location	Height(ft)	Floor	Year
1	One World Trade Center	New York City	1,776	104	2014
2	Willis Tower	Chicago	1,451	108	1974

中间结果：How many floors does the Willis Tower in Chicago have?

输出：SELECT 'Floor' WHERE Name='Willis Tower' AND Location='Chicago'

执行结果：108

图 4-1 中文自然语言问题生成 SQL 查询语句的示例

Figure 4-1 English caption

在图4-1中， w 代表中文自然语言问题“芝加哥的威利斯大厦有多少层？”，其中 w_0, w_1, w_2, \dots 分别代表单词“芝加哥”、“的”、“威利斯大厦”等。 c 代表数据库表的模式“Name Location Height(ft) Floor Year”，其中 c_0, c_1, c_2, \dots 分别代表单词

“Name”、“Location”、“Height(ft)”等。总输入 x 代表由 w 和 c 组成的集合。在将中文的自然语言问题转换为 SQL 查询语句的过程中会生成中间结果 “How many floors does the Willis Tower in Chicago have?”, 即将中文翻译为英文, 记作 m 。模型的输出 y 在该示例中对应的 SQL 查询语句为 “SELECT ‘Floor’ WHERE Name= ‘Willis Tower’ AND Location= ‘Chicago’”, 其在数据库中执行的结果 r 为 108。

基于多任务学习的 NL2SQL 生成的主要任务是要在将中文翻译为英文的同时, 理解中文自然语言语句的意图并在一次交互的状态下将意图映射到 SQL 查询语句上, 即在知道数据库表模式 c 的状态下, 将用户输入的自然语言问题 w 最终转换为一条 SQL 查询语句 y 并得到数据库执行结果 r 。

4.2 相关技术

4.2.1 迁移学习与多任务学习

4.2.2 元学习

4.3 解决方案

在本节中, 我们会介绍如何把中文自然语言翻译为英文自然语言任务与英文自然语言生成 SQL 查询语句任务进行统一, 之后提出一种基于多任务学习的神经网络结构并解释说明为何这样的网络结构可以有效地解决中文自然语言问题生成 SQL 查询语句问题。

4.3.1 TCR 模板


首先, 中文自然语言问题生成 SQL 查询语句可以被划分为两个子任务: 中文自然语言翻译为英文自然语言任务和英文自然语言问题生成 SQL 查询语句任务。在第三章中, 本文提出的基于深度强化学习的解决方案已经可以有效的解决 NL2SQL 问题, 并且在 WikiSQL 数据集和 spider 数据集上有优异的表现。所以, 我们会有一个非常自然的想法, 就是直接使用翻译工具或翻译软件将中文的自然语言问题直接转换为英文自然语言问题, 之后再使用 3.3.1 节中提出的增强解析器模型将英文自然语言转换为 SQL 查询语句。但是, 这种方法会遇到许多问题, 如图 4-2 所示 (其中中文自然语言均使用谷歌翻译自动翻译为英文自然语言):

从图 4-2 中的示例一可以看出, 数据库表模式中的列名为 “College”, 而谷歌翻译将中文 “大学” 翻译为 “University”。而示例二中的 “层” 翻译为了 “layers” 而不是 “floors”。形如这一类的问题会给 SQL 的生成带来很大的问题。

示例1:

中文：有多少来自约克大学的CFL球队？

Pick#	CFL team	Player	Position	College
-------	----------	--------	----------	---------

英文：How many CFL teams are from York University? 
How many CFL teams are from York College? 

示例2:

中文：芝加哥的威利斯大厦有多少层？

Rank	Name	Location	Height(ft)	Floor
------	------	----------	------------	-------

英文：How many layers are there in the Willis Tower in Chicago? 
How many floors does the Willis Tower in Chicago have? 

图 4-2 中文自然语言问题生成 SQL 查询语句的错误示例

Figure 4-2 English caption

究其原因，从中文自然语言到英文自然语言再到 SQL 查询语句这一过程是一个紧密关联的过程，如果将其拆分为两个部分，则翻译的过程中将会丢失大量的数据库、SQL 语言本身的很多信息。所以，我们提出 TCR 模板用以将这两个独立的任务有机的统一起来，并在作为下节中的多任务学习网络的输入。

<u>任务 (Task)</u>	<u>内容 (Content)</u>	<u>结果 (Result)</u>
What is the translation from Chinese to English?	告诉我南澳有哪些注意要点	Tell me what the notes are for South Australia
What is the translation from English to SQL?	The table has column names... Tell me what the notes are for South Australia	SELECT notes from table WHERE 'Current Slogan' = 'South Australia

图 4-3 TCR 模板（任务-内容-结果模板）

Figure 4-3 English caption

如图4-3所示，我们将中文自然语言翻译为英文自然语言任务和英文自然语言生成 SQL 查询语句任务使用 TCR 模板（任务-内容-结果）进行统一。例如，“告诉我南

澳有哪些注意要点”先翻译为英文，其任务为“**What is the translation from Chinese to English?**”（意为需要执行的任务是中文到英文的翻译），内容为“告诉我南澳有哪些注意要点”（指在表明所需要翻译的中文自然语句是什么），结果为“**Tell me what the notes are for South Australia**”。而后再将“**Tell me what the notes are for South Australia**”转换为 SQL 查询语句，其任务为“**What is the translation from English to SQL?**”（意为需要执行的任务是英文到 SQL 的生成），内容为“**The table has column names...Tell me what the notes are for South Australia**”（指在表明所需的信息由表的列名以及英文自然语言的查询构成），结果为“**SELECT notes from table WHERE ‘Current Slogan’ = ‘South Australia’**”，从而完成生成的全过程。

TCR 模板将两个任务的模式进行了统一，使得网络模型的输入和输出得以一致，而不用设计两个不同的网络结构来处理这两个任务。在后文中，我们还将这个模板用于自然语言处理领域中的很多任务，详见4.4节。

4.3.2 多任务网络

由于我们会使用 TCR 模板对每个任务进行统一并且在训练阶段采用联合训练的方法，我们把所使用的神经网络的结构称为多任务网络，如图4-4所示。近几年来，有许多研究者所研究的问答模型的和我们的模型比较类似，但是通常的问答模型往往假设模型得到的答案的片段是可以直接从上下文信息中复制，但是我们的多任务网络适用于更一般的问答场景。在之前的 TCR 模板中，任务对应于问答模型中的问题，内容对应于问答模型中的上下文，而结果对应于问答模型中的回答。由于 TCR 模板中的任务往往包含限制搜索空间的非常关键的信息，我们会在多任务网络中使用了协同注意力机制 [!! 引用!!] 并加以拓展，从而使得对模型输入表示更加丰富。此外，多任务网络还采用了指针机制 [!! 引用!!]，并将其改造为分层的多指针生成器，使得网络能够直接从任务和内容进行文本的复制。

在训练阶段，多任务网络的输入序列会有三个部分，分别为：由 l 个单词构成的内容 c ；由 m 个单词构成的任务 q ；由 n 个单词构成的结果 a 。其中，每个输入序列都是一个矩阵，矩阵中的第 i 行对应于序列中的第 i 个词的嵌入 (embedding)（可为字符向量或词向量），维度为 d_{emb} 。有公式4-1：

$$C \in \mathbb{R}^{l \times d_{emb}} \quad Q \in \mathbb{R}^{m \times d_{emb}} \quad A \in \mathbb{R}^{n \times d_{emb}} \quad (4-1)$$

接下来，我们将分别详细介绍多任务网络中的编码器和解码器。

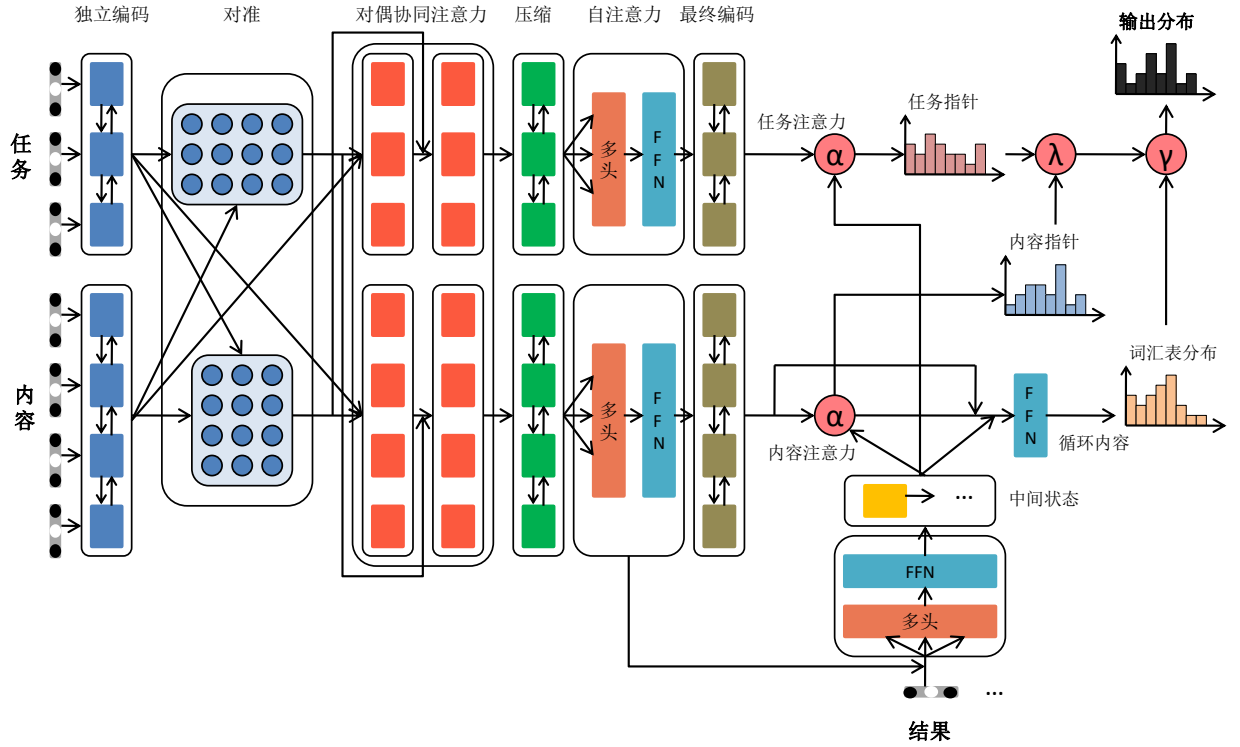


图 4-4 多任务网络

Figure 4-4 English caption

4.3.3 编码器

编码器将内容矩阵 C 、任务矩阵 Q 和结果矩阵 A 作为输入，使用深度栈式循环神经网络 [!! 引用!!]，结合协同注意力机制 (coattention)[!! 引用!!] 和自注意力机制 (self-attention)[!! 引用!!]，来生成编码阶段的最终的表示 $C_{fin} \in \mathbb{R}^{l \times d}$ 和 $Q_{fin} \in \mathbb{R}^{m \times d}$ (即编码之后的内容信息和任务信息)，并且还可以让网络学习到他们之间的局部和全局的相互依赖关系。在下文中，我们将对编码器的每个阶段进行进一步的说明。

单独编码阶段

首先使用一个简单的线性层将输入矩阵投射到 d 维的空间上，如公式4-2。

$$CW_1 = C_{proj} \in \mathbb{R}^{l \times d} \quad QW_1 = Q_{proj} \in \mathbb{R}^{m \times d} \quad (4-2)$$

这些投射之后的表示将被输入给一个共享的双向长短期记忆网络 (bi-LSTM)[!! 引用!!][[注释]，如公式4-3。

$$BILSTM_{ind}(C_{proj}) = C_{ind} \in \mathbb{R}^{l \times d} \quad BILSTM_{ind}(Q_{proj}) = Q_{ind} \in \mathbb{R}^{l \times d} \quad (4-3)$$

对准阶段

在获得协同注意力的表达之前，我们需要先对每个序列的编码表达进行对准。我们会分别给 C_{ind} 和 Q_{ind} 增加一个预训练的嵌入使得现在的 $C_{ind} \in \mathbb{R}^{(l+1) \times d}$ 以及 $Q_{ind} \in \mathbb{R}^{(m+1) \times d}$ ，这样做可以让每个单词不会和序列中的某个单词强制对准。

令 $\text{softmax}(X)$ 表示逐列进行 softmax 操作，即矩阵 X 中的每列进行标准化后的和为 1。我们将内容信息的序列表示与任务序列表示之间进行点积计算得到相似性得分，并使用归一化操作来获得数据的对准，如公式4-4：

$$\text{softmax}(C_{ind}Q_{ind}^\top) = S_{cq} \in \mathbb{R}^{(l+1) \times (m+1)} \quad \text{softmax}(Q_{ind}C_{ind}^\top) = S_{qc} \in \mathbb{R}^{(m+1) \times (l+1)} \quad (4-4)$$

对偶协同注意力阶段

再对准之后，我们要去计算一个序列中的某个单词和另一个序列中相关的单词的加权求和，如公式4-5。

$$S_{cq}^\top C_{ind} = C_{sum} \in \mathbb{R}^{(m+1) \times d} \quad S_{qc}^\top Q_{ind} = Q_{sum} \in \mathbb{R}^{(l+1) \times d} \quad (4-5)$$

对偶协同注意力就是要通过同样的权重将对准时得到的信息传递给原序列，如公式4-6。

$$S_{qc}^\top C_{sum} = C_{coa} \in \mathbb{R}^{(l+1) \times d} \quad S_{cq}^\top Q_{sum} = Q_{coa} \in \mathbb{R}^{(m+1) \times d} \quad (4-6)$$

可以发现，进行了加权求和和协同注意力的表示中的第一列为之前我们添加进去的冗余的嵌入。在这里我们不再需要这一个嵌入，所以将矩阵中的这一列去除掉，得到 $C_{coa} \in \mathbb{R}^{l \times d}$ 和 $Q_{coa} \in \mathbb{R}^{m \times d}$ 。

压缩阶段

为了将对偶协同注意力操作之前的信息压缩至 d 维，我们接着把之前得到的四种序列表示进行拼接并输入一个单独的双向 LSTM 中，如公式4-7。

$$\text{BiLSTM}_{comC}([C_{proj}; C_{ind}; Q_{sum}; C_{coa}]) = C_{com} \in \mathbb{R}^{m \times d} \quad (4-7)$$

$$\text{BiLSTM}_{comQ}([Q_{proj}; Q_{ind}; C_{sum}; Q_{coa}]) = Q_{com} \in \mathbb{R}^{m \times d} \quad (4-8)$$

自注意力阶段

接着，我们使用一个多头的放缩点积注意力机制（multi-head, scaled dot-product attention）[!! 引用 !!] 来捕获每个序列内部的长距离的依赖性，其表达式为公式4-9。

$$\text{Attention}(\tilde{X}, \tilde{Y}, \tilde{Z}) = \text{softmax}\left\{\frac{\tilde{X}\tilde{Y}^\top}{\sqrt{d}}\right\}\tilde{Z} \quad (4-9)$$

$$\text{MultiHead}(\tilde{X}, \tilde{Y}, \tilde{Z}) = [h_1; \dots; h_p]W^o \quad \text{其中 } h_j = \text{Attention}(\tilde{X}W_j^X, \tilde{Y}W_j^Y, \tilde{Z}W_j^Z) \quad (4-10)$$

公式4-9中变换都为线性变换,所以多头注意力机制可以维持 d 维不变,有公式4-11:

$$MultiHead_C(C_{com}, C_{com}, C_{com}) = C_{mha} \quad MultiHead_Q(Q_{com}, Q_{com}, Q_{com}) = Q_{mha} \quad (4-11)$$

之后,我们使用了一个残差前馈网络(feedforward networks, 简称 FFN)如公式4-12,其中 $U \in \mathbb{R}^{d \times f}$, $V \in \mathbb{R}^{f \times d}$ 。其激活函数为 Relu 函数 [!! 引用!!], 并且在输入和输出都使用了正则化(layer normalization)[!! 引用!!]。

$$FFN(X) = \max(0, XU)V + X \quad (4-12)$$

$$FFN_C(C_{com} + C_{mha}) = C_{self} \in \mathbb{R}^{l \times d} \quad FFN_Q(Q_{com} + Q_{mha}) = Q_{self} \in \mathbb{R}^{m \times d} \quad (4-13)$$

最终编码阶段

最后,我们使用两个双向 LSTM 将所有信息进行汇总并将矩阵 C_{fin} 和 Q_{fin} 输入给解码器,如公式4-14。

$$BiLSTM_{finC}(C_{self}) = C_{fin} \in \mathbb{R}^{l \times d} \quad BiLSTM_{finQ}(Q_{self}) = Q_{fin} \in \mathbb{R}^{m \times d} \quad (4-14)$$

4.3.4 解码器

在编码器得到最终的表示 $C_{fin} \in \mathbb{R}^{l \times d}$ 和 $Q_{fin} \in \mathbb{R}^{m \times d}$ (即编码之后的内容信息和任务信息)后进入到解码器解码。在下文中,我们将对解码器的每个阶段进行进一步的说明。

结果表示阶段

在训练阶段,解码器首先要将结果 a 的表示投射到 d 维的空间上,如公式4-15

$$AW_2 = A_{proj} \in \mathbb{R}^{n \times d} \quad (4-15)$$

之后紧接的是一个自注意力层,该注意力层与编码器中的自注意力层相对应。由于该过程不包含循环和卷积操作,所以我们为 A_{proj} 加上了一个位置编码信息 x , 如公式4-16。

$$A_{proj} + PE = A_{ppr} \in \mathbb{R}^{n \times d}, \quad \text{其中 } PE[t, k] = \begin{cases} \sin(\frac{t}{10000^{\frac{k}{2d}}}) & k \text{ 为偶数} \\ \cos(\frac{t}{10000^{\frac{k-1}{2d}}}) & k \text{ 为奇数} \end{cases} \quad (4-16)$$

自注意力阶段接着我们使用了自注意力机制 [!! 引用!!] 从而使得解码器能够知道之前的输出(若之前没有输出则认为之前的输出为某个初始化的单词)和相关内容是什么,从而产生下一个输出。在多头注意力机制之后使用了一个残差前馈网络(FFN, 其定义在4.3.3节中的自注意力阶段给出),如公式4-17。

$$MultiHead_A(A_{ppr}, A_{ppr}, A_{ppr}) = A_{mha} \in \mathbb{R}^{n \times d} \quad (4-17)$$

$$MultiHead_AC((A_{mha} + A_{ppr}), C_{fin}, C_{fin}) = A_{ac} \in \mathbb{R}^{n \times d} \quad (4-18)$$

$$FFN_A(A_{ac} + A_{mha} + A_{ppr}) = A_{self} \in \mathbb{R}^{n \times d} \quad (4-19)$$

获得中间状态阶段

之后，我们使用了一个标准的带注意力机制的 LSTM 网络来对每个时刻 t 产生一个内容状态 \tilde{c}_t 。这个 LSTM 又将使用之前的结果序列中的单词 A_{self}^{t-1} 和内容状态来生成一个中间状态 h_t ，如公式4-20。

$$LSTM([(A_{self})_{t-1}; \tilde{c}_{t-1}], h_{t-1}) = h_t \in \mathbb{R}^d \quad (4-20)$$

任务与内容注意力阶段

上一阶段中的中间状态可用来获得注意力权重 α_t^C 以及 α_t^Q ，解码器从而可以获得 t 时刻的编码信息，如公式4-21。

$$\text{softmax}(C_{fin}(W_2 h_t)) = \alpha_t^C \in \mathbb{R}^l \quad \text{softmax}(Q_{fin}(W_3 h_t)) = \alpha_t^Q \in \mathbb{R}^m \quad (4-21)$$

获得内容与任务状态阶段

内容的表示将于这些权重相结合并输入给一个前馈网络从而形成内容状态与任务状态，激活函数为 \tanh ，如公式4-22。

$$\tanh(W_4[C_{fin}^\top \alpha_t^C; h_t]) = \tilde{c}_t \in \mathbb{R}^d \quad \tanh(W_5[Q_{fin}^\top \alpha_t^Q; h_t]) = \tilde{q}_t \in \mathbb{R}^d \quad (4-22)$$

多指针生成阶段

由于我们的模型必须要能够生成那些未出现在任务或内容序列中的单词，所以需要额外声明一个词汇表 v 。我们会从任务、内容和外部的词汇表中分别获得每个单词的概率分布，如公式4-23。

$$\sum_{i:c_i=w_t} (\alpha_t^C)_i = p_c(w_t) \in \mathbb{R}^n \quad (4-23)$$

$$\sum_{i:q_i=w_t} (\alpha_t^Q)_i = p_q(w_t) \in \mathbb{R}^m \quad (4-24)$$

$$\text{softmax}(W_v \tilde{c}_t) = p_v(w_t) \in \mathbb{R}^v \quad (4-25)$$

这些概率分布将会覆盖任务、内容和外部词汇表中所有单词的集合，其中却是一些条目将会被赋为 0 值从而使所有分布都在 \mathbb{R}^{l+m+v} 。最后，我们设置两个标量开关来调节每个分布的重要程度并确认最终输出的分布，如公式4-26。

$$\sigma(W_{pv}[\tilde{c}_t; h_t; (A_{self})_{t-1}]) = \gamma \in [0, 1] \quad (4-26)$$

$$\sigma(W_{cq}[\tilde{q}_t; h_t; (A_{self})_{t-1}]) = \lambda \in [0, 1] \quad (4-27)$$

$$\gamma p_v(w_t) + (1 - \gamma)[\lambda p_c(w_t) + (1 - \lambda)p_q(w_t)] = p(w_t) \in \mathbb{R}^{l+m+v} \quad (4-28)$$

需要说明的是，在训练的全时域中，我们采用的是单词级别的对数似然损失，如公式4-29。

$$\mathcal{L} = - \sum_t^T \log p(a_t) \quad (4-29)$$

4.4 实验与分析

4.4.1 多任务网络的实验结果

表 4-1 多任务网络的实验结果

Table 4-1 A Table with footnotes

数据集	单任务				多任务				
	S2S	w/SAtt	+CAtt	+QPtr	S2S	w/SAtt	+CAtt	+QPtr	+ACurr
SQuAD	48.2	68.2	74.6	75.5	47.5	66.8	71.8	70.8	74.3
IWSLT	25.0	23.3	26.0	25.5	14.2	13.6	9.0	16.1	13.7
CNN/DM	19.0	20.0	25.1	24.0	25.7	14.0	15.7	23.9	24.6
MNLI	67.5	68.5	34.7	72.8	60.9	69.0	70.4	70.5	69.2
SST	86.4	86.8	86.2	88.1	85.9	84.7	86.5	86.2	86.4
QA-SRL	63.5	67.8	74.8	75.2	68.7	75.1	76.1	75.8	77.6
QA-ZRE	20.0	19.9	16.6	15.6	28.5	31.7	28.5	28.0	34.7
WOZ	85.3	86.0	86.5	84.4	84.0	82.8	75.1	80.6	84.1
WikiSQL	60.0	72.4	72.3	72.6	45.8	64.8	62.9	62.0	58.7
MWSC	43.9	46.3	40.4	52.4	52.4	43.9	37.8	48.8	48.4
总得分	-	-	-	-	473.6	546.4	533.8	562.7	571.7

如图4-4以及4-3所示, 我们的模型的训练样本为包含任务序列、内容序列以及结果序列的三元组。我们所做的对比实验的基准为使用了指针生成器的序列到序列的模型(简称 S2S 模型) [!! 引用!!]。不一样的是, S2S 模型的输入为自然语言问题序列, 而多任务网络的输入为任务及内容序列的集合。

实验结果如表4.4所示, 表中 S2S 代表序列到序列(sequence to sequence)模型, (w/SAtt) 代表增加带自注意力机制的 S2S 模型, (+CAtt) 代表将任务与内容序列拆分为两个输入序列并增加对偶协同注意力机制, (+QPtr) 代表增加任务指针器, (+ACurr) 代表增加递归学习过程。总得分表示多任务学习时各个子任务得分的和, 由于单任务不存在多任务的得分和所以标记为'-'。

在表4.4中的指标可以看出, S2S 模型在 SQuAD 任务上的效果较差, 在 WikiSQL 数据集上的得分比之前的序列到序列的基准方案要高, 但是普遍都没有多任务网络(+QPtr)的效果优异。在 S2S 模型的基础上增加了带自注意力机制层(w/SAtt)的编码器和解码器之后 [!! 引用!!], 模型具备了整合来自任务和内容的信息的能力。从表中可以看出, 新的模型在 SQuAD 任务上提升了 20 nF1, QA-SRL 任务上提升了 4 nF1, WikiSQL 上提升了 12 LFEM。模型在 WikiSQL 上的表现接近了最高水平的范围, 在没有使用结构化的方法 [!! 引用!!] 下达到了 72.4%。

接着, 我们尝试将任务和内容从输入中进行拆分并增加了对偶协同注意力机制(+CAtt)。可以看到, 模型在 SQuAD 和 QA-SRL 上的性能提升了超过 5 nF1, 但是在其他任务上没有明显的提升, 甚至在 MNLI 和 MWSC 任务上的性能有所降低。其原因可能是由于两个任务往往能够从输入中直接复制一些单词作为输出, S2S 模型的输入为任务和内容的集合, 所以其中的指针生成器可以直接完成从输入中复制信息到输出的操作。而将任务和内容信息拆分为两个输入序列的时候, 模型就失去了这个能力。

为了解决这个问题, 我们在之前的模型上又增加了一个任务指针(+QPtr)进行实验。实验结果显示, 任务指针有效的提高了模型在 MNLI 和 MWSC 上的性能。同时, 模型在 SQuAD 任务上提升到了 75.5 nF1, 达到了 SQuAD 领域中使用直接跨度检测方法的水平, 也就是说多任务网络达到了不明确将问题进行跨度检测的方法中的最高水平。

最后的模型在 WikiSQL 任务上达到了 72.4% 的 LFEM 以及 80.4% 的数据库执行准确率, 超过了之前的最高水平 [!! 引用!!] 的 71.7% 和 78.5%。

在单任务与多任务的比较中, 我们也得到了类似的结果以及注意到了一些额外的特点。多任务与单任务模型相比, 在 QA-ZRE 任务上的 F1 得分又有 11 点提升, 也验证了多任务的模型在解决零样本学习(zero-shot learning)上的能力。

4.4.2 不同优化策略下的实验结果

在多任务的训练过程中我们尝试了各种批量采样 (batch-level sampling) 的策略, 包括联合训练、递进学习、反递进学习等, 其实验结果如表4-2所示。

我们首先考虑的是完全联合 (fully joint) 的策略。从训练开始到结束, 我们对所有的任务按照固定的顺序循环的进行批量采样。这种策略在迭代 (iteration) 次数较少的单任务下的表现良好, 见表4-2, 但在迭代次数较多的单任务下性能不佳。经过总结我们发现, 单任务与多任务场景下的性能的差异与单任务的迭代次数之间有很强的关联。

由于上述原因, 我们还尝试了一些反递进学习的策略 [!! 引用!!]。这些训练的策略都是由两阶段组成的: 1) 在第一阶段中对部分任务进行联合训练 (往往选择较难的任务)。2) 第二阶段根据完全联合策略进行对所有任务进行训练。

首先, 我们尝试了先将 SQuAD 任务进行单独训练, 之后再对其他的所有任务进行完全联合训练。由于我们的多任务网络对于所有任务来说是一种基于问答的方法, 所以我们联想到是否可以先与训练 SQuAD 这样的 QA 数据集然后再对其他的类 QA 的任务进行训练。这种方法可以使得解码器先学会如何从内容信息中检索出有用信息, 然后学习不同的任务以及如何产生属于不同任务领域的单词。在 SQuAD 这个任务场景下进行预训练已经被证明可以提高 NLI 的性能 [!! 引用!!]。根据我们的实验来看, 这种方式相对于其他的训练策略而言是一种更有效的训练策略, 在总得分上获得了 571.7 的最高分。需要特别提到的是, 这种策略在 IWSLT 任务上的得分较低, 但是在其他的任务中会有更大的提升, 特别是使用指针的任务。

为了探索在递进学习策略的初始状态下增加额外的任务是否会进一步提高性能, 我们还尝试了将 IWSLT、CNN/DM 任务和 IWSLT、CNN/DM、MNLI 任务加入到第一阶段进行训练。这些任务包含了与其他任务非常相关的训练样本, 并且也包含了最长的结果序列。除此之外, 这些任务具有很高的多样性, 应该能够让解码器学会更多的解码方式, 例如针对 IWSLT 的词汇表、针对 SQuAD 与 CNN/DM 的内容指针以及针对 MNLI 的任务指针。然而, 实验结果表明这些新增的任务并没有提升模型的性能。在将 SQuAD, IWSLT, CNN/DM 和 MNLI 加入到递进学习的初始状态下时, 其他的一些任务 (例如 QA-SRL, WikiSQL 和 MWSC) 的性能明显下降。

另外, 我们也尝试了在递进学习策略的初始状态下使用了 SST, QA-SRL, QA-ZRE, WOZ, WikiSQL 和 MWSC, 这也就意味着先选择一些最简单的任务并进行训练。实验证明, 这个策略是一个非常低劣的策略, 不仅在初始递进学习中的任务的性能很差, 在 SQuAD 和 IWSLT 上的表现更差。

事实证明, 对于我们的任务而言, 使用反递进学习策略的效果优于递进学习策略, 也就是说明了先学习简易的任务未能使模型学到能应用于其他任务的表示, 甚至使得模

型想要学习到其他任务有用的内部表示变得更加地困难。除此之外，实验的结果也说明了多任务学习的挑战。通过改变所学习的任务的先后次序可能可以提高某些任务性能，但也很有可能使其他的任务的性能下降。这也是多任务学习中一个非常值得研究的任务。

表 4-2 多任务网络在不同训练策略下的实验结果

Table 4-2 A Table with footnotes

数据集	完全联合	递进	反递进		
			SQuAD	+IWSLT+CNN/DM	+MNLI
SQuAD	70.8	43.4	74.3	74.5	74.6
IWSLT	16.1	4.3	13.7	18.7	19.0
CNN/DM	23.9	21.3	24.6	20.8	21.6
MNLI	70.5	58.9	69.2	69.6	72.7
SST	86.2	84.5	86.4	83.6	86.8
QA-SRL	75.8	70.6	77.6	77.5	75.1
QA-ZRE	28.0	24.6	34.7	30.1	37.7
WOZ	80.6	81.9	84.1	81.7	85.6
WikiSQL	62.0	68.6	58.7	54.8	42.6
MWSC	48.8	41.5	48.4	34.9	41.5
总得分	562.7	499.6	571.7	546.2	557.2

4.4.3 实验分析

多指针生成器与任务识别

多任务网络会从三个方面进行选择：从词汇表生成词汇、任务序列上的指针以及内容序列上的指针。尽管模型没有明确的监督信号来训练这些选择，但网络仍可以学会如何在这三个选择之间进行切换。图 xx 显示了我们最好的模型选择每个选项的频率是怎样的。可以看出，对于 SQuAD，QA-SRL 和 WikiSQL 任务，模型主要是从内容信息中进行选择和复制，这正是因为这些任务中的正确结果主要囊括在内容中。由于答案的摘要主要由内容中的单词组成，模型在 CNN/DM 任务中也是主要从内容中进行复制。对于 SST，MNLI 和 MWSC，由于任务序列中会包含需要的类别，所以模型也偏向选择任务指针。对于 IWSLT 和 WOZ，模型更喜欢选择从词汇表生成单词，因为中文的单词和

对话的状态字段很少在对应的内容信息中。

同时，使用了 TCR 模板之后的所有任务之间不会使得模型产生混淆。例如：中文单词仅仅会在中文翻译成英文的过程中出现，而在情感分析任务中也从未输出过非“正面”或“负面”的情感类别。

模型的拓展性

多任务网络具有很好的拓展性，除了中文翻译为英文任务和英文生成 SQL 自然语言任务外对图4.4中的其他任务都有很好的表现。在这十个任务共同学习的模型的基础上进行其他任务的学习更加表现了模型良好的拓展性。对于两个新的任务，英文到捷克语的翻译任务以及命名实体识别（NER）上，使用训练好的模型进行微调可以在更少的迭代次数下达到模型的收敛，达到很好的效果，如图 xx。其中，英文到捷克语的翻译任务使用的是 En->Cs 数据集，NER 任务使用的是 OntoNotes 5.0 数据集 [!! 引用!!]。

文本分类的零样本学习

由于 MNLI 包含在是个任务中，因此模型也可以适应斯坦福自然语言推理语料库（SNLI） [!! 引用!!]。在预训练的多任务网络之上进行微调可以达到 87% 的精确匹配分数，比随机初始化再训练提升了 2 分，也比现有的最高水平提升了 2 分 [!! 引用!!]。更值得注意的是，在没有对 SNLI 进行任何微调的情况下，在原模型上预训练的多任务网络仍然达到了 62% 的精确匹配分数。因为十个任务包含 SST 任务，所以它也可以在其他二元情感分类任务上表现良好。在亚马逊和 Yelp 评论数据集 [!! 引用!!] 上，预训练的多任务网络分别达到了 82.1% 和 80.8% 的精确匹配分数并且不需要对模型进行任何微调。这些结果表明在多任务网络模型的基础上进行微调可以是的模型拓展到域外的问题上，甚至可以适应更广的文本分类问题上。

4.5 本章小结

第五章 总结与展望

5.1 本文工作小结

5.2 展望

参考文献

- [1] ANDROUTSOPOULOS I, RITCHIE G D, THANISCH P. Natural Language Interfaces to Databases - An Introduction[J]. Natural Language Engineering, 1995, 1(1): 29–81.

致 谢

感谢所有测试和使用交大学位论文 L^AT_EX 模板的同学！

感谢那位最先制作出博士学位论文 L^AT_EX 模板的交大物理系同学！

感谢 William Wang 同学对模板移植做出的巨大贡献！

感谢 @weijianwen 学长一直以来的开发和维护工作！

感谢 @sjtug 以及 @dyweb 对 0.9.5 之后版本的开发和维护工作！

感谢所有为模板贡献过代码的同学们, 以及所有测试和使用模板的各位同学！

攻读学位期间发表的学术论文

- [1] CHEN H, CHAN C T. Acoustic cloaking in three dimensions using acoustic metamaterials[J]. Applied Physics Letters, 2007, 91:183518.
- [2] CHEN H, WU B I, ZHANG B, et al. Electromagnetic Wave Interactions with a Metamaterial Cloak[J]. Physical Review Letters, 2007, 99(6):63903.

攻读学位期间参与的项目

- [1] 973 项目 “XXX”
- [2] 自然科学基金项目 “XXX”
- [3] 国防项目 “XXX”

简 历

基本情况

某某，yyyy 年 mm 月生于 xxxx。

教育背景

yyyy 年 mm 月至今，上海交通大学，博士研究生，xx 专业

yyyy 年 mm 月至 yyyy 年 mm 月，上海交通大学，硕士研究生，xx 专业

yyyy 年 mm 月至 yyyy 年 mm 月，上海交通大学，本科，xx 专业

研究兴趣

L^AT_EX 排版

联系方式

地址：上海市闵行区东川路 800 号，200240

E-mail: xxx@sjtu.edu.cn