计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.190800138

基于字段嵌入的数据库自然语言查询接口

田 野[1] 寿黎但[1,2] 陈 珂[1,2] 骆歆远[1,2] 陈 刚[1,2]
1 浙江大学计算机科学与技术学院 杭州 310027
2 浙江省大数据智能计算重点实验室 杭州 310027
(tianye_zju@zju.edu.cn)

摘 要 将自然语言转化成数据库可以执行的查询语句,是目前智能交互和人机对话系统的核心难题,也是新型供电列车大数据运用支撑平台对接应用平台及建立城轨列车个性化运维系统的难点。现有的基于神经网络的方法没有充分利用数据表的丰富信息,影响了查询的准确率。针对数据表内容作为输入的情况下,如何提升自然语言查询接口的查询准确率的问题,文中创新地提出了基于数据表内容的字段嵌入方法,利用数据表中每个字段存储的内容对字段进行嵌入表示,并据此提出了新的模型嵌入层结构;此外,提出了一种基于数据表内容的数据增强方法,通过用数据表相同字段中的其他记录去代替查询语句中的属性值,来产生新的训练样本。最后,针对提出的字段嵌入表示和数据增强方法,在 WikiSQL 数据集上进行了对比实验。实验结果显示,相比当前效果最好的模型,单独使用这两种方法时能够提升 0.6%~0.8%的查询准确率,共同使用时则能够提升接近1%的查询准确率,证明所提字段嵌入和数据增强方法对查询准确率有一定的提升作用。

关键词:数据库查询;自然语言处理;SQL;词嵌入
中图法分类号 TP391.1

Natural Language Interface for Databases with Content-based Table Column Embeddings

TIAN Ye[1],SHOU Li-dan[1,2],CHEN Ke[1,2],LUO Xin-yuan[1,2] and CHEN Gang[1,2]
1 College of Computer Science and Technology,Zhejiang University,Hangzhou 310027,China
2 Key Laboratory of Big Data Intelligent Computing of Zhejiang Province,Hangzhou 310027,China

Abstract Converting natural language into query statements that can be executed in database is the core problem of intelligent interaction and human-computer dialogue system,and is also the urgent need of personalized operation and maintenance system for urban rail trains. At the same time,it is the difficulty of docking the bottom application platform with the support platform for large data application of the new power supply train. The existing neural network-based methods don't utilizing semantic-rich table content or utilize it partially,which limits the improvement of the execution accuracy. This paper studies how to improve the query accuracy of natural language query interfaces when table content is included in the inputs. Aiming at this problem,this paper proposes a table column embedding method based on table content which embeds the table columns by utilizing the content stored in each table column. Based on the method,this paper proposes a new structure of embedding layer. This paper also proposes a method of data augmentation by utilize table content. It generates new training samples by replacing attribute values in queries with other records in the same column of the table. This paper finally conducts experiments on WikiSQL dataset for the proposed methods of column embedding and data augmentation. The experimental results show that,on the basis of the state-of-the-art methods,the two methods can improve the query accuracy by 0.6%~0.8% when they are used separately and nearly 1% when they are used together. Therefore,it proves that the methods of column embedding and data augmentation proposed in this paper can achieve good improvements on execution accuracy.

Keywords Database query,Natural language processing,SQL,Word embedding

到稿日期:2019-08-28 返修日期:2019-10-14 本文已加入开放科学计划(OSID),请扫描上方二维码获取补充信息。
基金项目:国家重点研发计划(2017YFB1201001);国家自然科学基金(61672455);浙江省自然科学基金(LY18F020005)
This work was supported by the National Key R&D Program of China (2017YFB1201001),National Natural Science Foundation of China (61672455) and Natural Science Foundation of Zhejiang Province,China (LY18F020005).
通信作者:陈珂(chenk@zju.edu.cn)

1　引言

城轨列车个性化运维系统是新型供电轨道交通系统的重要组成部分。为了提高系统的易用性，把自然语言转化为数据库可执行的查询语句是当前迫切需要解决的问题。传统的数据库查询方式是通过编写满足特定语法的查询语句进行查询，然而这种方法要求查询者高度熟悉数据库的结构和语法，而且需要耗费时间来构建查询语句。与传统查询方式相比，把自然语言转化为数据库查询语句，能够使查询者无须了解数据库的具体结构和查询语法，快速地实现查询意图。因此，构建数据库的自然语言查询接口，是城轨列车个性化运维系统建设的迫切需求，该问题的解决对智能交互、人机对话等其他领域的发展也有着重要意义。

近年来，随着人工智能的快速发展，对数据库的自然语言接口方面的研究主要为基于神经网络的方法。这类方法通常将自然语言查询语句和相关数据表的字段名作为模型输入，通过序列-序列[1-2]或者基于槽填充[3-4]的模型结构，输出对应的 SQL 查询语句。然而，与 SQL 查询语句相比，自然语言查询语句很少显式指定数据表的字段名，通常将字段名省略，或者使用其他方式表达，这给自然语言向 SQL 语句的转换带来了很大挑战。针对该问题，研究者们通过利用外部的知识库[4]、概念网络[5]以及预训练语言模型[6]等额外知识增强对自然语言问句和字段之间联系的学习，取得了不错的效果。但是，作为数据库本身的组成部分，数据表的内容在这些工作中没有被充分利用。实际上，数据表的内容在数据库的自然语言查询接口中具有重要作用，原因如下：

(1)利用外部的知识库等信息进行语义编码，对网络和存储空间有很大的要求，很多应用场景不具备此条件，而利用本地的数据表内容可以在不需要额外资源的情况下达到良好的语义编码效果；

(2)由于字段名命名的灵活性和用户表达习惯的影响，自然语言查询往往不会显式地指定字段名，而是通过属性值隐式地指定字段，因此使用每个字段存储的内容来表示这个字段的语义更合理；

(3)数据表的内容规模一般很大，可以用于扩充数据集，增强模型的泛化性。

为了在将数据表内容作为输入的条件下，提高自然语言向 SQL 语句转换的准确率，本文提出了一种基于字段嵌入的数据库自然语言接口技术，主要的贡献概括如下：

(1)提出了一种基于数据表内容的字段嵌入方法，并将其结合到基于槽填充的模型中，有效地增强了模型对数据表字段的语义理解能力；

(2)提出了一种基于数据表内容的数据增强方法，使用扩充后的训练集训练模型，能够提升模型的准确率和泛化性；

(3)在 WikiSQL 数据集上进行了充分的对比实验，实验结果显示，本文提出的字段嵌入和数据增强方法，使模型在测试集上的查询准确率得到了一定的提升。

2　相关工作

数据库的自然语言接口在近几十年一直是数据库领域的热点问题。早期的研究[7]大多是针对特定的数据库，人工构建匹配规则，完成自然语言向 SQL 查询的转换。这种方法不仅耗费人力，而且不具有移植到其他数据库的能力。近年来，随着神经网络技术的广泛应用，更多的学者使用神经网络来解决该问题。Zhong 等[2]提出了利用强化学习的序列-序列模型 Seq2SQL，根据 SQL 语句中的不同成分划分槽位，用基于槽填充的方法生成 SQL 语句，并发布了 WikiSQL 数据集。该模型不受限于特定的数据表结构，具有迁移到其他数据库的能力。Xu 等[3]在 Seq2SQL 的基础上提出了 SQLNet，其使用了序列-集合模型以及基于字段的注意力机制，在不使用强化学习的情况下大幅提升了准确率。Yu 等[4]在此基础上提出了 TypeSQL，利用外部的知识库对问句和字段的类型信息进行编码，并考虑不同子任务间的影响，对模型结构进行了改动，取得了良好的效果。2018 年以来，随着预训练语言模型的发展，更多的研究者将预训练语言模型应用到此任务上，在 WikiSQL 上达到了最好的效果，如结合了 BERT 模型[8]的 SQLova[6]。

以上工作重点关注的是在无法获取数据表内容的情况下如何对自然语言查询进行转换，因此很多方法借助了外部的知识来解析语义。对于在能够获取数据表内容的情况下使用神经网络的方法进行转换，已有工作不多，对数据表内容的利用也比较片面。

Petrovski 等[9]以数据表内容为字段来训练 Word2Vec 向量，并替换 SQLNet 模型中的字段名称嵌入，但是模型在 WikiSQL 上的准确率有所下降。本文利用数据表内容进行嵌入表示的方法更加快速、有效，而且认为字段名称也可以提供有用的信息，因此同时使用字段名称和数据表内容对字段进行嵌入表示。

Sun 等[10]在序列-序列模型的解码阶段考虑了数据表字段和记录的相互影响，将每个字段记录的隐层向量加权求和之后拼接到字段的隐层向量中，同时保留最新预测出的字段，使属性值的预测倾向于此字段下的记录。本文将数据表内容应用到基于槽填充的模型中，而且对自然语言问句和数据表字段都进行了基于数据表内容的嵌入表示。

Yavuz 等[11]关注现有方法在 WikiSQL 数据集上预测查询条件的准确率不高的问题，针对 WHERE 子句预测，直接在数据表记录中匹配搜索属性值得到候选。实际上，数据表内容对所有子句的预测都有影响，因此不仅需要关注 WHERE 子句，更需要从整个模型的角度进行实验和分析。

Yu 等[4]提出了 TypeSQL＋TC 模型，与使用实体类型信息的 TypeSQL 相比，该模型使用数据表内容为查询语句进行嵌入表示，准确率得到大幅提升。与查询语句类似，对字段进行类型编码能更加充分地体现字段的语义；同时，利用更大范围的数据表内容进行数据增强也能提升模型效果。

3　问题定义

为了更准确地描述本文所研究的数据库自然语言查询接口问题，现给出该问题所涉及相关概念的具体形式化定义。

3.1　问题输入

首先对问题的输入进行定义。输入一共由 3 个部分组

成，分别是自然语言查询语句、数据表结构和数据表内容。

定义 1（自然语言查询语句(Q)） 能够表明查询意图的自然语言祈使句或疑问句表示为 $Q=\{w_1,w_2,\cdots,w_k,\cdots,w_{nq}\}$。其中，$w_k$ 表示组成问句 Q 的第 k 个词语，n_q 表示 Q 中词的个数。

定义 2（数据表结构(T_s)） 要查询的数据表的字段名称集合表示为 $T_s=\{col_1,col_2,\cdots,col_j,\cdots,col_{n_s}\}$。其中，$col_j$ 表示数据表 T 中的第 j 个字段的名称，n_s 表示 T 中字段的个数。

定义 3（数据表内容(T_c)） 数据表中以行为单位存储的记录集合表示为 $T_c=\{row_1,row_2,\cdots,row_i,\cdots,row_{n_c}\}$。其中，$row_i$ 表示数据表 T 中的第 i 个记录，$row_i=\{val_{i1},val_{i2},\cdots,val_{ij},\cdots,val_{in_s}\}$，$val_{ij}$ 表示 T 中第 i 个记录的第 j 个字段存储的数据；n_c 表示数据表 T 的记录数。

3.2 问题输出

本节对问题的输出进行定义。输出，即 SQL 查询语句。

定义 4（SQL 查询语句(R)） 符合 SQL 语法的 SELECT 语句。本文所研究的查询为指定数据表的单表查询，且不考虑 Group By，Order By 等复杂子句，因此本文中输出的 SQL 查询语句 R 满足图 1 所示的形式，图中的“*”表示 0 个或多个。

```
SELECT $AGG $COLUMN
FROM T
WHERE $COLUMN $OP $VALUE
(AND $COLUMN $OP $VALUE)*
```

图 1 本文研究的 SQL 语句形式

Fig. 1 Form of SQL query discussed in this paper

3.3 问题描述

最后在上述概念的基础上，给出对数据库自然语言接口这一问题的具体定义。

定义 5（数据库的自然语言查询接口） 在给定数据表结构和数据表内容的基础上，将自然语言查询语句转化为对应的 SQL 查询语句的问题，其形式化定义为：

$$f:Q\xrightarrow{(T_s,T_c)}R \tag{1}$$

4 算法设计

4.1 模型结构

本文针对上述数据库自然语言查询接口问题，提出了如图 2 所示的模型嵌入层结构。

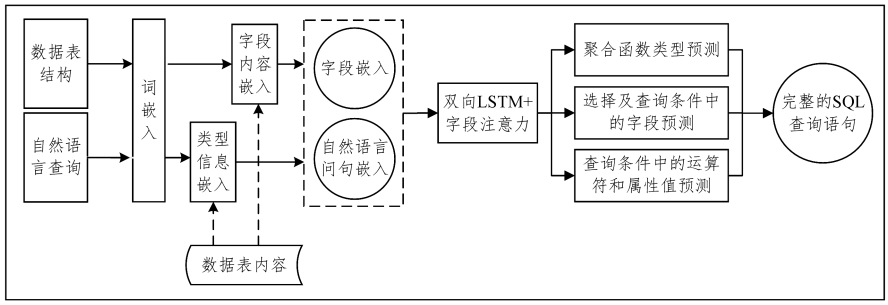

图 2 本文使用的模型结构

Fig. 2 Model architecture in this paper

模型结构沿用了典型的基于槽填充的方法，即根据 SQL 语句的结构特点，将 SQL 语句生成问题转换为预测目标槽位取值的多个子问题。具体来说，首先根据输入的问句和字段名称集合，结合数据表内容对输入进行嵌入表示，形成嵌入层的输出；然后，对于每个子问题，通过双向长短期记忆神经网络（Long Short-Term Memory，LSTM）对输入进行编码，并建立基于注意力机制的序列-集合模型或者指针网络（Pointer Network）[12]，完成对 SQL 语句中不同子句元素的预测；最后根据多个子模型的预测结果生成最终的 SQL 查询语句。除了在嵌入层使用数据表内容之外，本文还提出了一种利用数据表内容对训练集进行数据增强的方法。

4.2 结合数据表内容的嵌入表示

4.2.1 字段嵌入

本文认为字段也需要类似的类型编码来更加充分地体现字段的语义，同时利用更大范围的数据表内容进行数据增强也能提升模型效果。

首先，字段所代表的语义信息与字段名有关。因为如果查询涉及到数据表中的某个字段，自然语言查询语句中可能会出现与该字段的名称语义相同或相近的词语。因此，对于字段名称 $col_j=\{w_{j1},w_{j2},\cdots,w_{jk}\cdots,w_{jnj}\}$，我们对这 n_j 个词语的词向量进行平均，得到 col_j 的字段名称嵌入。使用 WE 代表计算词向量的函数，则字段名称嵌入 NE 按以下公式计算：

$$NE(col_j)=\frac{1}{n_j}\left(\sum_{k=1}^{n_j}WE(w_{jk})\right) \tag{2}$$

其次，字段所代表的语义信息与该字段存储的内容有很大关系。一方面，字段名本身可能不包含语义信息，如“col1”“col2”作为字段名；另一方面，用户在进行查询时通常按照人的表达习惯组织语言，可能不会显式地指定字段名，这种情况下，自然语言查询通过指定属性值隐式地确定字段，那么与字段发生直接语义关联的则是查询语句中的属性值。为了能够让字段的表示体现出这种关联性，考虑到查询中出现的属性值和数据表中对应字段存储的内容总是语义相似的，我们利用数据表中的字段内容对字段进行嵌入表示，具体方法如下。

（1）将数据表 T 存储的内容 $T_c=\{row_1,row_2,\cdots,row_i,\cdots,row_{n_c}\}$ 改成按列组织的形式，即 $T_{c\_col}=\{colv_1,$

$col v_2,\cdots,col v_j,\cdots,col v_{n_s}\}$，其中 $col v_j$ 表示第 j 字段所存储的数据集合，$col v_j=\{val_{1j},val_{2j},\cdots,val_{ij},\cdots,val_{n_r j}\}$，$val_{ij}$ 表示 T 中第 i 个记录的第 j 个字段的字段内容。

(2)对于每个 $col v_j$，随机选择 $\min(n_c,10)$个不同的 val_{ij}，并对它们进行分词，将分词得到的所有单词加入集合 S_j。

(3)对于每个 $S_j=\{vw_{j1},vw_{j2},\cdots,vw_{jk},\cdots,vw_{jm_w}\}$，我们对这些单词的词向量进行平均，得到每个字段的字段内容嵌入 VE。

$$VE(col_j)=\frac{1}{n_w}\left(\sum_{k=1}^{n_w}WE(vw_{jk})\right) \tag{3}$$

最终的字段嵌入由上述字段名称嵌入 NE 和字段内容嵌入 VE 拼接而成：

$$E(col_j)=\{NE(col_j);VE(col_j)\} \tag{4}$$

4.2.2　自然语言查询语句的嵌入

自然语言查询语句的嵌入由词嵌入和类型信息嵌入两部分构成。

在词嵌入 $WE(Q)$部分，与字段名不同，对于自然语言查询语句 Q，在分词时考虑了数据表内容，即列举出 Q 中长度不大于 6 的单词序列，在数据表记录中和数据表字段名中进行检索，对于完全匹配到的词组，将其划分为一个词，在计算词向量时对词组中的每一个词取平均。

在类型信息嵌入 $TE(Q)$部分，我们使用了 TypeSQL+TC[4]的思想，根据上一步的分词结果，如果某个词组是在字段名中匹配到的，则将其标记为 *column*，如果是在数据表记录中匹配到的，则将其标记为所在字段的字段名，否则将其标记为 None。将该标记作为类型信息，对每个标记求词向量，得到问句 Q 的类型信息，将其嵌入 $TE(Q)$。

对于输入的由 n_q 个词组成的自然语言查询语句 $Q=\{w_1,w_2,\cdots,w_k,\cdots,w_{n_q}\}$，每个位置的嵌入表示由该位置单词的词嵌入 WE 和类型信息嵌入 TE 拼接而成。

$$E(Q_k)=\{WE(Q_k);TE(Q_k)\} \tag{5}$$

4.3　槽填充模型

得到嵌入层的输出之后，模型使用 TypeSQL[4]中提出的槽填充模型对 SQL 查询语句的成分进行预测。根据预测槽位的不同，使用如下 3 个子模型。

(1)选择和查询条件中的字段预测模型。该模型预测 3 个部分：SELECT 子句中的字段、WHERE 子句的个数和 WHERE 子句的字段。

(2)聚合函数类型预测模型。该模型负责预测聚合查询的类型。

(3)查询条件预测模型。该模型预测每个 WHERE 子句中的运算符和属性值。

这 3 个子模型都有 2 个双向 LSTM 构成的编码层，分别对输入的自然语言问句嵌入和字段嵌入进行编码，得到 H_Q 和 H_{COL} 作为输出。然后通过基于字段的注意力机制，得到在给出 H_{COL} 的条件下对问句 Q 的表示：

$$\alpha_{Q|COL}=\text{softmax}\left(H_{COL}W_{CA}H_Q^{\mathrm{T}}\right) \tag{6}$$

$$H_{Q|COL}=\alpha_{Q|COL}H_Q \tag{7}$$

得到 $H_{Q|COL}$ 之后，根据预测目标的不同，使用不同的模型结构进行预测。

(1)选择和查询条件中的字段预测模型一共有 3 个子任务，它们都是分类问题。其中，预测 SELECT 子句中字段的公式如下：

$$s=U^{sel}\tanh(W_{col}^{sel}H_{COL}^{\mathrm{T}}+W_q^{sel}H_{Q|COL}^{\mathrm{T}}) \tag{8}$$

$$P_{\text{sel}}=\text{softmax}\ (s) \tag{9}$$

其中，P_{sel}表示预测的每个字段位于 SELECT 子句中的概率。

我们把预测 WHERE 子句的个数这一任务，处理成一个分类问题，这里只考虑 WHERE 子句在 4 个以内的情况，因此从 0～4 这些类别中进行预测。P_{num}表示预测的每种结果的概率：

$$P_{\text{num}}=\text{softmax}(U^{num}\tanh(W_q^{num}\sum_j H_{Q|col_j}^{\mathrm{T}})) \tag{10}$$

因为 SELECT 子句中的字段一般不会出现在 WHERE 子句中，所以在预测 WHERE 子句的字段时需要考虑 SELECT 子句的影响。假设 SELECT 子句中字段的预测结果是 *SCOL*，我们用上面提到的基于字段的注意力机制得到问句 Q 的条件表示 $H_{Q|SCOL}$，然后计算得到每个字段在 WHERE 子句中的概率：

$$c=U^{cond}\tanh(W_{col}^{cond}H_{COL}^{\mathrm{T}}+W_q^{cond}H_{Q|COL}^{\mathrm{T}}+W_q^{scol}H_{Q|SCOL}^{\mathrm{T}}) \tag{11}$$

$$P_{\text{cond}}=\text{softmax}\ (c) \tag{12}$$

(2)聚合函数类型预测模型的任务是从{NULL，MAX，MIN，SUM，AVG，COUNT}这 6 个聚合查询的关键词中预测一个，这是一个分类问题。模型采用以下公式：

$$P_{\text{agg}}=\text{softmax}\ (U^{agg}\tanh(W_q^{agg}H_{Q|COL}^{\mathrm{T}})) \tag{13}$$

(3)查询条件预测模型是对预测出的每个 WHERE 子句中的字段进行运算符预测和属性值预测。其中运算符预测是一个分类问题，考虑{=，>，<}3 种运算符，P_{op} 代表这 3 种运算符的概率：

$$P_{\text{op}}=\text{softmax}\ (U^{op}\tanh(W_{col}^{op}H_{COL}^{\mathrm{T}}+W_q^{op}H_{Q|COL}^{\mathrm{T}}) \tag{14}$$

属性值预测使用 Pointer Network 的思想，在问句 Q 首尾补充表示开始和结束的字符。解码过程的每一步都在问句 Q 中选择一个单词，直到选择出结束字符后停止。在每次解码过程中，选择问句 Q 中第 k 个词的概率为：

$$v=U^{val}\tanh(W_q^{val}H_Q^k+W_{col}^{val}H_{COL}+W_h^{val}h) \tag{15}$$

$$P_{\text{val}}=\text{softmax}\ (v) \tag{16}$$

其中，h 为上一个预测出的词的隐藏层。

4.4　结合数据表内容的数据增强

我们不仅用数据表内容来增强自然语言问句和字段的语义表示，还通过结合数据表内容提出了一种数据增强的策略。数据增强是指通过对已有训练数据进行变换操作，产生更多的训练集来训练模型。数据增强的意义体现在两方面：1)现有模型对 WHERE 子句预测的效果有待提高，因为 WHERE 子句的预测比较复杂，可能存在因为训练数据量不够而过拟合的问题；2)增加的训练集中问句的多样性大大增强，进一步减少了整个模型的过拟合问题，能够使模型在测试集和验证

集上的表现更加接近。

本文数据增强采用的基本思想是：用数据表中存储的同一字段的其他内容替换训练样本中的属性值，产生新样本。这种替换能够在保证新样本正确性的情况下，丰富训练集蕴含的语义信息。

(1)首先将数据表内容 T_c 修改成按列组织的形式 $T_{c\_col}=\{colv_1, colv_2, \cdots, colv_j, \cdots, colv_{n_s}\}$，其中 $colv_j$ 表示所有记录在第 j 个字段下存储的数据集合；然后对训练集中的每个样本，进行步骤(2)－步骤(5)的操作。

(2)从训练样本中抽取出四元组 $S=\langle Q, R, Cond\_Col, Cond\_Val\rangle$，其中自然语言查询语句 $Q=\{w_1, w_2, \cdots, w_k, \cdots, w_{nq}\}$，$R$ 表示 Q 对应的 SQL 查询语句，WHERE 子句的字段集合 $Cond\_Col=\{col_1, col_2, \cdots, col_j, \cdots, col_{mum}\}$，对应的 WHERE 子句的属性值集合 $Cond\_Val=\{val_1, val_2, \cdots, val_j, \cdots, val_{mum}\}$。

(3)对于 $Cond\_Val$ 中的每个 val_j，如果能够在问句 Q 中唯一地匹配到同样的字符串，则将 $Cond\_Col$ 中的 col_j 加入可替换字段集合 S_{replace}。

(4)若 S_{replace} 的值为 1，则从其中唯一的字段 col_j 对应的记录集合 $colv_j$ 中选择 α 个不同的记录去替换 Q 和 R 中的属性值 val_j，产生 α 个新的训练样本。

(5)若 S_{replace} 的值大于 1，则从 S_{replace} 中的每个字段 col_j 对应的记录集合 $colv_j$ 中随机选择 β 个不同的记录作为该字段的替换值集合。循环以下过程 β 次：对于 S_{replace} 中所有的字段，随机从替换值集合中取出 1 个并且不放回，用每个字段 col_j 本次取出的值去替换 Q 和 R 中的属性值 val_j。此过程将产生 β 个新的训练样本。

因为训练样本中存在有多个 WHERE 子句的情况，这种情况下能够利用的数据表内容更多，样本的变化更加丰富，产生的新样本质量更高，所以上述算法要对 WHERE 子句数量多于 1 个的训练样本产生更多的新样本。我们设置 $\alpha=3$，$\beta=5$。

5 实验及结果

5.1 数据集

使用 WikiSQL 数据集[2]作为实验数据集。如表 1 所列，WikiSQL 数据集提供了大量的数据表，并针对每个数据表提供人工标注的自然语言查询语句和对应的 SQL 查询语句。它有如下几个特点：

(1)数据集所覆盖的查询均为单表查询，表名在输入中给出，SELECT 子句中只有 1 个字段，且不包括 Order By，Group By 等复杂子句；

(2)该数据集假设 WHERE 子句中的属性值在自然查询语句中出现，这说明自然语言查询语句与数据表内容存在联系，获取并有效地利用数据表内容将能有效提升该数据集上的预测效果；

(3)将数据集划分为训练集、验证集、测试集，针对同一个数据表的查询只会在一个集合中出现，这说明该数据集要求模型具有迁移到新数据表的能力。

表 1 WikiSQL 数据集的规模

Table 1 Size of WikiSQL

	样本数量	数据表数量
训练集	56355	18585
验证集	8421	2716
测试集	15878	5230

5.2 评价指标

对于模型最终预测的 SQL 查询语句，我们采用以下两个指标进行评价。

(1)语句生成准确率(Acc_{qm})[3]。判断预测的 SQL 查询语句和正确的 SQL 查询语句是否具有相同的标准形式，Acc_{qm} 指满足这一条件的测试样本占测试集的比例。这里的标准形式是指一个大小为 3 的集合，元素分别是聚合查询类型、SELECT 子句中的字段、WHERE 子句集合。其中，WHERE 子句集合中的每个元素也是一个集合，代表一个 WHERE 子句，包含字段、运算符、属性值 3 个元素。这种方法在判断准确率时不考虑 WHERE 子句的顺序。

(2)查询准确率(Acc_{ex})[2]。判断预测的 SQL 查询语句和正确的 SQL 查询语句是否在数据库中具有相同的查询，Acc_{ex} 指满足这一条件的测试样本占测试集的比例。

除了对完整 SQL 查询语句的评价，我们还分别对标准形式中每个元素的预测准确率进行了评价。

5.3 实验设置

本文提出了结合数据表内容的字段嵌入方法以及数据增强方法，并将其应用在以 TypeSQL 为基础的模型结构上。为了评价这些工作，我们设置以下实验。

(1)不利用数据表内容的方法。SQLNet[3]是一种基于槽填充的方法，对 WHERE 子句采用了序列-集合的结构和基于字段的注意力机制；TypeSQL(w/o type awareness)是 TypeSQL[4]去掉类型信息之后的基本模型结构，与 SQLNET 相比，它将 SELECT 子句的字段、WHERE 子句的个数、WHERE 子句的字段放在同一个子模型中进行预测；TypeSQL 则在基本模型上结合了外部知识库的实体类型信息。

(2)利用数据表内容的方法。TypeSQL + TC[4]是在 TypeSQL(w/o type awareness)模型上根据数据表内容对自然语言问句进行类型编码。

(3)本文方法。Ours 是本文中提到的模型，与 TypeSQL 相比，Ours 在嵌入层根据数据表内容对自然语言问句和字段进行嵌入表示，并且使用数据增强后的训练集进行训练，扩充后的训练集样本数为 232258，是原有训练集的 4.12 倍；Ours(w/o data augmentation)是只使用原始训练集进行训练得到的模型；Ours(w/o value embedding)使用扩充后的训练集，但模型中只使用字段名的词向量对字段进行嵌入表示，不使用基于数据表内容的嵌入方法。

模型使用不同的参数在训练集上进行多次训练，保留在验证集上效果最好的模型，最后用保留的模型在测试集上进行一次测试。

5.4　实现细节

本文模型的实现建立在基于 PyToch[13] 的 TypeSQL 模型上，词向量则是对各 300 维的预训练的 Glove 向量[14] 和释义向量[15] 进行拼接。除此之外，隐藏层的维度为 120，dropout rate 设置为 0.5，使用 Adam 算法[16] 进行优化，每批样本个数为 64，损失函数则与 SQLNet 相同。

5.5　实验结果

5.5.1　总体结果

表 2 列出了不同模型在 WikiSQL 数据集上的语句生成准确率和查询准确率，其中 Dev 代表验证集，Test 代表测试集。

表 2　WikiSQL 数据集的总体结果统计

Table 2　Overall results on WikiSQL

（单位：%）

算法	Dev		Test	
	Acc_{qm}	Acc_{ex}	Acc_{qm}	Acc_{ex}
SQLNet	63.2	69.8	61.3	68.0
TypeSQL(w/o type awareness)	66.5	72.8	64.9	71.7
TypeSQL	68.0	74.5	66.7	73.5
TypeSQL+TC	79.2	85.5	75.4	82.6
Ours	**79.7**	**85.8**	75.6	**83.5**
Ours(w/o data augmentation)	**79.7**	85.5	75.5	**83.3**
Ours(w/o value embedding)	**79.5**	85.5	75.5	**83.2**

在模型结构方面，因为 TypeSQL(w/o type awareness)和 SQLNet 的输入是相同的，都只有自然语言问句和数据表结构，所以从结果可以看出，TypeSQL 的模型结构具有较好的效果，说明本文模型建立在 TypeSQL 的基础上是有意义的。

在输入方面，TypeSQL 增加了外部知识库作为额外知识来源，给问句和字段添加实体类型信息；TypeSQL+TC 和我们提出的 3 种模型都是以数据表内容为额外知识来源。结果显示，TypeSQL 的准确率落后其他模型约 10%，说明使用数据表内容作为额外输入比实体类型信息更加有效。

在是否充分利用了数据表内容方面，我们将文中提出的 3 种模型与 TypeSQL+TC 进行了对比。从结果上看，结合数据表内容对字段进行嵌入表示以及数据增强都能够微弱地提升模型的查询语句生成准确率和查询准确率；而将两种方案同时使用时，模型在测试集上的查询准确率提升了近 1%。结果说明：TypeSQL+TC 对数据表内容的使用不够充分，而我们提出的两种方法都能够更好地利用数据表内容来提升模型效果。

5.5.2　分类结果

表 3 列出了模型对不同子句元素预测的准确率，其中 Acc_{agg} 表示对聚合函数预测的准确率，Acc_{sel} 表示对 SELECT 子句中列名预测的准确率，Acc_{where} 表示对 WHERE 子句预测的准确率。从表 3 中可以看出，这种细粒度的结果统计在 WHERE 子句上体现的差异最明显，因为 WHERE 子句最为复杂，预测难度很高。而本文提出的 3 个模型对 WHERE 子句的预测准确率都明显高于 TypeSQL+TC，说明本文提出的两种方法对 WHERE 子句的预测效果具有重要的提升作用；特别是仅使用数据增强的模型，在对测试集 WHERE 子句的预测上达到了 88.8%的准确率，这说明通过结合数据表的内容扩充训练集，能够使模型学习到更多关于属性值和字段的知识，从而对新的数据表也能够有非常好的迁移效果。

表 3　WikiSQL 数据集的分类结果统计

Table 3　Breakdown results on WikiSQL

（单位：%）

算法	Dev			Test		
	Acc_{agg}	Acc_{sel}	Acc_{where}	Acc_{agg}	Acc_{sel}	Acc_{where}
SQLNet	90.1	91.5	74.1	90.3	90.9	71.9
TypeSQL	90.3	93.1	78.5	90.5	92.2	77.8
TypeSQL+TC	90.3	93.5	92.8	90.5	92.1	87.9
Ours	**90.8**	93.3	**93.2**	90.5	92.0	**88.3**
Ours(w/o data augmentation)	90.5	93.3	**93.1**	90.4	92.1	**88.3**
Ours(w/o value embedding)	90.5	93.7	92.9	90.3	92.1	**88.8**

此外，我们注意到，本文模型对验证集聚合函数的预测效果都超过了 TypeSQL+TC，特别是 Ours 模型，准确率提升了 0.5%。这说明结合数据表内容不仅对 WHERE 子句的预测有帮助，通过学习到每个字段存储的数据的语义，对于聚合函数类型的预测也有帮助。但是在测试集上，我们提出的模型的 Acc_{agg} 与 TypeSQL+TC 持平，这说明所提模型可能产生了轻微的过拟合，未来会考虑对于聚合函数类型的预测寻找更加有代表性的特征。

从表 3 中的 Acc_{sel} 指标可以看出，通过对字段进行基于数据表内容的嵌入表示，Acc_{sel} 没有明显提升，因为 SELECT 子句中的字段通常是直接出现在问句中，直接使用基于字段名的嵌入表示就足以达到较好的效果。

5.5.3　实验总结

从以上实验中可以看出，结合了外部知识的模型在准确率上有了明显提升，而结合数据表的内容比结合实体类型信息更加有效。另外，在 TypeSQL+TC 模型结构的基础上，本文提出的两种结合数据表内容的方法都在测试集的查询准确率上取得了一定的提升。通过分析细粒度的实验结果得出，这种提升主要归功于对 WHERE 子句预测准确率的提升。而两种方法的提升能力相当，其中结合数据表内容的数据增强对 WHERE 子句提升的效果更好。总体来说，充分利用数据表内容能够以很小的获取和存储代价提升数据库自然语言接口的效果。

此外，需要说明两点：1）WikiSQL 数据集的大多数自然语言问句显式地指定了字段名；2）WikiSQL 数据集中的属性值基本都原封不动地出现在了自然语言问句中。数据集的这两个特点使得 TypeSQL+TC 中的类型编码方法能达到很好的效果，因此本文方法的提升程度有限，而本文方法从原理上更加适用于自然语言问句复杂多变的情况。

结束语　本文针对在数据库的自然语言接口中如何结合数据表内容这一问题，提出了一种基于字段嵌入的数据库自然语言查询接口技术，还提出了一种基于数据表内容的数据

增强方法来扩充训练集。在 WikiSQL 数据集上进行的实验说明了这两种方法的有效性，也说明了数据表内容对于数据库的自然语言接口是很重要的信息。WikiSQL 数据集中的查询语句比较简单，之后会研究在查询语句更加复杂、自然语言查询更加模糊的情况下如何更好地结合数据表内容完成自然语言到 SQL 查询的转换。此外，使用神经网络的方法实现中文问句到 SQL 语句的转换也是未来研究的重点。

参 考 文 献

[1] DONG L, LAPATA M. Language to Logical Form with Neural Attention[C]// Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics. 2016: 33-43.

[2] ZHONG V, XIONG C, SOCHER R. Seq2sql: Generating structured queries from natural language using reinforcement learning[J]. arXiv: 1709.00103, 2017.

[3] XU X, LIU C, SONG D. Sqlnet: Generating structured queries from natural language without reinforcement learning[J]. arXiv: 1711.04436, 2017.

[4] YU T, LI Z, ZHANG Z, et al. TypeSQL: Knowledge-Based Type-Aware Neural Text-to-SQL Generation[C]// Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. 2018: 588-594.

[5] GUO J, ZHAN Z, GAO Y, et al. Towards Complex Text-to-SQL in Cross-Domain Database with Intermediate Representation [J]. arXiv: 1905.08205, 2019.

[6] HWANG W, YIM J, PARK S, et al. A Comprehensive Exploration on WikiSQL with Table-Aware Word Contextualization [J]. arXiv: 1902.01069, 2019.

[7] ANDROUTSOPOULOS I, RITCHIE G D, THANISCH P. Natural language interfaces to databases-an introduction[J]. Natural Language Engineering, 1995, 1(1): 29-81.

[8] DEVLIN J, CHANG M W, LEE K, et al. Bert: Pre-training of deep bidirectional transformers for language understanding[J]. arXiv: 1810.04805, 2018.

[9] PETROVSKI B, AGUADO I, HOSSMANN A, et al. Embedding Individual Table Columns for Resilient SQL Chatbots [J]. EMNLP 2018, 2018: 67.

[10] SUN Y, TANG D, DUAN N, et al. Semantic Parsing with Syntax-and Table-Aware SQL Generation[C]// Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics. 2018: 361-372.

[11] YAVUZ S, GUR I, SU Y, et al. What It Takes to Achieve 100% Condition Accuracy on WikiSQL[C]// Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing. 2018: 1702-1711.

[12] VINYALS O, FORTUNATO M, JAITLY N. Pointer networks [C]// Advances in Neural Information Processing Systems. 2015: 2692-2700.

[13] PASZKE A, GROSS S, MASSA F, et al. Pytorch: An imperative style, high-performance deep learning library[C]// Advances in Neural Information Processing Systems. 2019: 8026-8037.

[14] PENNINGTON J, SOCHER R, MANNING C. Glove: Global vectors for word representation[C]// Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). 2014: 1532-1543.

[15] WIETING J, GIMPEL K. Paranmt-50m: Pushing the limits of paraphrastic sentence embeddings with millions of machine translations[J]. arXiv: 1711.05732, 2017.

[16] KINGMA D P, BA J. Adam: A method for stochastic optimization[J]. arXiv: 1412.6980, 2014.

TIAN Ye, born in 1996, postgraduate. His main research interests include knowledge graph and natural language processing.

CHEN Ke, born in 1977, Ph.D, associate professor. Her main research interests include spatial temporal data management, Web data mining and data privacy protection, etc.