软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, 2022,33(12):4727–4745 [doi: 10.13328/j.cnki.jos.006686]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

基于树状模型的复杂自然语言查询转 SQL 技术研究*

赵 猛[1,2], 陈 珂[1,2], 寿黎但[1,2,3], 伍 赛[1,2], 陈 刚[1,2]

[1](浙江大学 计算机科学与技术学院, 浙江 杭州 310027)

[2](浙江省大数据智能计算重点实验室(浙江大学), 浙江 杭州 310027)

[3](浙江大学 计算机辅助设计与图形学国家重点实验室, 浙江 杭州 310027)

通信作者: 陈珂, E-mail: chenk@zju.edu.cn

摘 要: 自然语言查询转 SQL(NL2SQL)是指将自然语言表达的查询文本自动转化成数据库系统可以理解并执行的结构化查询语言 SQL 表达式的技术. NL2SQL 可以为普通用户提供数据库查询访问的自然交互界面, 从而实现基于数据库的自然问答. 复杂查询的 NL2SQL 是当前数据库学术界的研究热点, 主流方法采用序列到序列(Seq2seq)的编解码方式对问题进行建模. 然而, 已有的工作大多基于英文场景, 面向中文领域实际应用时, 中文特殊的口语化表达导致复杂查询转化困难; 此外, 现有工作难以正确输出包含复杂计算表达式的查询子句. 针对上述问题, 提出一种树状模型取代序列表示, 将复杂查询自顶向下分解为多叉树, 树结点代表 SQL 的各组成元素, 采用深度优先搜索来预测生成 SQL 语句. 在 DuSQL 中文 NL2SQL 竞赛的两个官方测试集中, 该方法分别取得了第 1 名和第 2 名的成绩, 验证了其有效性.

关键词: 自然语言查询转 SQL; 语义解析; 自然语言处理

中图法分类号: TP311

中文引用格式: 赵猛, 陈珂, 寿黎但, 伍赛, 陈刚. 基于树状模型的复杂自然语言查询转 SQL 技术研究. 软件学报, 2022, 33(12): 4727–4745. http://www.jos.org.cn/1000-9825/6686.htm

英文引用格式: Zhao M, Chen K, Shou LD, Wu S, Chen G. Converting Complex Natural Language Query to SQL Based on Tree Representation Model. Ruan Jian Xue Bao/Journal of Software, 2022, 33(12): 4727–4745 (in Chinese). http://www.jos.org.cn/1000-9825/6686.htm

Converting Complex Natural Language Query to SQL Based on Tree Representation Model

ZHAO Meng[1,2], CHEN Ke[1,2], SHOU Li-Dan[1,2,3], WU Sai[1,2], CHEN Gang[1,2]

[1](School of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China)

[2](Key Laboratory of Big Data Intelligent Computing of Zhejiang Province (Zhejiang University), Hangzhou 310027, China)

[3](State Key Laboratory of CAD&CG, Zhejiang University, Hangzhou 310027, China)

Abstract: NL2SQL refers to a technology that automatically converts query expressed in natural language into a structured SQL expression, which can be parsed and executed by the DBMS. NL2SQL can provide ordinary users with a natural interactive interface for database query access, thereby realizing question-answering atop database systems. NL2SQL for complex queries is now a research hotspot in the database community. The most prevalent approach uses the sequence-to-sequence (Seq2seq) encoder and decoder to convert complex natural language to SQL. However, most of the existing work focuses on English language. This approach is not ready to address the special colloquial expressions in Chinese queries. In addition, the existing work cannot correctly output query clauses containing complex calculation expressions. To solve the above problems, this study proposes to use a tree model instead of the sequence representation. The proposed approach disassembles complex queries from top to down to comprise a multi-way tree, where the tree nodes represent the elements of SQL. It uses a depth-first search to predict and generate SQL statements. The proposed approach has achieved the championship and 1st runner-up in two official tests of DuSQL Chinese NL2SQL Competition. The experimental results confirm the

* 基金项目: 浙江省重点研发计划(2021C01009); 国家自然科学基金(62050099); 高校基本科研业务费专项

收稿时间: 2021-01-27; 修改时间: 2021-08-04, 2021-12-15; 采用时间: 2022-03-21; jos 在线出版时间: 2022-05-24

effectiveness of the proposed approach.

Key words: NL2SQL; semantic parsing; natural language processing

大数据时代, 大多数海量数据存储在数据库中. 日常生活中, 用户频繁使用网购、网银、社交活动等服务与数据库进行交互. 传统的实现方式是使用预先封装好的 SQL 模版来完成对数据库的查询操作, 这样的方式灵活性较差, 当用户有新的服务需求, 往往需要技术工程师再次撰写和封装 SQL 语句, 使得使用数据库的门槛提高. 自然语言查询转 SQL 是指将人类口头表达的查询自动转化成数据库可以理解并执行的结构化查询语言 SQL 的表达式. 该技术可以有效打通普通用户和数据库的壁垒, 缩短普通用户和数据库之间的距离,从而实现基于数据库的自动问答能力, 这个问题一直是数据库领域的经典难题.

近年来, 随着大数据和新一代人工智能的发展, 自然语言查询转 SQL 技术已经成为人工智能、人机交互和数据库领域交叉的热点问题, 受到了学术界和工业界的广泛关注. 2017 年, 国外机构和学者们率先发布了大规模跨领域的 WikiSQL[1]数据集, 随之出现了很多经典的工作, 如: TypeSQL[2]模型获得了 68%的 SQL 完全匹配准确率; X-SQL[3]更是将 SQL 执行准确率提升到了 91.8%, 超出了人类获取答案能力水平近 4 个百分点. 不过, 考虑到 WikiSQL 中的 SQL 查询语句较为简单, 问题对应数据库也只有一个表格, 不能满足实际需求, 2018 年末, 耶鲁大学发布了多表复杂查询数据集 Spider[4], 整体难度大幅提升. 后来也涌现了一些优秀的模型, 如 IRNet[5], RAT-SQL[6], 都采用了 SQL 语法树的思想来解码生成复杂的 SQL 语句, 也是将准确率大幅度提升到了 60+%; 在 WikiSQL 单表简单查询任务上, 模型 SQL 的解析水平已经超过人类, 多表复杂查询转 SQL 领域仍存在很大挑战, 当前主流模型的性能离实际落地还有些距离. NL2SQL 在主流英文领域的研究进展吸引着国内学术界和工业界的目光, 通过竞赛发布中文特色数据集 TableQA[7], DuSQL[8]以及翻译英文数据集 CSpider[9]大力推动 NL2SQL 技术在中文场景下的发展以及落地.

然而, 上述的优秀工作和成果极大推动了大数据人工智能的发展, 但在其实际应用落地时还面临着巨大的挑战. 中文的自然语言相比于英文本身就更灵活多变, 并且自然语言查询转 SQL 针对的是非正式的用户口语表达, 机器也就更加难以理解; 同时, 口语化的复杂查询也使得 SQL 语句生成难度陡增, 所以当前学术界主流的英文场景下的模型在中文应用中性能有所限制. 在商业智能场景下, 存在许多涉及 SQL 计算的查询, 现有的 NL2SQL 研究工作很少有针对这样的查询形式展开模型优化. 在 NL2SQL 技术实际落地中, 还有可能会遇到数据库规模较大导致信息无法完全输入到模型, 以及新的数据库结构发生变化导致性能下降的问题, 这也是需要考虑和解决的.

本文将解决复杂多变的中文口语查询转 SQL 问题作为核心研究点, 针对复杂自然语言查询 SQL 语句自顶向下分解为多叉树形式, 提出了一种可生长的树状模型, 解决了商业智能等场景中 SQL 计算查询等新查询类型的支持; 针对跨领域查询数据库结构变化导致性能下降的问题, 提出了一种表信息增强算法来提高模型的稳定性; 针对低资源条件下模型无法处理任意规模数据库的问题, 提出了一种对数据库表进行预筛选的模型; 针对复杂查询中的值抽取难以得到充分训练的问题, 提出了独立的支持嵌套 SQL 的统一值抽取模型. 最终搭建了一套基于树状模型的复杂自然语言查询转 SQL 系统框架, 并在中文 DuSQL 数据集上取得了优异的成绩, 证明了解决方案的有效性, 具有一定的学术价值和现实意义.

本文的贡献点可以概括为如下几点:

(1) 提出了一种针对多表跨领域查询的预筛选模型和表格数据增强算法, 解决了实际应用中低资源条件下模型无法处理任意规模数据库以及跨领域查询时数据库结构变化导致性能下降的问题. 通过预先筛选相关数据表格和训练过程的表数据增强, 使得 NL2SQL 模型能够在低资源跨领域场景下更加稳定, 性能表现及泛化能力更强;

(2) 设计了一种针对复杂查询转 SQL 的树状模型, 解决了模型将复杂口语查询转化为 SQL 语句的问题; 同时, 通过升级模型适配了 SQL 计算的需求. 本文将复杂 SQL 语句基于自顶向下的思想分解为多叉树的形式, 各子句结点通过复用模块有效组织起来, 同步输出 SQL 结果. 同时, 对模型简单扩展解决了面向商业智

能场景下的 SQL 计算等问题;

(3) 提出了一种针对嵌套 SQL 的值抽取模型和 SQL 解析算法, 解决了树状模型无法有效训练值抽取模块和复杂 SQL 嵌套时同列抽取不同值的问题. 将值抽取从树状模型中分离出来, 整合统一了不同子句, 更好地利用训练数据提升了值抽取模型表现. 针对嵌套 SQL 同列抽取不同值的场景, 给出了训练不同 SQL 模块值抽取功能的有效解决方案;

(4) 实现了基于树状模型的复杂自然语言查询转 SQL 系统框架, 并在中文 DuSQL 数据集上取得较好的性能表现, 在不同测试集中分别排名第 1 和第 2.

本文第 1 节介绍自然语言转查询任务的相关工作. 第 2 节对本文研究的问题进行描述, 提出面向复杂自然语言查询的 SQL 树状分解模型, 并系统性地给出基于该模型的整个技术框架. 第 3 节给出技术框架中层层递进的 3 个模型的具体实现方法. 第 4 节设计实验对本文提出的方法进行验证和分析. 第 5 节对本文工作进行小结, 并对未来工作进行展望.

1 相关工作

语义解析是一门将人类口语化的自然语言转化为标准的机器可理解执行的有意义的逻辑表达式的技术, 是人机交互领域重要的研究方向. 自然语言查询转 SQL 任务正是将自然语言转化为数据库查询语言 SQL 的逻辑表达式, 可以看作是对于数据库的自然语言接口, 帮助非专业用户与储存着大量数据信息的机器数据库进行交互, 机器返回用户想要查询的内容结果, 所以也可以看作是针对数据库的自动问答系统. 本节将对 NL2SQL 的一些相关数据集和相关研究进展进行介绍.

1.1 相关数据集

早期的语义解析数据集关注于单领域固定多表数据库的查询如 ATIS[10], GeoQuery[11]等, 这些数据集规模较小. 2015 年, Pasupat 等人[12]从维基百科(Wikipedia)的半结构化数据表格中收集了大量的问题和对应的答案, 构建了 WikiTableQuestions 数据集. 来到 2017 年, Zhong 等人[1]和 salesforce 机构发布了第 1 个大规模跨领域的语义解析 NL2SQL 数据集 WikiSQL, 包含 24 241 个各种领域的数据表格和 80 654 对自然语言问题以及对应的 SQL 逻辑表达式. 大规模的数据集极大了推动了 NL2SQL 领域技术的研究, 受到学者们的广泛关注, 提出了各种各样的模型, 如 SQLNet[13], TypeSQL[2], SQLova[14], X-SQL[3]等. 不过, WikiSQL 是一个单表简单查询数据集, 查询问题对应一个单独的表格, 且 SQL 语句只包含“select”“where”这样的关键字. 然而在实际应用中, 数据库大多包含多个表格, 并且 SQL 语句也会包含“having”“groupby”“orderby”这样的关键字, 同时还可能出现 SQL 嵌套的情况. 针对这些问题, Yu 等人在 2018 年发布了一个多表复杂查询数据集 Spider[4], 包含有 10 181 个查询问题和 5 693 个唯一的复杂 SQL 语句, 每个数据库内有多个表格, 并且横跨了 138 个领域.由于多表复杂查询转 SQL 的难度较高, Spider 数据集的提出给学术界带来了挑战, 在 WikiSQL 上表现优秀获得 68.0% SQL 匹配准确率的 TypeSQL 模型, 在 Spider 数据集上随机切分样本情况下测试集准确率为 33.0%,在跨领域查询随机切分数据库情况下, 测试集准确率仅为 8.2%, 这体现了跨领域查询的难度, 本文也正是针对跨领域查询所面临的问题给出了优化方案.

NL2SQL 不仅受到国外学术界和工业界的极大关注, 近年来也吸引着国内研究学者和工业界的目光, 2019 年, 阿里云天池大数据竞赛发布第 1 个中文 NL2SQL 数据集 TableQA[7]. 该数据集是一个单表简单查询数据集, 包含有 49 974 条数据; 随之, Min 等人对 Spider 数据集进行中文翻译, 发布了 CSpider[9]中文数据集; 2020 年, 由中国中文信息协会、中国计算机协会和百度公司联合举办的 2020 语言与智能技术竞赛上发布了中文多表复杂查询数据集 DuSQL, 针对 NL2SQL 在商业智能领域落地会遇到的一些复杂计算需求, 增加了多种 SQL 计算任务, 这也提高了复杂查询转 SQL 的难度.

1.2 单表简单查询

自然语言查询转 SQL 任务可以根据预测 SQL 的过程不同分成两种方法——序列到序列 Sequence-to-

Sequence 和槽填充 Slotfilling. 其中, Sequence-to-Sequence 的方法就是在解码阶段一步一步地完成 SQL 语句的生成, 早期的工作将注意力、Copy 机制加入到 Seq2Seq 模型中, 后续 Seq2SQL[1], Pointer-SQL[15]采用了指针网络 Pointer Network 对解码阶段生成的空间加以限制; 而 Slotfilling 的方法则是根据 SQL 语法的特性, 将 SQL 生成划分为多个子任务设计模板槽位, 然后让模型去预测这些输出, 最终整合为 SQL 语句. SQLNet[13]将 SQL 预测分解为多个子任务分别去训练模型来预测. TypeSQL[2]则简化了 SQLNet 的预测过程, 并且给模型融入了数据表类型(Type)信息. SQLova[14]则是率先将预训练语言模型引入了 NL2SQL 任务当中, 证明了预训练模型 BERT 对于跨领域表格理解的有效性. 2019 年, SOTA 模型 X-SQL[3]采用多任务的预训练语言模型 MT-DNN[16]作为编码器, 并且在对表格列名的编码理解方面提出将 BERT 的 CLS 标记替换为新的标记 CTX 来更好地增强上下文信息, 并且利用 CTX 标记得到的上下文编码与列名编码做 Attention 来获取列名编码的降维输出, 在 WikiSQL 上取得了 91.8%的 SQL 执行获取正确答案的准确率, 大幅超出人类对于表格问答的水平.

1.3 多表复杂查询

虽然在单表简单查询中模型的表现已经超出人类的水平, 但是在多表复杂查询的场景 Spider 数据集下, NL2SQL 的难度更是达到了一个新的层次. Spider 数据集将 SQL 按照所含关键字、表格的个数以及 SQL 之间的嵌套情况分成了 4 个等级——Easy, Medium, Hard, Extra Hard. 作者使用 SQLNet, TypeSQL, 在不考虑跨领域数据库的情况下进行实验, 在 Easy 难度的测试集中获得了 40%左右的 SQL 形式匹配准确率, 然而在 Extra Hard 复杂查询的测试集样本中仅仅分别取得了 3.3%和 14.4%的准确率, 甚至在跨领域查询的测试集下, TypeSQL 的准确率仅仅为 0.8%. SyntaxSQLNet[17]利用 SQL 构建语法树定义一种复杂的模板形式, 将复杂 SQL 的解码分成了 9 个模块分别训练模型, 最终递归地生成复杂的 SQL, 在测试集上获得了 27.2%的准确率.

经典模型 IRNet[5]提出对 SQL 语法制定了规约文法, 将 SQL 转化为中间表达形式 SemQL, 然后在解码阶段, 利用基于语法的 LSTM 解码器, 根据定义好的符号规则去生成 SemQL, 最终再转为 SQL 语句, 加上 BERT 模型后, 取得了 2019 年的 SOTA 效果 54.7%. IRNet 中也提到了如何对自然语言查询和数据库模型做 Schema Linking, 通过事先对查询问题和数据库表名列名做匹配打类型标记来给模型输入额外的信息, 更好地理解输入的查询和数据库模式, 从而提高生成 SQL 的效果.

而当前, 在 Spider 数据集上的 SOTA 模型 RAT-SQL[6]也是基于 IRNet 的结构, 主要在 Schema Linking 方面进行改进, 更多考虑了数据库表格之间关系的信息, 提出了 RAT-Layer 网络来更好地编码数据库模式的特征信息. 由于值抽取对于复杂查询转 SQL 较为困难, 上述模型均没有值抽取功能. 最近的一些工作如 BRIDGE[18]做了含值的 SQL 生成任务, 在 Spider 上获得 59.9%的分数, 低于 RAT-SQL 不含值的成绩.

2 问题描述与技术框架

2.1 问题描述

自然语言查询转 SQL(NL2SQL)定义为给定自然语言口语查询 Q 和与其相对应的数据库模式 S, 返回正确可执行的数据库查询语言 SQL 语句 SQL:

$$(Q,S)\rightarrow SQL \tag{1}$$

其中,

输入 $Q=\{W_1,W_2,\ldots,W_i,\ldots,W_n\}$代表自然语言口语查询, Q 总共有 n 个字符, W_i 代表着其中第 i 个字符;

输入 $S=\{T_1,T_2,\ldots,T_i,\ldots,T_n,E\}$代表数据库模式, 由基本的数据库模式对象组成, 各个数据列、数据类型构成了数据表 T, 表之间主外键关系构成了表连接 E;

输出 $SQL=\{K,T,C,V,OP\}$代表用于访问和处理数据库的标准的计算机语言结构化查询语言, K 代表 SQL 语句的关键字, 如“select”“where”“groupby”等; T 和 C 分别代表 SQL 语句中涉及的数据表以及数据表的列名字段; V 代表 SQL 语句中出现的值, 其与对应字段的列数据类型有关; OP 则代表 SQL 语句中的一些数据操作符.

表 1 给出了在中文 DuSQL 数据集上的一些自然语言转 SQL 的样例.

表 1 DuSQL 数据集上的一些自然语言转 SQL 的样例

样例编号	*Q*	*SQL*
1	在 2017 年美国发生的枪击案中，德克萨斯受影响的学生占所有受影响学生总数的比例是多少	select a. 受影响学生数量/b. 受影响学生数量 from (select 受影响学生数量 from 美国近几年校园枪击事件 where 年份==2017 and 发生城市=='德克萨斯') a, (select sum (受影响学生数量) from 美国近几年校园枪击事件 where 年份==2017) b
2	哪些高校拥有的学生总数不低于所有高校的平均值	select 名称 from 高校 where 本科生数量+研究生数量 ≥(select avg (本科生数量+研究生数量) from 高校)
3	成立时间不到 14 年且年营业额超过 2000 万的公司有哪些	select 名称 from 公司 where TIME_NOW-成立时间<14 and 年营业额>20000000

自然语言查询转 SQL 任务可以根据给定的数据库的形式将其划分为单表查询和多表查询. 输入只有一个数据表的情况下为单表查询, 仅仅涉及到数据表列名的选择, 不需要考虑相关表的选择以及表连接等问题. 而在实际应用场景中, 数据库往往会包含众多数量的表, 用户想要获得的查询结果也往往需要连接多个相关表执行查询才能获取结果, 这对该任务提出了更多的难题和挑战, 因此本文的研究针对多表查询场景. 而在多表自然语言查询转 SQL 任务中, 数据集 Spider[4]还根据问题生成的 SQL 语句的逻辑形式复杂程度将该任务划分成 4 个难度等级: 简单、中等、困难、极难. 其中: 简单难度的查询主要是单或多条件匹配, 结果只包含“select”子句和“where”子句的预测, “where”子句由多个条件三元组(列名,操作符,条件值)通过“and”和“or”这样的连接符组成; 随着难度的提升, 不仅查询涉及的表数量增加, 生成的 SQL 语句关键字数量也不断增加, 例如扩展到聚合、分组、排序等查询, 甚至在极难难度中包含嵌套子查询等逻辑形式. 而中文数据集 DuSQL 在 Spider 已有难度的基础上, 针对实际应用中面向商业智能和购物等相关业务咨询的场景, 额外扩展了 3 种不同形式的 SQL 计算任务, 进一步提高了查询的复杂性. 表 1 中的 3 条样例分别代表了行计算查询、列计算查询和常数计算查询. 在实际的自然语言查询转 SQL 落地中, 假如有一个手机业务咨询场景, 简单查询只能回答“3000 元以下的手机有哪些”这种问题, 覆盖面较窄, 而对于“华为手机最贵的是哪一款, 它的价格是多少? 配置如何?”“华为 Mate 40 Pro 这款产品 256 G 内存版本比 128G 版本贵了多少”这些用户更可能关注的具体问题, 需要一个更全面更智能解决这些复杂查询的技术框架来实现. 本文致力于解决包含上述各类场景的宽泛的复杂自然语言查询转 SQL 问题.

2.2 基于树状模型的复杂自然语言查询转SQL技术框架

2.2.1 复杂自然语言查询的 SQL 树状分解模型

在复杂自然语言查询转 SQL 任务中, 与处理技术最直接相关的是结果 SQL 语句的结构复杂度. 本文对 DuSQL 数据集上出现的 SQL 形式进行了归纳和分解, 创新性地提出了用于 SQL 结构预测的树状统一模型, 如图 1 所示. 图 1 所示的 SQL 树状分解模型采用自顶向下的形式.

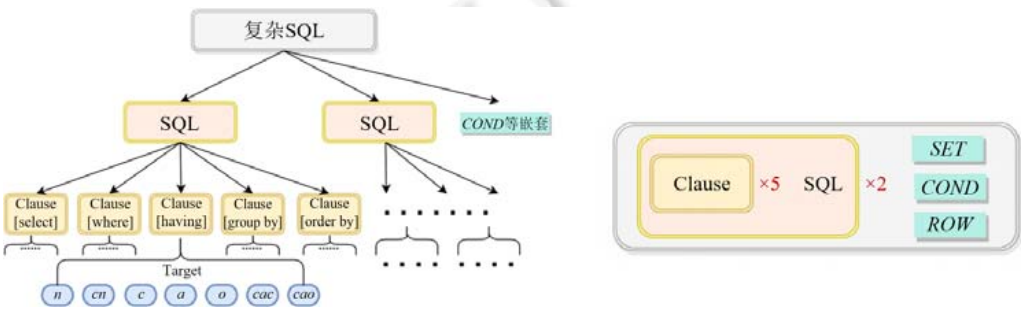


图 1 复杂 SQL 树状分解模型

第 1 层为最顶端的 SQL 嵌套层. 由于复杂自然语言查询包含了 SQL 之间互相嵌套的形式, 需要将它们划分出主 SQL 和副 SQL 语句这两个 SQL 模块. 本文对于 DuSQL 数据集上出现的嵌套 SQL 形式进行了总结, 在 SQL 嵌套层得出了 3 种嵌套情况 SET, COND 和 ROW, 见表 2. 其中, SET 为集合运算嵌套, 即两个 SQL

语句通过“union”“except”“intersect”等关键字进行嵌套, 组成“主 SQL 集合运算副 SQL”的形式; COND 为条件操作嵌套, 在主 SQL 语句的“where”和“having”子句中, 条件值来自 SQL 子句, 组成“主 SQL where/having COLOP 副 SQL”的形式; ROW 为行计算嵌套, 其结果 SQL 中包含 3 个“select”关键字, 但是第 1 个“select”只是起到了对两个操作列执行行计算返回查询的作用,这两个操作列可以作为主 SQL 语句和副 SQL 语句, 组成“select COL_a OPCOL_b (主 SQLCOL_a, 副 SQL COL_b)”的形式;

第 2 层为单 SQL 层. 对于每一个 SQL 语句, 仍可以向下分解成多个组件. 主要是由不同的 SQL 关键字构成, 如“select”“from”“where”“having”“groupby”“orderby”, 每个关键字都带有一个子句(clause 模块), 如 where 子句“where 年份=2017”. 其中, “from”子句表示当前查询来自哪些表以及多表之间如何链接, 该子句的形式主要通过组合 SQL 预测的表列信息, 结合主外键来得到. 对其余的子句来说, 它们具有很强的共性, 可以进一步向下分解;

第 3 层为 Clause 层. 不同关键字的子句可以归纳为相似的结构, 比如所有的子句都包含列名, “where”“having”子句中还包含数据操作符(OP), “select”“orderby”“having”子句中还包含聚合操作符(AGG). 对这些 Clause 的基本组成进行归纳统一后, 在下一层形成 SQL 的最细粒度预测目标对象(target 模块). 由于不同的 Clause 有着不同的结构, 在 SQL 结构预测时, 模型可以通过参数控制选择 Target 的一个子集进行预测;

第 4 层 Target 层. Target 层的核心就是实现了模型预测 SQL 结构中每个模块下的输出目标, 主要是对不同的 Clause 进行整合, 定义一些基本组成成分形成 Target 集合, 分别为 *n*, *cn*, *c*, *a*, *o*, *cao*, *cac*.

n 表示当前 Clause 中列的个数, 为 0 时代表当前 Clause 不存在, 在“orderby”子句中表示排序的方向, 取值为{“none”,“asc”,“desc”};

cn 表示条件之间的连接符, 在“where”和“having”子句中取值为{“none”,“and”,“or”}, 在“orderby”子句中表示是否取最值, 从而控制生成“limit 1”子句;

c 表示当前 Clause 涉及的列, 会在 SQL 结构预测模型中对输入列特征序列进行打分判断, 并根据 *n* 的预测得到 TOP *n* 个答案列;

a 表示聚合操作(AGG), 取值为{“none”,“max”,“min”,“count”,“sum”,“avg”};

o 表示条件操作(OP), 取值为{“not in”,“in”,“==”,“>”,“<”,“≥”,“≤”,“!=”,“like”};

cao 和 *cac* 用于解决 SQL 列计算任务, 其中: *cao* 表示列计算使用的操作符(OP), 取值为{“none”, “–”,“+”,“*”,“/”}; *cac* 表示列计算涉及的两个操作列, 同 *c* 的预测相似但互不影响.

表 2 DuSQL 数据集不同的嵌套形式

计算类型	Query	SQL	SQL(分离)
SET	哪些球队从未获得过欧冠冠军	(select 词条 id from 球队) except (select 冠军球队 id from 欧冠冠亚军)	主 SQL: select 词条 id from 球队 副 SQL: select 冠军球队 id from 欧冠冠亚军
COND	哪些高校拥有的学生总数不低于所有高校的平均值	select 名称 from 高校 where 本科生数量+研究生数量≥(select avg (本科生数量+研究生数量) from 高校)	主 SQL: select 名称 from 高校 where 本科生数量+研究生数量≥SQL 副 SQL: select avg (本科生数量+研究生数量) from 高校
ROW	在 17 年美国发生的枪击案中,德克萨斯受影响的学生占所有受影响学生总数的比例是多少?	select a. 受影响学生数量/b. 受影响学生数量 from (select 受影响学生数量 from 美国近几年校园枪击事件 where 年份==2017 and 发生城市==‘德克萨斯’) a, (select sum (受影响学生数量) from 美国近几年校园枪击事件 where 年份==2017) b	主 SQL: select 受影响学生数量 from 美国近几年校园枪击事件 where 年份==2017 and 发生城市==‘德克萨斯’ 副 SQL: select sum (受影响学生数量) from 美国近几年校园枪击事件 where 年份==2017

至此, 对复杂 SQL 语句的分解完毕, 形成了一个可生长变化的多叉树结构. 对于每一个 SQL 的生成, 其输出都对应树状模型中的各个节点, 最终通过根节点上这些可共用的单元预测模块, 形成最终的输出结果. 本文提出的技术框架以该树状模型为基础, 实现了从复杂自然语言查询预测生成 SQL 语句的整体流程和具体模型方法.

2.2.2 系统整体框架

本文设计的基于树状模型的复杂自然语言查询转 SQL 整体系统框架如图 2 所示. 整体的系统框架包含 3 个部分——数据预处理、模型训练推理和 SQL 解析后处理.

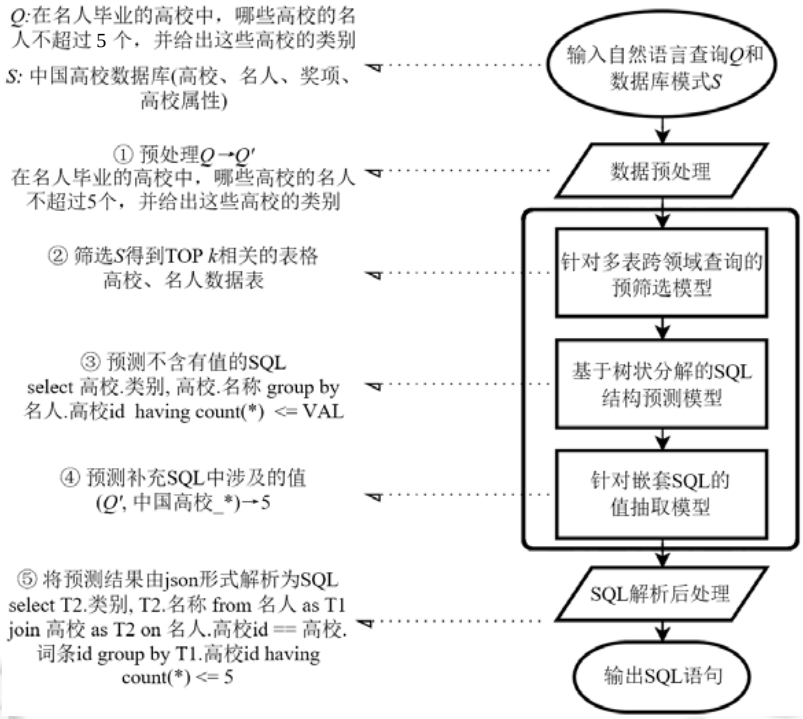


图 2 基于树状模型的复杂自然语言查询转 SQL 整体系统框架

针对数据预处理这一步骤, 主要对用户问句 Q 中涉及的一些不规范和不一致的表达进行统一化、规范化的预处理. 比如在查询涉及时间和数值类型的条件值时, 由于用户输入的自然语言查询是口语形式的表达, 年份 2019 可能被表达为“二零一九年”“去年”“19 年”等, 金额 100 000 可能被表达为“10 万”“十万”等,而 SQL 语句需要一个确切的值才能执行查询操作, 因此需要对这些时间日期、货币、数值、百分数等内容进行统一处理, 输出预处理后的自然语言查询问句 Q', 才能确保后续的 SQL 解析以及值抽取任务不受干扰.

模型训练及推理部分, 主要完成对输入的自然语言和数据库模式进行理解并对 SQL 各模块进行预测的工作, 也是整体框架中最为关键的一步, 最终输出 json 形式的预测结果, 包含各个 SQL 关键字子句的列名、操作符等基础组成部分, 提供给最终的 SQL 解析后处理模块作为输入. 区别于在 Spider 数据集上学术界当前主流的 SQL 预测生成单模型, 本文针对中文多表复杂自然语言查询转 SQL 的特点, 新增两个辅助模型用于提升基于树状模型的 NL2SQL 预测任务的性能表现和泛化能力, 形成层层递进的 3 个模型, 来更好地解决跨领域多表复杂查询带来的技术难点, 分别是针对多表跨领域查询的预筛选模型、基于树状分解的 SQL 结构预测模型和针对嵌套 SQL 的值抽取模型. 本文将在第 3 节详细介绍这 3 个模型, 各模型在整个框架中的作用如下.

(1) 针对多表跨领域查询的预筛选模型

本模型主要完成对数据库表格进行召回筛选的工作. 定义如下:

$$(Q',S)\rightarrow\{T_1,T_2,\ldots,T_k\} \tag{2}$$

输入预处理后的自然语言查询文本 Q'和相对应的数据库模式信息 S, 训练一个基于预训练模型的筛选模型对这些表进行预筛选, 得到与查询 Q'最为相关的 TOP k 的表格 T. 本阶段的输出极大降低了 NL2SQL 模型所需要输入的数据库规模, 从而解决低资源场景下多表查询数据库过大导致信息无法全部输入到 NL2SQL 模

型的难题. 并且合并 TOP k 个表的信息作为后续主模型的输入, 也可以把多表查询问题转化为单表查询问题, 并简化了最终 SQL 解析后处理进行多表连接的过程.

(2) 基于树状分解的 SQL 结构预测模型

本模型主要完成从自然语言查询文本预测生成不包含值信息的 SQL 主干结构的工作. 定义如下:

$$(Q',\{T_j\}_{j=1}^{k})\rightarrow\{K,C,OP\} \tag{3}$$

输入预处理后的自然语言查询文本 Q'和预筛选的 TOP k 个表结构, 基于自顶向下的思想, 在可生长变化的树状模型中, 从根节点 SQL 嵌套层到单 SQL 层(基本 SQL 语句), 再到 Clause 层("select""where"等子句),最后到 Target 层(列名、操作符等预测目标), 层层递进执行得到最终的预测结果. 本阶段的输出为自然语言查询对应的目标 SQL 的主干结构$\{K,C,OP\}$, 缺少了"where""having""limit"子句所需要的条件值, 主要考虑的是各层次里的模型需要分别单独训练, 如果包含值抽取, 会带来欠训练欠拟合的问题, 所以本阶段将值抽取统一交给后续的值抽取模型.

(3) 针对嵌套 SQL 的值抽取模型

本模型主要完成从自然语言查询文本中抽取 SQL 基本结构中条件列对应的条件值. 定义如下:

$$(Q',\{C_i\}_{i=1}^{m})\rightarrow\{V\} \tag{4}$$

输入自然语言查询文本 Q'和每一个列名 C_i, 通过阅读理解答案抽取模型, 返回其可能在一个或者多个 SQL 语句(指嵌套情况下)中相关的值. 本模型将"where""having""limit"子句的值抽取统一视作值抽取任务,统一训练. 针对嵌套 SQL 的情况, 学习用于主副 SQL 的答案抽取权重, 从而区分开主副 SQL 中同样的列对应的值来提升值抽取的性能表现. 本模型的输出结合前面的模型就完成了整个目标 SQL 各个组成部分的预测,形成完整的 json 形式 SQL 预测结果, 作为后续 SQL 解析后处理步骤的输入来得到最终输出 SQL.

在 SQL 解析后处理步骤中, 主要是将预测的 SQL 结果从各模块构成的 json 形式转换为 SQL 语句的表达形式. 对于自然语言查询文本中抽取到的值, 虽然经过一定的预处理, 但是并不一定能完全跟数据库存储的值一致, 所以首先通过值的后处理来完成值与数据库存储元素的匹配对齐, 本文采用了编辑距离、相似度匹配等算法来执行模糊匹配. 然后, 实现一个 SQL 解析算法, 将树状模型每一个层上的各个基本结构按照 SQL 规范进行元素拼接组装, 将 json 转变为 SQL 形式, 最后根据查询涉及的列用图模型生成表连接关系, 填充 from 子句. 至此完成自然语言查询转 SQL 的全部流程.

3 基于树状模型的复杂自然语言查询转 SQL 模型实现

3.1 针对多表跨领域查询的预筛选模型

本文设计了针对多表跨领域查询的预筛选模型, 对输入的数据库表格进行预筛选, 得到与查询最相关的 TOP k 表格, 极大地降低了输入表格的规模, 可以有效缓解数据库规模较大时无法完全将信息输入到后续 SQL 结构预测模型的难题. 模型网络结构如图 3 所示.

模型的输入为预处理后的自然语言查询 $Q'=W_{1,\cdots,m}$ 和数据库模式 $S=T_{1,\cdots,n}$, 在对数据库多表格进行筛选时, 考虑到各表格字段中含有的信息, 模型同时关注数据表格 $T=C_{1,\cdots,s}$的表名和字段名部分, 并定义表格的输入形式为

$$Total=T_{name}\_C_{1\_name}\_C_{2\_name}\_\cdots\_C_{s\_name} \tag{5}$$

其中, T_{name} 表示当前表名, C_{s_name} 表示表格中最后一个字段的列名, 它们通过"_"和" "的形式拼接起来组成表格的输入字符串, 接着, 问题查询 Q 和数据库多个 *Table* 字符串进行 Wordpiece 分词[19]加上特殊标记后构成模型的输入, 形式为

$$Input=Concat([CLS],Q',[SEP],Table_1,[SEP],\ldots,Table_n,[SEP]) \tag{6}$$

其中, '[*CLS*]'和'[*SEP*]'代表 BERT 中的特殊标记, *Concat* 表示 Q'与 n 个 *Table*, 同特殊标记一起进行拼接操作得到输入序列 *Input*, 如图 3 所示. 表的预筛选问题可以看作求取问题与表格相似度任务, 如果遇到信息无法

完全输入到本模型的极端情况, 可以采用滑动窗口构建多次输入获取全部表格的分数来完成模型的预测.

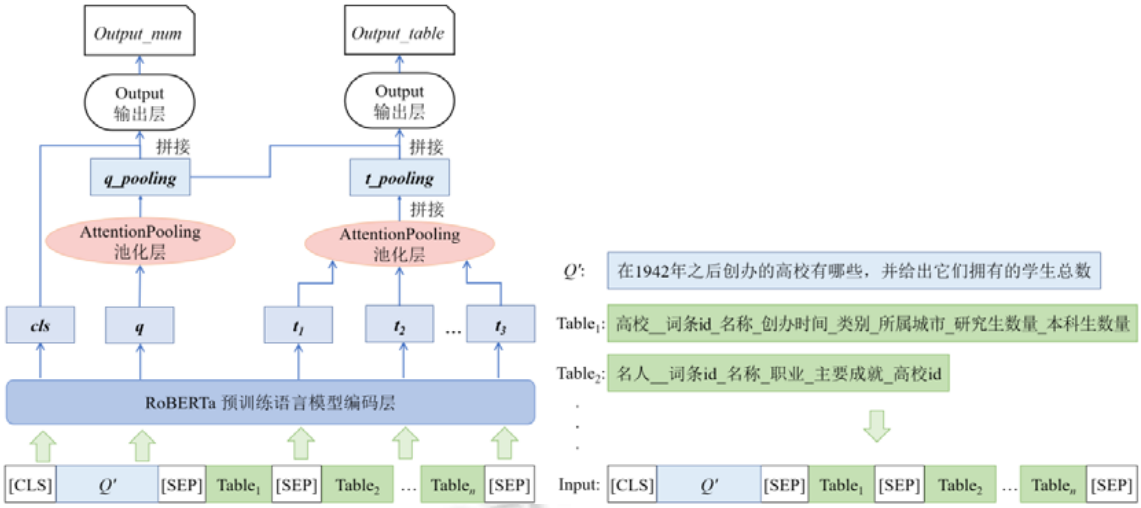


图 3 针对多表跨领域查询的预筛选网络和输入样例

在编码层, 考虑到模型在实际应用中面临跨领域查询的问题, 并且口语查询以及各种数据库的理解需要很多的先验知识, 本文采用了预训练语言模型 RoBERTa[20]作为编码器, 能够更好地提取自然语言查询和数据表信息的特征. 同时, 分离得到输入中各部分的编码:

$$\boldsymbol{cls},\boldsymbol{q},\boldsymbol{t}_1,\boldsymbol{t}_2,\ldots,\boldsymbol{t}_n=Split(RoBERTa(Input)) \tag{7}$$

其中, $\boldsymbol{q}\in\mathfrak{R}^{n*D}$ 为预处理后的自然语言查询 Q'的编码, 长度为 m, D 代表 RoBERTa 编码维度; $\boldsymbol{t}_i\in\mathfrak{R}^{L_i*D}$ 为第 i 个数据表的编码, L_i 表示各表编码的长度. 由于每个数据表的编码长度不一, 为了后续的表格筛选任务目标, 需要对各表编码 $\boldsymbol{t}_i$ 进行降维. 同时, 为了后续查询和数据表信息的交互融合, 也要对 $\boldsymbol{q}$ 进行降维处理. 本模型采用注意力(attention)的形式进行降维:

$$score=softmax(\boldsymbol{x}\cdot W) \tag{8}$$

$$AttentionPooling(\boldsymbol{x})=score^T\cdot\boldsymbol{x} \tag{9}$$

其中, $\boldsymbol{x}\in\mathfrak{R}^{n*d}$ 为输入, $W\in\mathfrak{R}^{n*1}$ 为可学习的参数权重矩阵. 获得输入序列中各位置对整体信息编码的贡献度 $score\in\mathfrak{R}^{n*1}$ 后, 再通过 weight-sum 的形式进行降维得到最终的输出 $AttentionPooling(\boldsymbol{x})\in\mathfrak{R}^{1*d}$. 将问题编码 $\boldsymbol{q}$ 和各表格编码 $\boldsymbol{t}_i$ 分别经过 AttentionPooling 层, 最终得到降维后的问题特征 $\boldsymbol{q\_pooling}\in\mathfrak{R}^{D}$, 以及各表特征沿着长度维度进行拼接得到的表序列编码 $\boldsymbol{t\_pooling}\in\mathfrak{R}^{n*D}$, n 为表的个数.

本模型还包括特征融合层. 由于对数据表进行预筛选主要是给每个表进行打分, 上一步经过注意力池化后已经得到了长度为 n 的表序列编码 $\boldsymbol{t\_pooling}$, 本模型将打分看作是序列标注任务. 表打分参考的是表同问题查询的相关程度, 所以在此加入特征融合层, 将输入的问题查询编码信息分别融合到各表编码上, 从而得到用于后续标注的表特征来进行表的筛选. 其中, 先对问题编码 $\boldsymbol{q\_pooling}$ 进行复制(repeat)变成长度为 n, 再和表序列编码 $\boldsymbol{t\_pooling}$ 进行拼接得到最终的表格序列特征 $\boldsymbol{t\_feature}\in\mathfrak{R}^{n*2\cdot D}$:

$$\boldsymbol{t\_feature}=Concat(\boldsymbol{t\_pooling},Repeat(\boldsymbol{q\_pooling})) \tag{10}$$

在最终模型输出部分, 需要先获得每个表与查询问题的相关度分数, 然后获取 TOP k 最相关的表格(k 可设为查询中允许关联的表的最大个数). 对上一步融合了查询问题特征的表序列特征 $\boldsymbol{t\_feature}$ 进行序列标注:

$$Output\_table=Dropout(\boldsymbol{t\_feature})\cdot W_a \tag{11}$$

$$Predict\_table=Topk(Output\_table) \tag{12}$$

其中, $W_a\in\mathfrak{R}^{D*1}$ 是可学习的参数矩阵, Dropout 为随机丢弃部分神经元来防止过拟合的机制, $Topk$ 函数按照从大到小的顺序从预测的各表分数 $Output\_table\in\mathfrak{R}^{n*1}$ 中取得最大的 k 个表格 $Predict\_table\in\mathfrak{R}^{k}$. 虽然本模型任务目

标是筛选 TOP k 个表格, 但是为了提升模型的性能表现和充分利用数据集的信息, 本文基于多任务学习的思想, 给模型引入了一个额外的训练目标——与查询相关表的数量 *Output_num*:

$$Output\_num=Dropout(Concat(\boldsymbol{q\_pooling},\boldsymbol{cls}))\cdot W_b \tag{13}$$

$$Predict\_num=\text{argmax}(Output\_num) \tag{14}$$

其中, $W_b\in\mathfrak{R}^{D*k}$ 是可学习的参数矩阵, $Output\_num\in\mathfrak{R}^k$ 为表个数的预测分数, 经过 argmax 操作得到最终的结果 $Predict_{num}\in\{1,2,\ldots,k\}$. 该预测目标还可以解决不涉及具体列的非常规查询如“查询高校表所有信息?”的表名预测, 其对应 SQL“select * from 高校”由于没有列名, 在后续 SQL 解析中无法通过表列关系来直接确定查询表, 此时通过表预筛选的排序和数量确定.

根据上述训练目标, 该模型训练时的输入为查询问题和数据库模式, 输出包括[0,1]取值的相关表序列标注(表示表是否出现在 SQL 中)和数值形式的相关表个数标注. 如查询样例“在名人毕业的高校中, 哪些高校的名人不超过 5 个, 并给出这些高校的类别”对应的数据库——中国高校(高校、名人、奖项、高校属性)的相关表序列标注为[1,1,0,0], 相关表个数标注为 2. 因此, 模型损失函数分别采用二进制交叉熵 $BCE(y,p)$和交叉熵 $CE(y,p)$损失函数, 各预测目标的损失函数分别为:

$$Loss\_table=CE(label\_table,sigmoid(Outpute\_table)) \tag{15}$$

$$Loss\_num=CE(label\_num,softmax(Outpute\_num)) \tag{16}$$

整个筛选模型总的损失函数为

$$Total\_Loss=\alpha\cdot Loss\_table+Loss\_num \tag{17}$$

其中, 对于表标注的分数 *Output_table* 需要先通过 *sigmoid* 函数得到概率分布, 再计算二进制交叉熵; 对于表个数的分数计算交叉熵损失前进行 *softmax* 函数归一化得到概率. 对于整体的联合损失函数 *Total_Loss*, 其由表选择任务的损失和预测表个数任务的损失通过α系数权重相加, 系数代表着模型对表选择任务的关注度. 为了提升主要的筛选任务性能表现, α会相对较高, 如本文后续实验中, 设定α=3.

以上表预筛选模型在跨领域查询时, 面对各种不同的数据库结构虽然可以去除大部分无用输入信息的干扰, 但对于模型跨领域查询时面对的新数据库模式很可能相比训练数据有较大变化, 导致模型预测效果不稳定, 产生模型性能下降的严峻挑战. 本文提出了一种优化算法对数据库表进行信息增强, 通过对数据表、表列名进行随机采样、丢弃、排序的操作, 缓解训练过拟合的问题, 从而提高模型的稳定性. 该算法在模型训练过程中, 每个训练周期(epoch)进行操作, 是一种动态的数据增强算法对数据集进行扩充, 从而提升模型的训练效果和推理性能表现. 由于数据库中表之间的顺序没有强烈的关联性, 同时大部分 SQL 语句关联的数据表数量都在一个合理范围内, 过多冗余的表输入也会给模型带来干扰性导致性能下降, 过少的表输入会导致训练不充分, 所以本算法的思想在于随机保留部分表, 且不仅完全包含查询问题相关的数据库, 也会适当随机选取其他不相关数据库来增加模型学习的难度. 同时, 这些表会随机打乱顺序, 防止模型关注到不必要的表之间的顺序信息. 为防止某些表数据列过多导致输入过长, 本算法也随机保留固定个数的数据列.

3.2 基于树状分解的SQL结构预测模型

本节基于 SQL 树状分解模型, 提出了完整的 SQL 结构预测模型, 将自然语言查询转换为包含 SQL 关键字、表列、计算操作符等完整信息的 SQL 主干结构信息. 基于树状分解的 SQL 结构预测模型整体框架如图 4 所示, 包含输入层、编码层、特征融合层, 以及与复杂 SQL 树状分解所对应的包含各可复用 SQL 模块的树状模型和所有 Target 的输出.

模型的输入为预处理后的自然语言查询 Q'和经过筛选后的 TOPk 数据表. 由于输入的表数量减少, 通过将每个表的表名信息同列名信息进行融合一起输入到模型中, 再合并作为列序列可以转化为单表查询预测问题, 方便后续的答案列预测等序列标注任务. 同时, 考虑到 SQL 语句中特殊的标记“*”的存在, 其和行数有关, 大多跟聚合操作 count 一起使用, 本文将其作为第 0 号特殊列输入, 并且为了解决特殊的基于时间的 SQL 常数计算任务, 在 *cac* 列计算的预测中将第 0 号特殊列视作“TIME_NOW”常数, 这样也使得其转化为列计算任务. 本文定义列序列输入形式为

$$Col_{1,\dots,n}=\{T^i_{name}\_C_{i,1\_name},T^i_{name}\_C_{i,2\_name},\dots,T^i_{name}\_C_{i,s\_name}\}_{i=1}^{K} \tag{18}$$

其中，T^i_{name} 表示第 i 个表名，C_{i,s_name} 表示第 i 个包含 s 个字段的表格中最后一个字段的列名，它们通过“_”的形式拼接起来，组成列序列的输入字符串；接着，Q'和列序列 $Col_{1,\dots,n}$ 分别进行 Wordpiece 分词成模型的输入，形式为

$$Input=Concat([CLS],Q',[SEP],Col_1,[SEP],\dots,Col_n,[SEP]) \tag{19}$$

图 4 给出了 DuSQL 数据集上一条样例数据对应的模型输入.

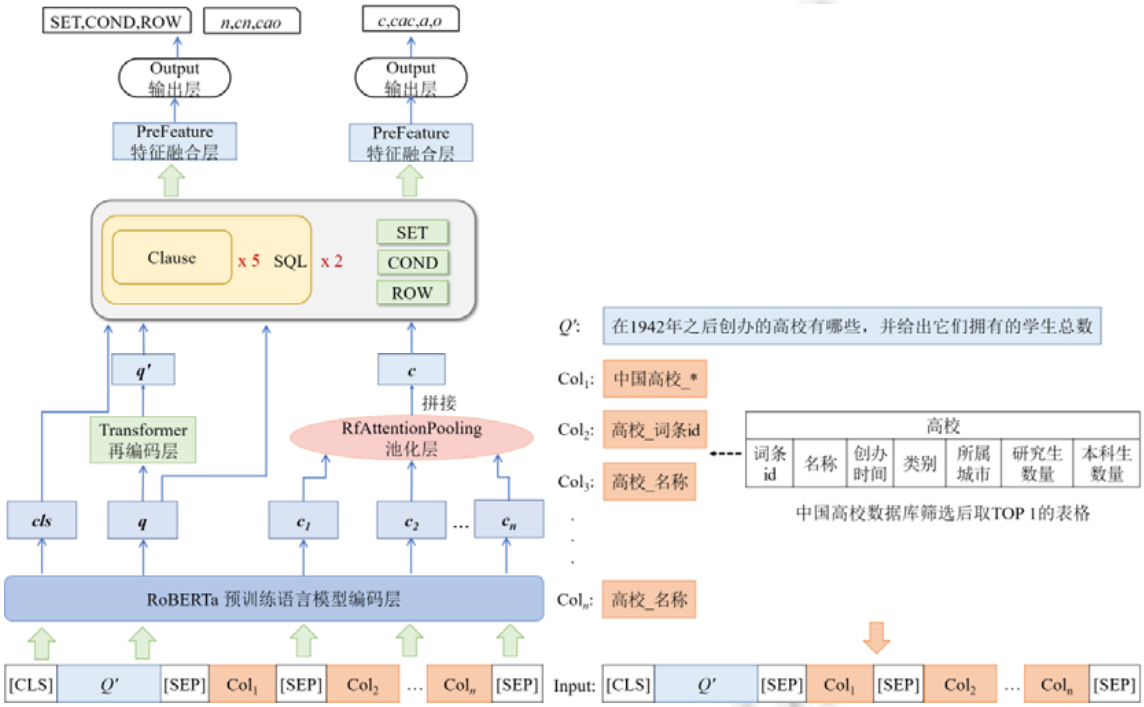


图 4　基于树状模型的 SQL 结构预测网络和输入样例

在编码层，本模型依然采用预训练语言模型 RoBERTa 作为编码器，提取自然语言查询 Q'和融合了表名的列序列的特征. 同时，分离得到输入中各部分的编码 ***cls***, ***q***, ***col***. 为了使得核心的表列名信息去更好地融合查询问题信息，本文还借鉴了 X-SQL[3]提出的一种利用隐式问题信息通过特殊标记进行表达的增强上下文信息列名编码方案的思想，对其进行了改进，提出了一种显式融合查询问题信息的注意力形式 RfAttentionPooling，直接显式使用问题查询特征 q 去辅助学习列名中各位置的权重分数，起到了增强信息的作用，定义如下:

$$score=softmax\left(\max\left(\frac{(\boldsymbol{q}\cdot W_1)\cdot(\boldsymbol{col}\cdot W_2)^T}{\sqrt{D}}\right)\right) \tag{20}$$

$$RfAttentionPooling(\boldsymbol{q},\boldsymbol{col})=score^T\cdot\boldsymbol{col} \tag{21}$$

对每一个列名编码 $\boldsymbol{col}^n_{i=1}$ 和问题 ***q*** 编码结合进行降维后，拼接得到最终的列序列特征 $\boldsymbol{c}\in\mathbb{R}^{n*D}$，作为后续任务预测的输入. 此外，考虑到复杂 SQL 语句的主副 SQL 对于自然语言查询问句的关注点不同，两个 SQL 模块输入相同的问题查询编码会产生信息干扰混淆，本文提出再编码机制，对进入副 SQL 模块的问题查询编码 ***q***，采用 *Transformer* 进行再编码处理得到 $\boldsymbol{q}'$.

所有的编码特征信息都要逐层经过树状模型各层次的不同预测模块，最终用于预测输出. 其中，Clause 层的每个模块都接收着来自上层单 SQL 层模块的编码信息，包括查询问题编码 ***q*** 或者 $\boldsymbol{q}'$以及列序列编码 ***c***. 而 Clause 层的各个不同模块又由 Target 层的模块组成，对于不同的 Target，本文针对其任务特点划分出两种特征

形式, 来接收上游传来的各编码信息包括{**cls**,**q**,**q**′,**c**}, 并进行特征的融合来方便最后的预测输出:

$$ClsFeature(\boldsymbol{cls},\boldsymbol{q},\boldsymbol{c})=LayerNorm(\boldsymbol{cls}\cdot W_3+\boldsymbol{q}_{pooling}\cdot W_4) \tag{22}$$

$$PreFeature(\boldsymbol{q},\boldsymbol{c})=LayerNorm(\boldsymbol{c}\cdot W_5+Repeat(\boldsymbol{q}_{pooling})\cdot W_6) \tag{23}$$

其中, $\boldsymbol{q}_{pooling}$ 为经过注意力 Attention 降维的问题编码, $\boldsymbol{q}$ 在副 SQL 模块下表现为 $\boldsymbol{q}'$; $W_{3,4,5,6}\in\mathfrak{R}^{D\cdot D}$ 为可学习的参数权重矩阵. LayerNorm 对编码维度进行归一化, 输出各特征形状为 $ClsFeature(\boldsymbol{cls},\boldsymbol{q},\boldsymbol{c})\in\mathfrak{R}^{2\cdot D}$ 以及 $PreFeature(\boldsymbol{q},\boldsymbol{c})\in\mathfrak{R}^{n*2\cdot D}$, n 为列序列的长度. ClsFeature 为分类特征层, 包含 *n*, *cn*, *cao* 这 3 个 Target, 用于多分类输出; PreFeature 为标注特征层, 包含 *c*, *cac*, *a*, *o* 这 4 个 Target, 用于序列标注输出.

对于模型输出部分每个单 SQL 模块——每个 Clause 模块——每个 Target 模块的预测, 本文将他们的预测形式做了统一. 将上一步得到的 ClsFeature 和 PreFeature 的 ***feature*** 送入 Output 预测得到输出即可:

$$Output(\boldsymbol{feature})=Dropout(\boldsymbol{feature}\cdot W) \tag{24}$$

除了每个 Target 的预测输出, 在与单 SQL 层模块并列的位置, 还有解决 SET, COND, ROW 嵌套预测的 3 个预测目标. 由于它们都是多分类任务, 故也当做 ClsFeature 融合特征进行最后的 Output 预测, 输出维度与它们可能取值的范围相关, 在本文实验的数据集中分别设置为{4,3,5}. 在全部的 Target 中, 对于 ClsFeature 形式的如 *n*, *cn*, *cao*, 通过 argmax 函数得到结果. 对于 PreFeature 形式情况则各有不同: 由于列计算 *cac* 在一个 Clause 中只出现一次, 只有两个列, 亦取 argmax 函数得到结果; *c* 答案列的预测要根据列个数 *k* 的预测取 TOP *k* 的答案列; *a*, o 聚合操作符和条件操作符是每个都与特定列进行绑定, 所以先取 argmax 函数得到全部列对应的结果, 再根据 *c* 预测的答案列下标取出对应的操作符. 整个模型的输出内容由 SQL 的复杂程度决定, 如在图 1 的复杂 SQL 拆解后的多叉树中, 共含有 2×5=10 个 Clause 节点, 在不考虑某些 Target 输出为空的情况下, 该例子共有 2×5×7+3=73 个输出, 其中 3 个用于预测嵌套情况, 可以根据是否嵌套来确定是否需要副 SQL 模块的预测结果.

根据模型的训练目标, 模型在 Target 层输出的标注形式中, 列预测为[0,1]取值的序列标注, 操作符预测为[−1,*n*]取值的序列标注, 其中: −1 表示当前序列位置不对应答案列, 在计算 loss 的时候进行忽略; 其余为类别的数值标注. 对于不存在的 Clause 输出来说, 除了 *n* 为 0, 其余都用−1 进行填充来避免损失的计算; 同时, 不存在嵌套副 SQL 的情况下, 也对副 SQL 的输出进行同样处理. 损失函数分别采用二进制交叉熵和交叉熵损失函数, 各输出的损失函数具体为

$$\begin{aligned}
&Loss_c = BCE(label_c, sigmoid(Output_c))\\
&Loss_t = CE(label_t, softmax(Output_t)) \text{ for } t \text{ in } (n,cn,a,o,cao)\\
&Loss_{cac} = CE(label_{cac}, softmax(Output_{cac})) \text{ if } cao>0 \text{ else } 0
\end{aligned} \tag{25}$$

每个 Clause 模块损失为

$$Loss\_Clause = \alpha\cdot Loss_c + Loss_{cac} + \sum Loss_t \tag{26}$$

其中, α控制列名预测的关注度, 由于列预测难度大, 本文实验中取其为 3. 每个 SQL 模块损失函数如下:

$$Loss\_SQL_{1,2} = \sum Loss\_Clause \text{ for } Clause \text{ in } (\text{‘select’},\text{‘where’},\text{‘having’},\text{‘order by’},\text{‘group by’}) \tag{27}$$

其中, 副 SQL 模块在不含嵌套情况下的损失为为 0:

$$Loss\_SQL_2=0 \text{ if } SET+COND+ROW=0 \tag{28}$$

而对于 3 种嵌套(*SET*,*COND*,*ROW*)部分的损失预测为

$$Loss_e=CE(Loss_e,softmax(Outpute)) \tag{29}$$

最终得到模型整体的联合损失函数:

$$Total\_Loss = Loss\_SQL_1 + \beta\cdot Loss\_SQL_2 + \sum Loss_e \tag{30}$$

整体的联合损失函数 *Total_Loss* 含有一个控制副 SQL 模块损失比例的系数β, 由于训练集中包含嵌套 SQL 的数据比例较低, 需要给与其更高的关注度, 本文实验中将其设置为 4.

3.3 针对嵌套SQL的值抽取模型

SQL 语句中值是重要的组成部分, 会出现在“where”“having”子句的条件下作为条件值以及在“orderby”子句中执行 limit 数量限制, 一般有数值、文字、日期这 3 种类型. 在多表复杂查询任务上, 本身复杂多变的 SQL 生成对模型已经是很大的挑战, 再在模型中加上值抽取会变得更加困难. 其中最主要的难题就是让 SQL 中的每种 Clause 在预测列才同时输出值会让训练不够充分, 因为训练数据中含有值的子句往往不足且对不同的 Clause 分布不均衡, 某些子句模块的参数优化达不到预期效果. 同时, 在复杂查询的嵌套SQL中, 相同的列名可能对应着不同的条件值, 比如“北京的户籍人口比上海多多少?”都是“名称”列, 但是要在两个 SQL 子句上分别需要得到“北京”“上海”的值, 这也给复杂查询下 SQL 的值抽取工作带来了挑战. 本文提出一种针对嵌套 SQL 的值抽取模型, 将值抽取任务从 SQL 预测生成的主模型中分离出来, 将不同类型 Clause 子句的值抽取视作相同任务共享权重进行训练和推理, 有效提升值抽取训练集的规模, 解决树状模型无法有效训练值抽取目标的问题. 并且针对于值抽取中可能存在的同列却对应不同值的情况, 本模型对嵌套 SQL 抽取进行改进,将模型分为两个抽取模块, 每个模块去关注问题查询中对应不同子 SQL 的部分信息, 最终共享权重分别输出不同子 SQL 所含有的条件值. 模型网络结构如图 5 所示.

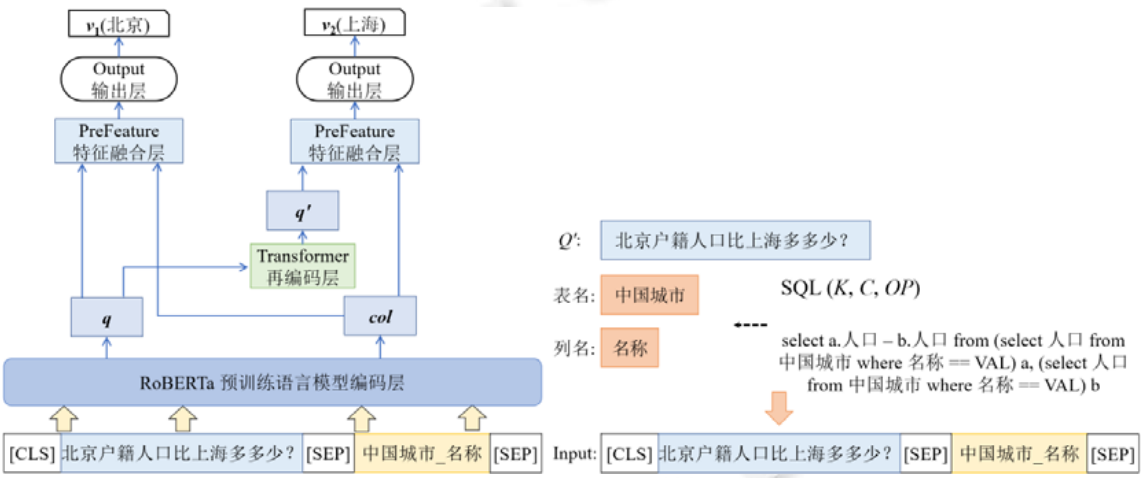


图 5 针对嵌套 SQL 的值抽取网络和输入样例

模型输入为预处理后的自然语言查询 Q'和 NL2SQL 树状模型预测出来的不包含值的 SQL 结果. 值可能出现在“where”“having”和“orderby”子句, 且必定有特定的列与其对应组成条件, 如“名称=北京”. 不过, 条件列并也不一定都对应值, 如 COND 嵌套中值部分是一个需要预测的副 SQL 语句. 存在一种特殊情况是“orderby”子句中的“limit”关键字涉及的值抽取: 对于“面积最大的城市是哪个”这种问题, 其“orderby”子句为“orderby 城市. 面积 limit1”, 这时“1”这个值无法从问题中抽取出来, 所以本文将其放到树状模型“orderby”的 Clause 模块下 Target 中的 cn 进行预测, cn 取{0,1,2}分别表示不含“limit”条件、含有最值条件“limit 1”、含“limit”条件但需要进行值抽取. 这里的值抽取模型只针对 cn=2 的“orderby”子句进行处理. 本模型将 SQL 的值抽取任务视作特定列对于自然语言查询的问答任务, 其中需要进行值抽取的特定列视为问答任务中的 Query, 查询 Q'视为 Context, 需要根据 Query 从 Context 中抽取出答案值区间. 由于 SQL 语句中可能包含多个需要抽取的值, 需要针对每个特定列分别去抽取答案, 且考虑到相同列名可能出现在不同表格, 但其对应的值在问题中是区分的, 所以定义特定列的输入为包含了表信息的形式:

$$Col=T_{name}\_C_{name} \tag{31}$$

其中, C_{name} 表示不包含值信息的 SQL 结构预测结果中需要进行值抽取的特定列名, T_{name} 表示该列对应的表名, 它们通过“_”的形式拼接起来组成问题的输入字符串. 接着, 上下文 Q'同查询 Col 通过 Wordpiece 分词构成模

型的输入, 形式为

$$Input=Concat([CLS],Q',[SEP],Col,[SEP]) \tag{32}$$

一个一般的输入样例如图 5 所示. 针对 DuSQL 中包含行计算的特殊情况, 特定列不是单独的列名而是两个计算列名, 此时 *Col* 通过 *cao* 的操作符拼接起来输入, 类似“高校_本科生数量+研究生数量”这样的形式. 针对表示数量的 *COUNT*(*)列特种特殊情况, 因为不包含表列名, 使用数据库名作为输入, 如“中国高校_*”.

在编码层, 本模型与前面两个模型一样, 仍然采用预训练语言模型 RoBERTa 作为编码器, 来提取自然语言 Context 和特定列 Query 的特征, 同时分离得到各部分的编码. 同前两个模型编码层不同的是, 本模型抛弃了 *cls* 编码信息.

值抽取是一个抽取式问答的任务, 其根据列名 Query 从问题查询 Context 中预测答案的起止位置构成答案区间, 所以可以看作是对于问题查询的序列标注. 在第 3.2 节中, 本文提出了针对序列标注任务的 PreFeature 特征融合形式, 将查询问题编码特征融合到列序列特征中; 而在本模型中要预测问题查询中的答案区间, 所以将问题查询上下文编码 q 变为标注序列. 另外, 考虑到具有嵌套 SQL 语句的样本较少, 意味着副 SQL 模块对应的值抽取不能充分训练. 本文采用了权重共享策略, 不同 SQL 模块的上下文特征共用一个 PreFeature 层, 唯一区别在于输入问题编码不同. 本模型还采用上一节所描述的再编码层对送入副 SQL 模块的问题编码进行再编码. 最终, 特征融合形式为

$$\boldsymbol{Context\_feature}_1=PreFeature(\boldsymbol{col},\boldsymbol{q}) \tag{33}$$

$$\boldsymbol{Context\_feature}_2=PreFeature(\boldsymbol{col},\boldsymbol{q}') \tag{34}$$

其中, $\boldsymbol{q}'$为 $\boldsymbol{q}$ 通过再编码层得到的用于副 SQL 值抽取的 Context 编码, 分别得到了用于不同 SQL 子句值抽取的上下文特征 $\boldsymbol{Context\_feature}_{1,2}\in\mathcal{H}^{m*2\cdot D}$.

在模型输出部分, 需要先获得特定列在不同 SQL 子句的答案区间预测, 接着根据特定列所在 SQL 子句位置决定其选择哪些结果. 将上下文特征 ***Context_feature*** 采用上一小节的输出层方式进行序列标注, 同时, 类似特征融合层的权重共享策略, 不同 SQL 子句的值抽取也采用共享同一个输出层的优化策略.

本模型为了解决嵌套 SQL 下的值抽取, 在训练阶段将不同 SQL 子句的值抽取部分损失分离开来. 结果 SQL 中每个需要进行值抽取的条件列可能出现在不同的 SQL 子句中, 也可能是在两个 SQL 子句同时出现. 比如, 图 8 输入样例中的“中国城市_名称”这个列对应着两个 SQL 子句下的“北京”和“上海”. 所以, 对于损失函数的定义为:

$$Loss_{v_1}=CE(label_{v_1},softmax(Output_{v_1})) \text{ if } col \text{ in } SQL1 \text{ else } 0 \tag{35}$$

$$Loss_{v_2}=CE(label_{v_2},softmax(Output_{v_2})) \text{ if } col \text{ in } SQL2 \text{ else } 0 \tag{36}$$

$$Total\_Loss=Loss\_v_1+Loss\_v_2 \tag{37}$$

其中, $label_{v_1}$, $label_{v_2}$ 答案值的标签信息通过对上下文进行分词处理, 然后与答案进行匹配获取到具体下标 Index 得到. 由于口语中对于值描述存在不完整的可能, 比如对于“浙江大学”这个值, 口语可能表达为“浙大”, 所以在打标签的过程中, 是通过与分词后的序列进行编辑距离、相似度等模糊匹配方式来完成. 并且在 SQL 解析后处理模块中, 需要对抽取到的答案值与数据库元素进行匹配.

经过以上 3 个模型的处理, 复杂自然语言查询转 SQL 解决方案接近完成, 得到的是 json 形式的 SQL 结果, 通过 SQL 解析后, 处理模块可将其转化为最终的 SQL 语句.

4 实验结果及分析

4.1 实验数据集

本文关注于面向复杂查询的 NL2SQL 任务, Spider 和 DuSQL 数据集都是跨领域的多表复杂查询数据集, 不过对于复杂的口语查询问题而言, 中文相比于英文由于语法、语言习惯、分词的不同会更加灵活多变, 导致更加难以被模型理解; 同时, 中文 DuSQL 数据集相比于 Spider 数据集增加了更加面向于实际落地场景, 如

商业智能、业务咨询等的复杂计算任务, 并且在其竞赛任务中, 评价指标参考了含有值的完全 SQL 的匹配准确率. 本文以中文复杂自然语言查询转 SQL 问题为研究对象, 所以选取 DuSQL 数据集作为实验数据集, 并且同 WikiSQL 以及 Spider 上的主流 SOTA 模型 X-SQL, IRNet, RAT-SQL 进行对比实验. DuSQL 数据集的一些样例见表 3.

表 3 DuSQL 数据集的数据样例

问题类型	问题实例	SQL 查询语句
单/多条件匹配	绿化率在 30%以上的城市有哪些?	select 名称 from 中国城市 where 绿化率>30%
排序	绿化率前 5 的城市?	select 名称 from 中国城市 order by 绿化率 desc limit 5
分组	哪个省的平均绿化率最高?	select 所属省 from 中国城市 group by 所属省 order by avg(绿化率) desc limit 1
行计算(SQL 计算的一种)	北京户籍人口比上海多多少?	select a. 人口/b. 人口 from (select 人口 from 中国城市 where 名称==‘北京’) a, (select 人口 from 中国城市 where 名称==‘上海’) b
列计算(SQL 计算的一种)	北京人口密度是多少?	select 人口/面积 from 中国城市 where==‘北京’
常数计算(SQL 计算的一种)	成立时间不到 14 年且年营业额超过了 2 000 万的公司有哪些	select 名称 from 公司 where TIME_NOW - 成立时间<14 and 年营业额>20000000

4.2 评价指标

自然语言查询转 SQL 任务中, 常用的性能评价指标为准确率(*accuracy*), 表示预测的 SQL 语句是否与真实标签 SQL 语句完全相同. 不过, 考虑到 SQL 语句某些组件如“where”子句中多个条件顺序不同但是意义却相同带来的影响, 一般将 SQL 语句分成若干个组件进行完全匹配, 通过各子组件的精确匹配来评价预测 SQL 语句的正确与否, 忽略多个条件的顺序不一致问题.

4.3 实验设置

由于预训练语言模型强大的编码能力, 本文模型全部使用了在中文维基百科大规模语料上采用全词 MASK 形式进行预训练的 RoBERTa 预训练语言模型(https://github.com/ymcui/Chinese-BERT-wwm), 采用 Transformers 开源框架(https://github.com/huggingface/transformers)载入预训练语言模型, 并且对词表进行修改, 使得其可以识别 X-SQL 提出的表示上下文信息标记的“CTX”特殊标记. 训练过程全部采用 fine-tuning 微调的形式.

训练过程中固定所有随机种子, 训练周期 epoch 为 10, batch size 为 24, 学习率为 3e–5; 采用默认参数的 Adam[21]自适应学习率优化器, 并使用 warmup 机制训练, Dropout[22]的比率设置为 0.1, 特征层中 Layer norm[23]采用的 eps 参数同 BERT[24]采用的配置相同, 为 1e–12, 防止过拟合的权重衰减参数 weight decay 设置为 0.01, 梯度裁剪防止梯度爆炸以及稳定训练设置最大梯度为 5.

4.4 实验结果和分析

单表简单查询 WikiSQL 数据集上, 2019 年, SOTA 模型 X-SQL 提出了一种增强上下文信息的列名编码方案, 由于列名编码是本模型的核心重点, 本文提出了一种新的显式融合问题查询信息的列名编码方案 RfAttentionPooling. 下面给出本模型的一些实验结果, 对比方法分别是:

- Ours: 本文提出的树状模型, 对于列名编码采用了显式融合问题查询信息的 RfAttentionPooling, 并且使用表格增强算法进行训练;
- Ours_X-SQL: 与 Ours 模型区别在于, 将列名编码方案换成 X-SQL 提出的增强上下文信息注意力机制;
- Ours_No-Aug: 与 Ours 模型相同, 但是训练中不采用表格增强算法;
- Ours_No-Re: 与 Ours 模型相同, 但是不采用针对于副 SQL 模块预测优化的再编码机制.

本次实验主要对比表列编码和筛选方案, 结果不包含值的预测. 表 4 展示了各模型在 DuSQL 数据集上的实验结果, 其中, “全部”代表预测 SQL 的各模块输出全部相同的准确率, SQL_1, SQL_2 分别代表各 SQL 子句

部分的准确率, SET, COND, ROW 则代表着 3 种嵌套情况预测的准确率. 对实验结果进行分析, 每个模型都在嵌套相关的预测中获取接近满分的成绩, 这也说明了树状模型将复杂 SQL 根据嵌套情况进行分解的可行性. Ours 同 Ours_X-SQL 模型相比, 整个 SQL 完全预测正确的比率差距不大, 但在 SQL_1, SQL_2 不同 SQL 子句下, 累计有 1.5 个百分点的效果提升, 说明了本文提出的 RfAttentionPooling 显式地在列名编码中融合问题查询特征辅助, 学习列名中各位置的权重分数起到了增强信息的作用, 相比于 X-SQL 利用“CTX”特殊标记隐式增强上下文信息的方法更有优越性.

表 4 树状模型下生成复杂 SQL 结构准确性实验对比

模型/准确率	全部	SQL_1	SQL_2	SET	COND	ROW
Ours	**0.848**	**0.851**	**0.915**	**1.0**	0.998	**1.0**
Ours_X-SQL	0.844	0.850	0.901	**1.0**	0.997	**1.0**
Ours_No-Aug	0.838	0.842	0.893	**1.0**	**0.999**	**1.0**
Ours_No-Re	0.839	0.844	**0.915**	**1.0**	0.998	**1.0**

Ours_No-Aug 与 Ours 模型的差距有 1 个百分点, 且各 SQL 子句累计 3.1 个百分点的差距, 是因为本文方法在训练数据表预筛选模型时采用了表信息增强算法, 对数据表、列名随机采样、丢弃、排序等操作, 很好地缓解了跨领域查询时面对不一样的数据库时因为模型过拟合导致性能下降的难题, 有力证明了该方法的有效性.

与 Ours_No-Re 模型进行对比, 由于 Ours 模型将送入不同 SQL 模块的问题特征进行了再编码, 有效避免了嵌套查询中不同 SQL 字句模块的预测输出不同会干扰模型对问题特征进行准确学习的难题, 整体提高了 0.9 个百分点整体 SQL 匹配准确率, 并且尤其表现在 SQL_1 子句部分的性能提升, 也验证了不同 SQL 子句模块对相同问题编码确实产生了混淆的观点.

至此生成了不含值的 SQL 各模块输出的结果, 经过值抽取模型补充值后以及 SQL 解析后得到最终的 SQL 语句. 接下来的实验指标参考的是 DuSQL 官方提供的评测脚本来计算预测 SQL 语句是否与标签 SQL 语句完全匹配的准确率.

Spider 数据集上 SOTA 模型 IRNet 和 RAT-SQL 不具备值抽取以及各种 SQL 计算的功能, 针对 DuSQL 数据集对其适配加入这些功能后, 本文给出了最终对比结果见表 5, 各对比模型简介如下.

- Baseline(https://github.com/PaddlePaddle/Research/tree/master/NLP/DuSQL-Baseline): 百度官方给出的基于 seq2seq 模型的系统框架, 采用 Bi-LSTM 进行编码;
- IRNet: Spider 数据集 2019 年的 SOTA 模型, 对其加入值抽取功能;
- RAT-SQL: Spider 数据集 2020 年的 SOTA 模型, 在 IRNet 模型基础上增加了对数据库表格之间关系的编码, 同 IRNet 实现都采用预训练语言模型 RoBERTa 的 base 权重进行编码;
- RAT-SQL+: 在 RAT-SQL 模型基础上加入 SQL 计算功能;
- Ours: 本文提出的基于树状模型的复杂自然语言查询转 SQL 系统框架, 分别使用 base 和 large 权重的 RoBERTa 预训练语言模型.

表 5 各模型完整生成复杂 SQL 的准确性实验对比

模型/全部准确率	验证集	测试集 1	测试集 2
Baseline	0.183	–	–
IRNet	0.723	–	–
RAT-SQL	0.745	–	–
RAT-SQL+	0.814	0.749	0.617
Ours	0.837	–	–
Ours(large)	**0.856**	**0.875**	**0.727**

Ours 系统框架是由上述 3 个模型组成的 Pipeline, 在全部准确率取得 0.837 的成绩. 可以看到: 虽然加入了值抽取模型, 最终生成 SQL 语句的完全匹配准确率相比不包含值的预测准确率 0.848 下降幅度不大. 这是因为本文将值抽取模型单独分离出来, 并基于统一的结构进行训练, 可以很好地利用训练集中所有包含值的

查询数据进行学习, 构建的模型具有良好的性能; 同时, 针对嵌套查询的副 SQL 样本较少的问题, 亦通过权重共享和再编码的机制进行优化, 使得其在复杂嵌套查询中表现稳定.

本文提出的Ours系统相比于其他模型在验证集上提升巨大, 相比于学术界主流的基于序列解码的复杂自然语言查询 SOTA 模型 IRNet 和 RAT-SQL 也有较大提升, 充分体现了 Ours 系统引入 SQL 树状分解模型的优势: 通过对复杂 SQL 建立统一体系的自顶向下分解模型, 将任意复杂 SQL 都可建模为若干可复用的根模块的形式进行训练和预测, 最终组装形成完整的 SQL. 该树状模型有效地提升了复杂查询的构建成功率.

实验还对 RAT-SQL 引入 DuSQL 中新增的计算查询类型的编解码支持, 并和 Ours 系统进行比较, Ours 系统仍然能取得明显的性能领先, 在两个测试集中均有10个百分点以上的效果提升且相比验证集上的提升更明显. 这充分体现了 Ours 系统引入表筛选模型和表信息增强算法对提高模型稳定性、防止过拟合造成跨领域数据下性能损失过大起到重要的作用.

在 large 参数配置下, 本框架在验证集、测试集 1 和测试集 2 上分别上取得了 0.856, 0.875 和 0.727 的成绩, 在 DuSQL 自然语言转 SQL 竞赛官方评测中取得了测试集 1 排名第 1、测试集 2 排名第 2 的成绩, 证明了整个技术框架解决中文复杂自然语言查询转 SQL 问题的可行性和有效性.

5 结论与展望

本文对复杂自然语言查询转 SQL 技术进行了探索, 研究如何让 NL2SQL 模型适配生成复杂多表的 SQL 语句, 并致力于解决低资源、跨领域情况下 NL2SQL 任务所面临的问题. 本文提出了一种针对复杂自然语言查询转 SQL 的通用技术框架, 该框架面向商业智能场景下各种嵌套查询、SQL 计算等复杂查询形式提出了一种复杂自然语言查询的 SQL 树状分解模型, 基于自顶向下的思想, 将 SQL 分解为多叉树的形式. 基于该树状模型的 SQL 预测模型可以有效支持多种形式的复杂查询, 同时提升复杂查询解析的准确率. 针对跨领域多表复杂查询的技术难点, 本文还为 SQL 的预测生成设计了两个辅助模型用于提升 NL2SQL 任务的性能表现和泛化能力, 形成由针对多表跨领域查询的预筛选模型、基于树状分解的 SQL 结构预测模型和针对嵌套 SQL 的值抽取模型这 3 个模型组成的层层递进的整体技术框架:

(1) 针对多表跨领域查询的预筛选辅助模型以及训练过程中的表数据增强算法, 能够使NL2SQL模型处理任意规模的数据库, 并且在低资源跨领域场景下表现更加稳定;

(2) 基于树状分解的 SQL 结构预测模型能够自顶向下逐层分解, 将树各层结点的复用模块进行有效组织, 同步输出不同查询类型复杂 SQL 的各基本组成模块结果;

(3) 针对嵌套 SQL 的值抽取辅助模型统一了不同的值抽取任务, 并支持了嵌套 SQL 中主副 SQL 的同时抽取, 可以充分利用好训练数据来提升值抽取模型表现.

实验结果表明, 本文提出的基于树状模型的复杂自然语言查询转 SQL 技术框架在中文 DuSQL 数据集上取得较好的性能表现, 充分验证了该方案的有效性, 具有一定的学术价值和现实意义.

然而在现实应用中, 由于自然语言交互的随意性, 自然语言查询任务依然面临诸多的挑战, 比如面向商业智能场景下的计算查询可能会不断出现新的概念、新的表模式无法被训练好的模型所理解造成预测性能下降(如实验中测试集 2 下的性能表现), 本工作在这些场景下依然存在局限性. 未来的研究将考虑在模型中引入外部知识对模型信息进行增强, 比如知识图谱、新实体描述信息等, 进一步优化整个解决方案. 此外, 当前的模型仅仅具备单轮生成 SQL 的能力, 面对实际交互场景中用户可能通过多次问答获取信息的需求, 未来的工作也将针对会话生成 SQL 进行研究, 以获得更高的实际应用价值.

References:

[1] Zhong V, Xiong C, Socher R. Seq2SQL: Generating structured queries from natural language using reinforcement learning. arXiv: 1709.00103, 2017.

[2] Yu T, Li Z, Zhang Z, *et al*. TypeSQL: Knowledge-based type-aware neural text-to-SQL generation. In: Proc. of the 2018 Conf. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. 2018. 588−594.

[3] He P, Mao Y, Chakrabarti K, *et al*. X-SQL: Reinforce schema representation with context. arXiv:1908.08113, 2019.

[4] Yu T, Zhang R, Yang K, *et al*. Spider: A large-scale human-labeled dataset for complex and cross- domain semantic parsing and text-to-SQL task. In: Proc. of the 2018 Conf. on Empirical Methods in Natural Language Processing. 2018. 3911−3921.

[5] Guo J, Zhan Z, Gao Y, *et al*. Towards complex text-to-SQL in cross-domain database with intermediate representation. arXiv: 1905.08205, 2019.

[6] Wang B, Shin R, Liu X, *et al*. RAT-SQL: Relation-aware schema encoding and linking for text-to-SQL parsers. arXiv:1911.04942, 2020.

[7] Sun N, Yang X, Liu Y. TableQA: A large-scale chinese text-to-SQL dataset for table-aware SQL generation. 2020. arXiv:2006. 06434, 2020.

[8] Wang L, Zhang A, Wu K, *et al*. DuSQL: A large-scale and pragmatic Chinese text-to-SQL dataset. In: Proc. of the 2020 Conf. on Empirical Methods in Natural Language Processing. 2020. 6923−6935.

[9] Min Q, Shi Y, Zhang Y. A pilot study for Chinese SQL semantic parsing. arXiv:1909.13293, 2019.

[10] Price PJ. Evaluation of spoken language systems: The ATIS domain. In: Proc. of the Workshop on Speech and Natural Language. 1990. 91−95. [doi: 10.3115/116580.116612]

[11] Tang L, Mooney R. Using multiple clause constructors in inductive logic programming for semantic parsing. In: Proc. of the European Conf. on Machine Learning. 2001. 466−477.

[12] Pasupat P, Liang P. Compositional semantic parsing on semi-structured tables. In: Proc. of the 53rd Annual Meeting of the Association for Computational Linguistics. 2015. 1470−1480.

[13] Xu X, Liu C, Song D. SQLNet: Generating structured queries from natural language without reinforcement learning. arXiv:1711. 04436, 2017.

[14] Hwang W, Yim J, Park S, *et al*. A comprehensive exploration on WikiSQL with table-aware word contextualization. arXiv:1902. 01069, 2019.

[15] Wang C, Brockschmidt M, Singh R. Pointing out SQL queries from text. Microsoft Research, 2018. https://www.microsoft.com/en-us/research/publication/pointing-sql-queries-text/

[16] Liu X, He P, Chen W, *et al*. Multi-task deep neural networks for natural language understanding. arXiv:1901.11504.80, 2019.

[17] Yu T, Yasunaga M, Yang K, *et al*. SyntaxSQLNet: Syntax tree networks for complex and cross-domain text-to-SQL task. In: Proc. of the 2018 Conf. on Empirical Methods in Natural Language Processing. 2018. 1653−1663.

[18] Lin XV, Socher R, Xiong C. Bridging textual and tabular data for cross-domain text-to-SQL semantic parsing. In: Proc. of the Findings of the Association for Computational Linguistics: EMNLP 2020. Association for Computational Linguistics, 2020. 4870−4888. [doi: 10.18653/v1/2020.findings-emnlp.438]

[19] Wu Y, Schuster M, Chen Z, *et al*. Google's neural machine translation system: Bridging the gap between human and machine translation. arXiv:1609.08144, 2016.

[20] Liu Y, Ott M, Goyal N, *et al*. RoBERTa: A robustly optimized BERT pretraining approach. arXiv:1907.11692, 2019.

[21] Kingma DP, Ba J. Adam: A method for stochastic optimization. arXiv:1412.6980, 2014.

[22] Srivastava N, Hinton GE, Krizhevsky A, *et al*. Dropout: A simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research, 2014, 15(1): 1929−1958.

[23] Ba JL, Kiros JR, Hinton GE. Layer normalization. arXiv:1607.06450, 2016.

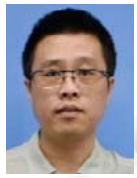
[24] Devlin J, Chang MW, Lee K, *et al*. BERT: Pre-training of deep bidirectional transformers for language understanding. In: Proc. of the Conf. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. 2019. 4171−4186.

赵猛(1997－), 男, 硕士, 主要研究领域为语义解析, 自然语言处理.

陈珂(1977－), 女, 博士, 副研究员, CCF 专业会员, 主要研究领域为数据库系统, 大数据技术, 隐私保护.

寿黎但(1974－), 男, 博士, 教授, 博士生导师, CCF 高级会员, 主要研究领域为数据库系统, 数据智能技术, 数据挖掘

伍赛(1980－), 男, 博士, 教授, 博士生导师, CCF 专业会员, 主要研究领域为分布式数据库检索和查询, 大数据分析处理, 基于机器学习的数据库智能化算法.



陈刚(1973－), 男, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为数据库系统, 大数据技术, 数据智能计算.