Text-to-SQL 泛化性探究与模型改进思路 2.0

日期：2023 年 4 月 21 日

摘　要

本次调研，我首先接着上次关于 Text-to-SQL 泛化性的知识点出发，发现了另外的一个可能的泛化性知识——跨领域 SQL 查询。后续我就 T4 知识点中模型需要学习到的映射能力举例说明。针对模型结构的改进部分，我阐述了上次调研的模态融合方面的两种创新的方法以及模型训练的细节。接下来，我将在这段时间的学习调研内的一些思考以较为口语化的杂谈形式呈现，提出了文本化数据库 Schema、关于大模型的一些思考以及训练方法的进一步思考。最后总结了设计的模型改进思路。

关键词: 泛化性，模型改进，思路杂谈

1　另外的泛化

在对 Spider-dk 数据集观察中，我发现数据集中包含风暴记录、航空公司、行凶者等领域各异的数据库。这也就让我想到对于**跨领域 SQL 查询**这个需要泛化能力的问题。既要求模型对人类语言有深刻理解——能理解带有专业领域特色的 NL 问题以及其中的领域词汇等；对数据库 Schema 有深刻的理解——能理解带有专业领域特色的数据库 Schema 不同表名，列名之间的联系以及含义；能联系起该领域的 NL 问题和数据库 Schema 信息。

2　对于 T4 知识的理解

T4 知识是论文 [2] 中提出的关于 Text-to-SQL 领域泛化性问题中，要求模型能区分数据库中一列数据是否为布尔值或者 N 元变量，要求模型可以解决含有此类查询需求的 SQL 语句生成的问题。

对于例子 “**NL**:How many students got accepted..... ; **SQL**:....where decision= ‘yes’” 我认为其中存在一定的映射关系。对于该例子所在数据库 soccer_2 中，decision 为 text 类型，原始列名 (column_names_original) 和解释性列名 (column_names) 都是 decision，该列所属原始表名 (table_names_original) 以及解释性表名 (table_names) 都是 tryout。我认为模型只需要从 **NL** 问题中的 accepted 映射到 tryout 表，后续进一步映射到其中的 decision 列即可。至于 T4 问题提到的辨认 decision 列数据为 N 元变量，且取值为 ‘yes’ 和 ‘no’，我则仍然认为应该是在解释性列名 (column_names) 上标注如：“decision，Binary variable，yes or no”。有了这点提示，模型则可以在之前的推理下继续深入，最终实现从 accepted 映射到 decision= ‘yes’。总的来说，对于这种涉及 T4 知识的问题，模型的基本能力应该可以实现从问题中的 got accepted 映射到正确的表中正确的列。而后续的模型能从 decision 的解释性列名 (column_names) 中学习到该列数据取值，并对应生成 decision= ‘yes’ 则是学到了 T4 知识的体现。对于这点，将完善后的解释性列名 (column_names) 与原始列名 (column_names_original) 以及其他数据 (表名、表间关系等) 一起作为数据库 Schema 模态的输入进行训练即可使模型学到。

如果有一个表格需要我们搜索通过考试的人，首先与 accepted 的例子相同，需要寻找相关的表以及表中的列，以及是否需要其他表的信息等，这里假设需要找的那个列名为 “do_pass”。根据之前学到的知识，模型会从 “do_pass” 修正后的解释性列名 (column_names) 中学习到其是一个二元变量，取值为 “T”

和 “F” [1]。由此生成 SQL 语句: “do_pass= ‘T’”。这即是从 accepted 例子中学到的知识的应用。在此，模型学到从解释性列名 (column_names) 判断该列变量性质即为泛化。

3 模型结构改进[2]

3.1 模态融合

加入 **Gate&Add 模块**，在多头交叉注意力机制计算结果 $\hat{H}$ 后将其与 query 模态隐变量 H^L 加权相加，以此平衡两个模态对于生成任务的贡献度，以达到更好的生成效果。

加入 **CNN，CBAM 提取**，在使用多头交叉注意力机制前，首先依次使用 CNN 和 CBAM 注意机制处理拼接后的文本特征和数据库特征，以减少干扰信息并增强网络对模式之间相关信息的关注，今进而提高特征表示能力。

上次提到这两种方法是为了说明，前沿领域除了使用多头交叉注意力机制外，还可以尝试实验的两种额外的方法以实现模态对齐，但这并不能解决上面提到的涉及 T4 知识的问题 (只有通过修正数据集，添加描述实现)。上述有关模态融合方面的工作或许可以提升模型在 Spider 以及 Spider-dk 数据集上的表现。

3.2 模型训练

接下来我会使用如图 1 所示的双塔结构的模型设计图举例阐述此部分的内容，其中模态融合部分使用多头交叉注意力机制。

初步定义的模型训练可以分为两部分，首先仅使用数据集中 NL 问题以及数据库 Schema 进行自监督学习实现模态 Encoder 分的参数学习[3]，实现模态对齐；后一部分，固定模态 Encoder 部分参数，使用 Spider 数据集进行有监督学习，训练模型的 Fusion 和 Decoder 部分参数。

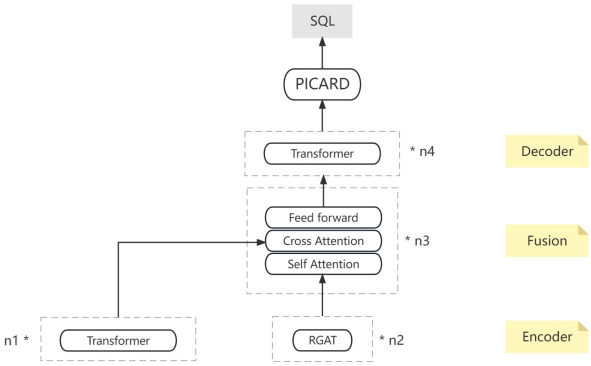

图 1: 初步设计的双塔结构

3.2.1 输入部分

对于 NL 问题部分的输入，考虑还是单独输入自然语言问题。但在对 Spider 数据集的观察时，发现其与 Spider-dk 类似，也同样有原始列名 (column_names_original) 和解释性列名 (column_names)，原始表

[1] 该假设的二元变量参考自 Spider-dk 数据集中一个名为 “If_first_show” 的列

[2] 先前我的用语并不准确，在此后参考 ALBEF[5] 做出统一修改，定义在图一中 Encoder 部分实现模态对齐，在 Fusion 部分实现模态融合

[3] 后续会有补充另外一个训练任务 RSP，需要使用完整 Spider 数据集

名 (table_names_original) 以及解释性表名 (table_names)，且对于主键，外键则含有相同的表示方法。对于该部分，考虑在每个表、列的节点中手动设置 prompt 模板 eg: 原本的列节点为“do_pass”后续可以改进为“do_pass(do the person passed yes or no)[4]”。

3.2.2 Encoder 训练

对于模型编码部分，考虑到此处的 NL 问题和数据库 Schema 与图文多模态以及代码智能领域中模态间关系不同，text-to-SQL 领域的文本和数据库两模态含有信息并不相同，故不使用 ALBEF 中的图文对比学习与掩码训练。采用如图 2 所示的匹配预测训练，获取文本和数据库 Schema 的 CLS 头，concat 拼接后使用额外的全连接分类层，判断该 NL 问题和数据库 Schema 是否匹配。由此训练实现 NL 问题和数据库

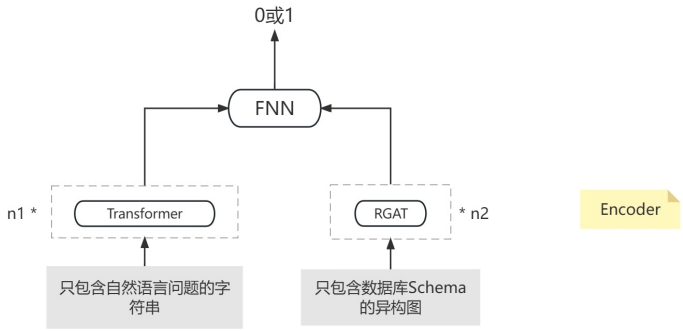

图 2: Encoder 部分训练

Schema 两个模态之间的对齐。后续保存两 Encoder 部分参数即可。

3.2.3 Fusion 和 Decoder 的训练

对于该部分，计划使用预训练的 Encoder 部分参数并在后续不再更新，使用完整的 Spider 数据集优化 Decoder 部分和 Fusion 部分参数。该处则与 Graphix-T5[4] 模型中训练方式相同，同样使用生成的 1 到 t-1 个 SQL 语句中的元素预测第 t 个元素，并最大化如下的对数似然：

$$\max_{\Theta} \log p_{\Theta}(y \mid x) = \sum_{i=1}^{|y|} \log p_{\Theta}\left(y_i \mid y_{1:i-1}, x, \mathcal{G}\right)$$

其中 Θ 为 Fusion 和 Decoder 部分模型参数，x 为输入的 NL 问题，y 为待生成的 SQL 语句，y_i 为待生成的 SQL 语句中第 i 个单词，$\mathcal{G}$ 为数据库 Schema 的异构图输入。

4 其他思考杂谈

4.1 文本化数据库 Schema

在设计本文 NL 问题与数据库 Schema 之间的模态对齐时突然联想到之前多轮 Text-to-SQL 问题模型 MIGA[1] 中直接将数据库 Schema 如字符串类型直接拼接作为输入，类似：

$$x = \left[t_1 : c_1^{t_1}, \ldots, c_{|C|}^{t_1} | \ldots | t_{|\mathcal{T}|} : c_1^{t_{|\mathcal{T}|}}, \ldots, c_{|C|}^{t_{|\mathcal{T}|}} \mid \ldots . \right]$$

[4]括号内部分为该列名对应的描述性列名，括号即为此处设置的 prompt 模板

MIGA[1] 中也并未做任何模态融合的操作，对此我想试着如果通过手工设定 prompt 模板优化数据库 Schema 部分输入，将结构化的数据库 Schema 数据用自然语言进行描述，直接与 NL 问题通过另一 prompt 模板拼接，以单一模态的形式使用最原始的预训练模型 T5 进行微调。

例如，使用如下自然语言描述一个数据库 Schema：

Name of database："new_orchestra". Table "conductor(conductor[5])" has columns："Conductor_ID(conductor id[6])), type is text, it is the primary key", "Name_len(name length), type is number" .....; table "orchestra(orchestra)" has columns："Is_Good(is good yes or no), type is text" ..... . Column "A(A's descriptive name)" is Table "T1(T1's descriptive name)" is the foreign key relative to Table "T2(T2's descriptive name)".

例子中黑色内容为人为设置的 prompt 模板，红色内容则可以通过 Spider 数据集中 table.json 中数据生成插入。由此通过流程式的算法程序实现数据库 Schema 与 NL 问题之间的模态对齐，此思路的模型结构如图 3 所示。

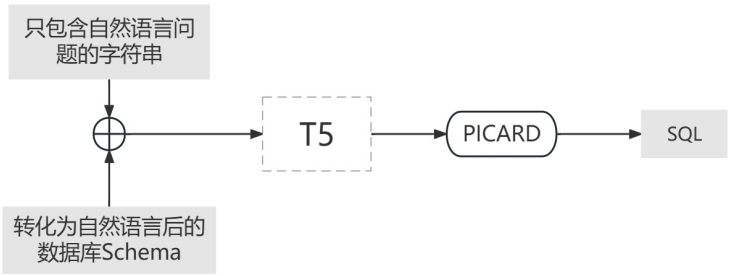

图 3: 转化 SQL 为自然语言的模型思路

在之前的调研记录中并未发现使用此类思路的模型。

4.2 在大模型时代 Text-to-SQL 领域的发展

关于这点的思考，一是来自我每周一去敝校Data Mining 实验室旁听组会收获的一点 idea，二是来自与一个同学的交谈中引发的思考。

上周组会时，就有一些研二以及博士学长学姐分享了他们关于大模型的论文调研，在当时已经有了一些关于大模型推理能力、共情能力等方向的研究了，也有学长尝试使用大模型辅助为数据打标签。这周的组会有一位研一的学长介绍了一篇论文，模型中一部分是使用一个大模型 (可能是 GPT3，有点记不清了) 来参与生成一部分内容 (并不是知识蒸馏类似的作为教师模型)，然后将输出继续传入下一部分。其中大模型不参与参数更新。这部分一开始给我的思路就是在 4.1 中使用大模型将数据库 Schema 转化为自然语言文本，但后续就发现可以使用 prompt 模板插槽的方式代替。但是反而我又直接尝试完全使用 prompt 让基于 GPT3.5 的 ChatGPT 解决单轮 Text-to-SQL 领域的问题[7]，随即使用 Spider 数据集中一个例子，通过先给 ChatGPT 描述该数据库，然后再输入 NL 问题，让 ChatGPT 生成 SQL 查询语句，测试结果如图 4 所示，结果发现 ChatGPT 能较优得解决 Text-to-SQL 领域知识，说不定可以借此超过当前 Graphix-T5[4] 模型的 EM(SQL 匹配)、EX(SQL 执行结果) 准确率。

与我交谈的那个同学是做代码智能领域的，他的指导老师近期让他开始调研大模型，在准备后续做代码智能这方面的大模型。这倒给我一点提醒，说不定 Text-to-SQL 领域后续也可以做出大模型。当前

[5]该部分是表名的描述性表示，是来自 table_names 的数据

[6]该部分同理，是列名的描述性名称

[7]对此使用他人代理的一个ChatGPT 网站，选择了两个样本测试

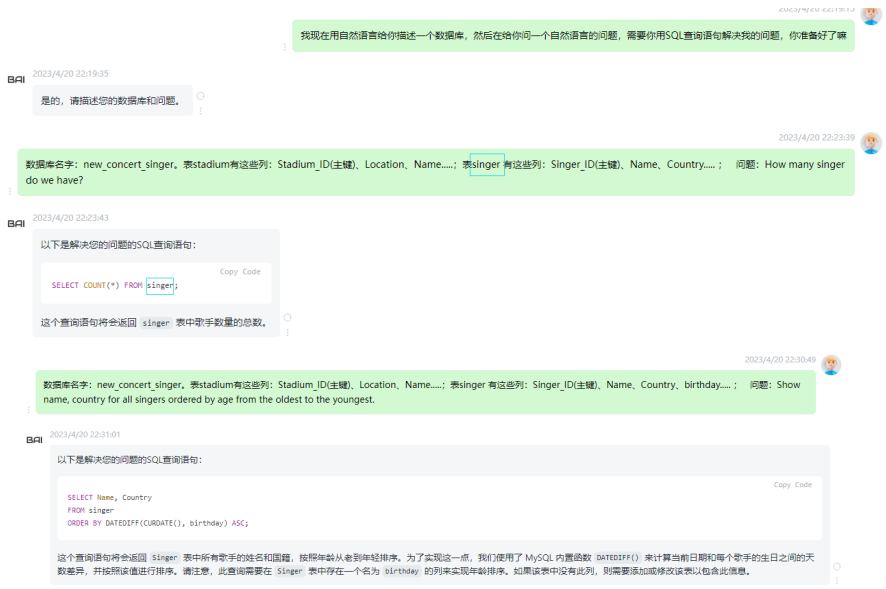

图 4: 完全基于 prompt 测试 ChatGPT 在解决 Text-to-SQL 单轮问题的能力

Text-to-SQL 领域使用的最大模型基本为 T5-3B，含有 30 亿参数，相比 GPT-3 的 1750 亿参数仍然有较大差距。而为了训练 Text-to-SQL 领域的大模型，最先面临的问题就是数据量的问题，目前对于 Text-to-SQL 领域常用的英文数据集只有 Spider、Wikisql、TableQA 等，并不足以训练大模型。对此我的一个初步想法是参考计算机视觉领域大模型 Segment Anything[3] 中，使用现有模型、现有数据生成新的数据，然后使用新的数据训练模型，然后再次生成新的数据的方式，但该方法我还没详细调研，因此并未过多阐述。

4.3 关于训练方式的思考

在本次详细设计模型训练方式时，联想到先前调研过的多轮 Text-to-SQL 问题的 sota 模型 MIGA[1] 中使用的多种模型预训练方法，就再次去查阅可以应用到单轮 Text-to-SQL 领域的学习方法，计划在 Encoder 部分与最初设计的匹配预测训练一起训练，实现两模态之间的对齐。

随后调研发现，有 Related Schema Prediction(RSP) 方法可以借鉴。该方法需要预测出 SQL 结果中将要出现的列和表，而不用考虑 SQL 结构，作者认为这可以缓解在 SQL 中生成错误的表和列，我认为这个任务可能让模型学会使用 NL 问题在数据库 Schema 中做简单查询，能建立两个模态之间的关联，或许有利于实现模态对齐。

5 模型改进总结

在对 Graphix-T5[4] 的模态融合部分的改进中，新模型与 Graphix-T5[4] 差距越来越大，原本 Graphix-T5[4] 的一个初衷就是尽可能多的使用预训练模型 T5 的参数，避免出现类似在使用 GNN 切断 T5 的 GNN-T5 模型中出现的灾难性遗忘 (学习退步) 问题，而我这里改进了属于是得重新训练除了 NL 文本 Encoder 部分的其余所有参数了。后续在一步步细化模型架构中改进，最后确定首先使用 3.2.2 提到的匹配预测 (TSM) 和本节中提到的关系模式预测 (RSP) 训练模型的 Encoder 部分，且尽可能调小 NL 文本的学习率，或者不优化该部分参数，在尽可能使用 T5 原始参数的同时使数据库 Schema 模态向 NL 文本模态靠拢。训练结束后，进行整体 SQL Generation(SG) 训练，并且不优化 Encoder 部分参数，仅优化 Fusion 和 Decoder 部分的参数，实现模型的有效生成[8]。

[8]PICARD[6] 部分不需要参与训练

参考文献

[1] Yingwen Fu et al. *MIGA: A Unified Multi-task Generation Framework for Conversational Text-to-SQL*. 2022. arXiv: 2212.09278 [cs.CL].

[2] Yujian Gan, Xinyun Chen, and Matthew Purver. *Exploring Underexplored Limitations of Cross-Domain Text-to-SQL Generalization*. 2021. arXiv: 2109.05157 [cs.CL].

[3] Alexander Kirillov et al. *Segment Anything*. 2023. arXiv: 2304.02643 [cs.CV].

[4] Jinyang Li et al. *Graphix-T5: Mixing Pre-Trained Transformers with Graph-Aware Layers for Text-to-SQL Parsing*. 2023. arXiv: 2301.07507 [cs.CL].

[5] Junnan Li et al. *Align before Fuse: Vision and Language Representation Learning with Momentum Distillation*. 2021. arXiv: 2107.07651 [cs.CV].

[6] Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. *PICARD: Parsing Incrementally for Constrained Auto-Regressive Decoding from Language Models*. 2021. arXiv: 2109.05093 [cs.CL].