



(12) 发明专利

(10) 授权公告号 CN 110945495 B

(45) 授权公告日 2022. 04. 29

(21) 申请号 201880033017.3

(22) 申请日 2018.05.17

(65) 同一申请的已公布的文献号
申请公布号 CN 110945495 A

(43) 申请公布日 2020.03.31

(30) 优先权数据
62/508,367 2017.05.18 US
15/885,613 2018.01.31 US

(85) PCT国际申请进入国家阶段日
2019.11.18

(86) PCT国际申请的申请数据
PCT/US2018/033099 2018.05.17

(87) PCT国际申请的公布数据
W02018/213530 EN 2018.11.22

(73) 专利权人 易享信息技术有限公司
地址 美国加利福尼亚州

(72) 发明人 V·钟 熊蔡明 R·佐赫尔

(74) 专利代理机构 北京市磐华律师事务所
11336

代理人 高伟

(51) Int.Cl.
G06F 16/13 (2019.01)
G06F 16/2452 (2019.01)
G06F 16/2457 (2019.01)
G06N 3/04 (2006.01)
G06N 3/08 (2006.01)

(56) 对比文件
CN 102622342 A, 2012.08.01
CN 106598948 A, 2017.04.26
CN 106663092 A, 2017.05.10
CN 102693303 A, 2012.09.26
US 2017109355 A1, 2017.04.20
US 2016140123 A1, 2016.05.19
US 2016171050 A1, 2016.06.16
US 2017061330 A1, 2017.03.02
US 8380645 B2, 2013.02.19 (续)

审查员 刘莹莹

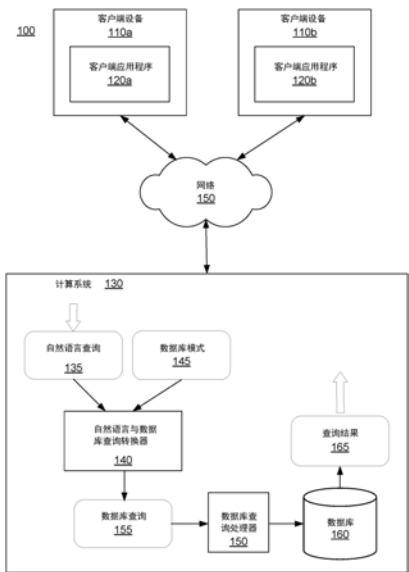
权利要求书4页 说明书27页 附图8页

(54) 发明名称

基于神经网络的自然语言查询到数据库查询的转换

(57) 摘要

计算系统使用神经网络将自然语言查询转换为数据库查询。该计算系统使用基于机器学习的多个模型,每个机器学习模型用于生成数据库查询的一部分。机器学习模型使用基于输入自然语言查询项、数据库模式的一组列和数据库查询语言(例如,结构化查询语言SQL)的词汇表生成的输入表示。基于机器学习的多个模型可包括:用于确定数据库查询的聚合算子的聚合分类器模型,用于确定数据库查询的结果列的结果列预测器模型,用于确定数据库查询的条件子句的条件子句预测器模型。该条件子句预测器是基于强化学习的。



[接上页]

(56) 对比文件

US 2007027905 A1, 2007.02.01

US 2011314010 A1, 2011.12.22

US 8156145 B2, 2012.04.10

US 2016342597 A1, 2016.11.24

1. 一种由包括一个或多个计算机的计算机系统执行的方法,其中所述计算机系统执行以下步骤:

基于使用数据库模式存储的数据,接收输入自然语言查询;

从多个项生成令牌序列,所述多个项包括:

所述输入自然语言查询的项,

所述数据库模式的一组列,以及

数据库查询语言的词汇表;

生成一个或多个输入表示,每个输入表示通过对所述令牌序列进行编码而获得;

访问多个基于机器学习的模型,每个基于机器学习的模型配置为预测与所述输入自然语言查询相对应的数据库查询的一部分,所述多个基于机器学习的模型包含:(1)用于预测所述数据库查询的选择子句的结果列预测器;(2)用于预测所述数据库查询的where子句的条件子句预测器;以及(3)用于预测所述数据库查询的聚合算子的聚合分类器,其中所述聚合分类器是使用交叉熵损失训练的神经网络,而所述条件子句预测器基于奖励度量使用强化学习;

对于所述多个基于机器学习的模型中的每一个,使用一输入表示执行所述基于机器学习的模型以生成所述数据库查询的一部分;

组合所生成的数据库查询的各部分,以获得所述数据库查询;以及

执行所述数据库查询,以获得结果集。

2. 根据权利要求1所述的方法,其中从客户端设备接收所述输入自然语言查询,并且所述方法还包括将所述结果集发送到所述客户端设备的步骤。

3. 根据权利要求1所述的方法,其中所述聚合分类器包括多层感知器。

4. 根据权利要求1所述的方法,其中所述结果列预测器包括多层感知器。

5. 根据权利要求1所述的方法,还包括:

基于地面实况数据库查询接收结果集;

基于从生成的查询获得的结果与从所述地面实况数据库查询获得的结果的比较来确定奖励值;以及

基于所述奖励值调整所述条件子句预测器模型的权重。

6. 根据权利要求1所述的方法,还包括:

确定与所述序列中的每个令牌相对应的列编码;

针对输入序列的每个令牌,确定包含标量注意力分数的向量;

使用softmax函数对所述向量进行归一化;以及

确定所述输入表示,作为由相应归一化的分数加权的列编码之和。

7. 根据权利要求1所述的方法,还包括:

使用梯度下降对所述多个基于机器学习的模型进行训练,以基于所述多个模型中每一个的结果最小化表示损失的目标函数。

8. 根据权利要求1所述的方法,其中生成所述一个或多个输入表示包括:通过以下步骤计算输入表示 $\mathbf{k}^{agg}$:

针对令牌序列中的每个第t个令牌,计算标量注意力分数 $\alpha_t^{inp} = W^{inp} h_t^{enc}$,其中 h_t^{enc} 是与输入序列中的第t个词相对应的编码器的状态,

归一化分数向量 $\alpha^{inp} = [\alpha_1^{inp}, \alpha_2^{inp}, \dots]$,以产生所述令牌序列中的令牌的分布,
获得所述输入表示 κ^{agg} 为

$$\kappa^{agg} = \sum_t \beta_t^{inp} h_t^{enc}$$

$$\alpha^{agg} = W^{agg} \tanh(V^{agg} \kappa^{agg} + b^{agg}) + c^{agg},$$

其中 W^{agg} 、 V^{agg} 、 b^{agg} 和 c^{agg} 表示应用于所述输入表示 κ^{agg} 的多层感知器的权重, α^{agg} 表示聚合运算的分数,所述聚合运算包括COUNT,MIN,MAX或无聚合运算NULL。

9. 根据权利要求1所述的方法,其中所述数据库查询包括SQL查询,所述数据库查询的一部分包括所述数据库查询的结果列,所述结果列包括所述SQL查询的选择子句中指定的列,所述方法还包括:

使用具有指针网络的长短期记忆网络LSTM,所述指针网络用一个或多个列名称来公式化所述SQL查询的选择子句,包括:

给出列表示和问题表示的列表,选择与问题最匹配的列,其中通过使用LSTM对每个列名称进行编码来获得列表示的列表,其中特定列j的表示 e_j^c 通过

$$h_{j,t}^c = \text{LSTM}(\text{emb}(x_{j,t}^c), h_{j,t-1}^c) \quad e_j^c = h_{j,T_j}^c \text{ 给出,}$$

其中 $h_{j,t}^c$ 表示第j列的第t个编码器状态。

10. 根据权利要求1所述的方法,还包括:

使用强化学习通过对数据库执行生成的SQL查询来公式化SQL查询的where条件,以获得奖励 $R(q(y), q_g)$, $R(q(y), q_g)$ 被定义为:

$$R(q(y), q_g) = \begin{cases} -2, & \text{如果 } q(y) \text{ 不是有效的 SQL 查询。} \\ -1, & \text{如果 } q(y) \text{ 是有效的 SQL 查询并执行得到不正确的结果} \\ +1, & \text{如果 } q(y) \text{ 是有效的 SQL 查询并执行得到正确的结果。} \end{cases}$$

其中 $q(y)$ 表示由所述基于机器学习的模型生成的查询, q_g 表示与所述输入自然语言查询相对应的地面实况查询。

11. 一种用于将自然语言查询转换到数据库查询的方法,其中所述数据库查询包括SQL查询,所述方法包括:

处理自然语言请求以生成所述SQL查询的部分,包括:

使用具有指针网络的长短期记忆网络LSTM,所述指针网络用一个或多个列名称来公式化所述SQL查询的选择子句;和

使用通过强化学习训练的增强指针解码器来公式化所述SQL查询的where条件。

12. 根据权利要求11所述的方法,还包括:

使用多层感知器确定适用于指定条件下选择的列的所述SQL查询的聚合运算。

13. 根据权利要求11所述的方法,其中使用具有指针网络的长短期记忆网络LSTM,所述指针网络用一个或多个列名称来公式化所述SQL查询的选择子句,包括:

给定列表示和问题表示的列表,选择与所述问题最匹配的列。

14. 根据权利要求13所述的方法,其中所述列表示的列表通过使用LSTM对每个列名称进行编码来获得,其中特定列j的表示 e_j^c 通过 $h_{j,t}^c = \text{LSTM}(\text{emb}(x_{j,t}^c), h_{j,t-1}^c) \quad e_j^c = h_{j,T_j}^c$

给出,

其中 $h_{j,t}^c$ 表示第j列的第t个编码器状态。

15. 根据权利要求13所述的方法, 还包括: 生成一个或更多个输入表示, 用于提供给所述LSTM以生成所述数据库查询的各个部分,

其中生成所述一个或更多个输入表示包括通过以下步骤计算输入表示 κ^{agg} :

针对令牌序列中的每个第t个令牌, 计算标量注意力分数 $\alpha_t^{\text{inp}} = W^{\text{inp}} h_t^{\text{enc}}$, 其中 h_t^{enc} 是与输入序列中的第t个词相对应的编码器的状态,

归一化分数向量 $\alpha^{\text{inp}} = [\alpha_1^{\text{inp}}, \alpha_2^{\text{inp}}, \dots]$ 以产生所述令牌序列中的令牌的分布,

获得所述输入表示 κ^{agg} 为

$$\kappa^{\text{agg}} = \sum_t \beta_t^{\text{inp}} h_t^{\text{enc}}$$

$$\alpha^{\text{agg}} = W^{\text{agg}} \tanh(V^{\text{agg}} \kappa^{\text{agg}} + b^{\text{agg}}) + c^{\text{agg}},$$

其中 W^{agg} 、 V^{agg} 、 b^{agg} 和 c^{agg} 表示应用于所述输入表示 κ^{agg} 的多层感知器的权重, α^{agg} 表示聚合运算的分数, 所述聚合运算包括COUNT, MIN, MAX或无聚合运算NULL。

16. 根据权利要求13所述的方法, 还包括:

使用强化学习通过对数据库执行生成的SQL查询来公式化SQL查询的where条件, 以获得奖励 $R(q(y), q_g)$, $R(q(y), q_g)$ 被定义为:

$$R(q(y), q_g) = \begin{cases} -2, & \text{如果 } q(y) \text{ 不是有效的 SQL 查询;} \\ -1, & \text{如果 } q(y) \text{ 是有效的 SQL 查询并执行得到不正确的结果;} \\ +1, & \text{如果 } q(y) \text{ 是有效的 SQL 查询并执行得到正确的结果。} \end{cases}$$

其中 $q(y)$ 表示由基于机器学习的模型生成的查询, q_g 表示与所述自然语言查询相对应的地面实况查询。

17. 一种非暂时性计算机可读存储介质, 包括计算机可执行代码, 所述计算机可执行代码在由一个或更多个计算机执行时, 使一个或更多个计算机执行步骤, 所述步骤包括:

基于使用数据库模式存储的数据, 接收输入自然语言查询;

从多个项生成令牌序列, 所述多个项包括:

所述输入自然语言查询的项,

所述数据库模式的一组列, 以及

数据库查询语言的词汇表;

生成一个或更多个输入表示, 每个输入表示通过对所述令牌序列进行编码而获得;

访问多个基于机器学习的模型, 每个基于机器学习的模型配置为预测与所述输入自然语言查询相对应的数据库查询的一部分, 所述多个基于机器学习的模型包含: (1) 用于预测所述数据库查询的选择子句的结果列预测器; (2) 用于预测所述数据库查询的where子句的条件子句预测器; 以及 (3) 用于预测所述数据库查询的聚合算子的聚合分类器, 其中所述聚合分类器是使用交叉熵损失训练的神经网络, 而所述条件子句预测器基于奖励度量使用强化学习;

对于所述多个基于机器学习的模型中的每一个, 使用一输入表示执行所述基于机器学习的模型以生成所述数据库查询的一部分;

组合所生成的数据库查询的各部分,以获得所述数据库查询;以及
执行所述数据库查询,以获得结果集。

18.一种用于生成循环神经网络RNN体系架构的计算机系统,所述计算机系统包括:
一个或多个计算机处理器;以及

非暂时性计算机可读存储介质,包括计算机可执行代码,所述计算机可执行代码在由
一个或多个计算机执行时,使一个或多个计算机执行步骤,所述步骤包括:

基于使用数据库模式存储的数据,接收输入自然语言查询;

从多个项生成令牌序列,所述多个项包括:

所述输入自然语言查询的项,

所述数据库模式的一组列,以及

数据库查询语言的词汇表;

生成一个或多个输入表示,每个输入表示通过对所述令牌序列进行编码而获得;

访问多个基于机器学习的模型,每个基于机器学习的模型配置为预测与所述输入自然
语言查询相对应的数据库查询的一部分;

对于所述多个基于机器学习的模型中的每一个,使用一输入表示执行所述基于机器学
习的模型以生成所述数据库查询的一部分,所述多个基于机器学习的模型包含:(1)用于预
测所述数据库查询的选择子句的结果列预测器;(2)用于预测所述数据库查询的where子句
的条件子句预测器;以及(3)用于预测所述数据库查询的聚合算子的聚合分类器,其中所述
聚合分类器是使用交叉熵损失训练的神经网络,而所述条件子句预测器基于奖励度量使用
强化学习;

组合所生成的数据库查询的各部分,以获得所述数据库查询;以及
执行所述数据库查询,以获得结果集。

基于神经网络的自然语言查询到数据库查询的转换

技术领域

[0001] 本公开总地涉及数据库查询的自动生成,并且更具体地,涉及用于将自然语言查询转换成数据库查询的基于神经网络的模型。

背景技术

[0002] 世界上大量可用的数据存储存储在关系数据库中,关系数据库为诸如医疗记录、金融市场、客户关系管理之类的应用程序提供了基础。但是,访问关系数据库中的信息需要了解数据库查询语言,例如结构化查询语言(SQL)。尽管诸如SQL之类的数据库查询语言在允许用户指定对来自关系数据库的数据的请求方面具有强大功能,但它们却很难学习。为了能够使用数据库查询语言有效地编写数据库查询,需要具备数据库方面的专业知识和强大的技术知识。

[0003] 一些系统支持自然语言来访问存储在系统中的数据。自然语言查询提供了易于表达,因为人们不需要培训如何使用自然语言。但是,这些系统不提供数据库查询语言(例如SQL)的表达能力。例如,自然语言查询可以以多种方式来解释,但相应地,执行自然语言查询来访问存储在关系数据库中的数据可能效率不高,并且可能无法检索所请求的确切信息。因此,使用自然语言查询或数据库查询来访问关系数据库中存储的数据的常规技术具有缺点,因为它们要么提供了易于表达,要么提供了表达的能力,但不能同时提供两者。

附图说明

[0004] 所公开的实施例具有其他优势和特征,这些优势和特征从具体实施方式、附加的权利要求以及所附图(附图)将更加显而易见。

[0005] 图1是示出了根据实施例的用于将自然语言查询转换为数据库查询的整个系统环境的高级框图。

[0006] 图2示出了根据实施例的用于将自然语言查询转换成数据库查询的计算系统的系统架构。

[0007] 图3示出了根据实施例的自然语言与数据库查询转换器执行的处理的细节。

[0008] 图4示出了根据实施例的用于将自然语言查询转换成数据库查询的总体过程。

[0009] 图5示出了根据实施例的聚合分类器基于自然语言查询来确定输出数据库查询的聚合算子的过程。

[0010] 图6示出了根据实施例的结果列预测器基于自然语言查询来确定输出数据库查询的SELECT子句的列的过程。

[0011] 图7示出了根据实施例的训练条件子句预测器以确定输出数据库查询的条件子句的过程。

[0012] 图8是说明用于实现图1的客户端设备和/或计算系统的示例计算机的高级框图。

[0013] 这些图(附图)和下面的描述仅通过举例来描述某些实施例。本领域技术人员将从以下描述中容易地认识到,在不脱离本文描述的原理的情况下,可以采用本文所示的结构

和方法的替代实施例。现在将详细参考几个实施例，其示例在附图中进行了说明。

具体实施方式

[0014] 计算系统使用深度神经网络将自然语言查询转换为相应的数据库查询，例如，使用结构化查询语言 (SQL) 指定的查询。实施例使用SQL查询的结构来大幅度减小所生成的查询的输出空间。该计算系统使用深度神经网络将自然语言查询转换 (translate) 为数据库查询。

[0015] 在实施例中，计算系统使用多个基于机器学习的模型，例如基于神经网络的模型，来生成输出数据库查询的不同部分。例如，计算系统可以使用聚合分类器模型来确定数据库查询中的聚合算子，使用结果列预测器模型来确定数据库查询的结果列，以及使用条件子句预测器模型来确定数据库查询的条件子句。在一个实施例中，聚合分类器模型和结果列预测器模型包括多层感知器。条件子句预测器模型使用基于策略的强化学习 (RL) 来生成数据库查询的条件子句。之所以如此，是因为条件子句本质上是无序的，并且条件子句的多个表示可以为数据库查询提供相同的输出结果。因此，条件子句不适合使用交叉熵损失进行优化。使用组合了交叉熵损失和RL奖励的混合目标来训练深度神经网络。

[0016] 例如，数据库可以存储带有列编号 (Pick_number)，CFL团队 (CFL_Team)，玩家 (Player)，位置 (Position) 和大学 (College) 的表CFLDraft。该表可以存储以下示例行：

编号	CFL 团队	玩家	位置	大学
27	HamiltonTiger-Cats	ConnorHealy	DB	Wilfrid Laurier(劳里埃)
28	CalgaryStampeders	AnthonyForgone	OL	York (约克)
29	OttawaRenegades	L.P.Ladouceur	DT	California (加州)
30	TorontoArgonauts	Frank Hoffman	DL	York (约克)
...	...	...	...	...

[0018] 系统接收自然语言查询，例如“`How many CFL teams are from York College?` (有多少个CFL团队来自约克大学?)”系统处理接收到的自然语言查询，该自然语言查询与包含表CFLDraft的数据库模式相关联，以使用SQL语言“`SELECT COUNT (CFL_Team) FROM CFLDraft WHERE College=“York”`”来生成数据库查询。系统使用数据库模式执行数据库查询。表CLFDraft的两行匹配数据库查询的WHERE子句，因为它们具有大学“约克 (York)”。

结果，系统返回结果2。

[0019] 整个系统环境

[0020] 图1是根据实施例的示出了用于将自然语言查询转换为数据库查询的整个系统环

境的高级框图。系统环境100包括通过网络150连接到计算系统130的一个或更多个客户端设备110。计算系统130可以是在线系统,但是也可以离线工作,例如通过执行将一组自然语言查询中的每一个转换成数据库查询的批处理。

[0021] 这里仅示出了两个客户端设备110a,110b,但是这些实体中的每个实体可以有多个实例。例如,可以存在多个计算系统130以及与每个计算系统130通信的数十个或数百个客户端设备110。附图使用相同的附图标记来标识相同的元件。诸如“110a”之类的附图标记后的字母表示该文本专门指代具有该特定附图标记的元素。文本中没有以下字母的参考数字,例如“110”,是指图中带有该参考数字的任何或所有元件。

[0022] 客户端设备110是诸如具有**ANDROID®**、**APPLE®IOS®**之类的操作系统的智能手机,平板计算机,膝上型计算机,台式计算机,汽车或其他车辆中的电子立体声设备,或任何其他类型的启用网络的设备(可以收听或以其他方式体验数字内容)。典型的客户端设备110包括连接到网络150所需的硬件和软件(例如,经由Wifi和/或4G或其他无线电信标准)。

[0023] 客户端设备110包括客户端应用程序120,客户端应用程序120允许客户端设备110的用户与计算系统130进行交互。例如,客户端应用程序120可以是允许用户输入发送到计算系统130的自然语言查询的用户界面。客户端应用程序120从计算系统130接收结果,并将结果经由用户界面呈现给用户。在一个实施例中,客户端应用程序120是允许客户端设备110的用户与在计算系统130上执行的网络服务器交互的浏览器。

[0024] 计算系统130包括用于执行一组协调功能或任务的软件。该软件可以允许计算系统130的用户执行某些任务或感兴趣的活动,或者可以包括向其他软件提供某些功能和服务的系统软件(例如,操作系统)。计算系统130从客户端设备110接收请求,并执行与所接收的请求相关联的计算机程序。作为示例,计算系统130可以执行响应于来自客户端设备110的请求的计算机程序,以将自然语言查询转换为数据库查询。在计算系统130上执行的软件可以包括以协作方式编写的计算机程序、库和相关数据的复杂集合,其中的多方或团队负责管理软件的不同组件。

[0025] 在一个实施例中,计算系统130从客户端设备110接收自然语言查询135。用户可以经由在计算系统130上执行的客户端应用程序120来提供自然语言查询135。计算系统130存储数据库模式145,该数据库模式145定义了存储在数据库中的数据结构。例如,数据库模式145可以标识存储在数据库中的各种表、每个表的列、表之间的关系(例如外键关系)、与表相关联的任何约束等等。

[0026] 自然语言与数据库查询转换器140接收自然语言查询135和数据库模式145作为输入,并生成与输入的自然语言查询135等效的数据库查询155。所生成的数据库查询155符合数据库模式145。所生成的数据库查询155由数据库查询处理器150接收,该数据库查询处理器150使用存储在数据库160中的数据来处理数据库查询155。数据库查询处理器150通过处理数据库查询155来生成查询结果165。计算系统130将所生成的查询结果165提供给在发送自然语言查询135的客户端设备110上运行的客户端应用程序120。

[0027] 在一个实施例中,自然语言与数据库查询转换器140执行序列到序列的转换。传统的基于神经网络的序列到序列转换器的搜索空间很大。相反,本实施例利用数据库查询语言中固有的结构来减少搜索空间。特别是,系统基于表模式、输入问题和SQL关键词的结合,

对生成的序列的输出空间进行了限制。在一个实施例中,自然语言与数据库查询转换器140使用深度神经网络,其中深度神经网络是具有增强的输入的指针网络。

[0028] 网络150在客户端设备110和记录管理系统130之间提供通信基础设施。网络150通常是互联网,但可以是任何网络,包括但不限于局域网(LAN)、城域网(MAN)、广域网(WAN)、移动有线或无线网络、专用网络或虚拟专用网络。网络150的部分可以使用通信技术通过链接提供,这些技术包括基于IEEE 802.11标准的WiFi、蓝牙短程标准和无线通用串行总线(USB)标准。

[0029] 系统架构

[0030] 图2示出了用于将自然语言查询转换成数据库查询的计算系统的系统架构。计算系统130包括输入编码模块210、训练模块240、自然语言与数据库查询转换器140、查询合成模块220,查询执行引擎230、训练数据存储215和数据库160。未示出诸如网络接口、安全功能、负载平衡器、故障转移服务器、管理和网络操作控制台等的常规组件,以免模糊系统架构的细节。

[0031] 输入预处理模块210预处理输入数据,以作为输入提供给自然语言与数据库查询转换器140。在一个实施例中,输入预处理模块210通过将来自数据库模式、输入的自然语言查询和数据库查询语言(例如,SQL)的词汇的列名称进行串接(concatenate)来生成令牌(token)序列(420)。输入预处理模块210生成一个或更多个输入表示,以提供给生成输出数据库查询的各个部分的各个模型。

[0032] 自然语言与数据库查询转换器140处理输入的自然语言查询,以生成与自然语言查询相对应的数据库查询。在一个实施例中,自然语言与数据库查询转换器140包括其他组件,例如,聚合分类器260、结果列预测器270和条件子句预测器280,结合图3进一步描述。

[0033] 自然语言与数据库查询转换器140使用不同的神经网络生成数据库查询的不同组件。在一个实施例中,自然语言与数据库查询转换器140使用不同的神经网络来生成数据库查询的组件,包括选择列、聚合算子和where子句。

[0034] 训练模块240使用存储在训练数据存储215中的历史数据来训练自然语言与数据库查询转换器140的神经网络。在一个实施例中,训练模块240使用交叉熵损失训练聚合分类器260和结果列预测器270,但是使用策略梯度强化学习训练条件子句预测器280,以解决查询条件的无序性质。使用SQL查询结构允许自然语言与数据库查询转换器140来减少数据库查询的输出空间。与不使用查询结构的其他技术相比,这将会导致显著更高的性能。

[0035] 查询合成模块220接收自然语言与数据库查询转换器140生成的数据库查询的各个组件,并将它们组合起来以获得数据库查询。查询执行引擎230使用存储在数据库160中的数据执行由查询合成模块220提供的数据库查询。计算系统130将查询的执行结果返回给结果的请求者,例如,在客户端设备110上执行的客户端应用程序120。

[0036] 图3示出了根据一实施例的自然语言与数据库查询转换器140执行的处理细节。如图3所示,自然语言与数据库查询转换器140的输入包括自然语言查询320和数据库模式320。在上面基于CFLDraft表说明的示例中,自然语言查询320为“How many CFL teams are from York College?(有多少个CFL团队来自约克大学?)”,数据库模式320包括各个列,其中各个列包括:编号(Pick_number),CFL团队(CFL_Team),玩家(Player),位置(Position)和大学(College)。示例输出数据库查询为:“SELECT COUNT(CFL_Team) FROM CFLDraft

WHERE College=“York””。

[0037] 输入预处理模块210生成一个或多个输入表示,并将输入表示提供给自然语言与数据库查询转换器140的每个组件,该自然语言与数据库查询转换器140包括聚合分类器260、结果列预测器270和条件子句预测器280。聚合分类器260、结果列预测器270和条件子句预测器280中的每一个都生成输出数据库查询的一部分。

[0038] 结果列预测器270生成结果列,例如,使用SQL表示的输出数据库查询的SELECT子句310中指定的列。结果列的示例是示例输出数据库查询中的列CFL团队。在一个实施例中,结果列预测器270是指针网络,其接收列序列的编码作为输入,并且指向对应于SELECT列的列序列中的列。

[0039] 条件子句预测器280生成输出数据库查询的WHERE子句320,该WHERE子句320指定用于过滤输出数据库查询的输出行的条件。在上面的示例中,WHERE子句“College=“York””是输出数据库查询中的条件子句。

[0040] 聚合分类器260在输出数据库查询中生成聚合算子330(如果有的话),例如在示例输出数据库查询中生成COUNT算子。聚合算子产生SQL选择的行的摘要。聚合分类器260可以生成的聚合算子的示例包括最大值(MAX)、最小值(MIN)、平均值(AVG)、和(SUM)等等。如果在输出查询中没有聚合算子,则聚合分类器260可以生成NULL聚合算子。

[0041] 提供包括SELECT子句310、WHERE子句320和聚合算子330在内的输出数据库查询的各个组件,以作为查询合成模块270的输入。查询合成模块270组合输出数据库查询的各个组件以生成完整的输出数据库查询340。

[0042] 整个过程

[0043] 图4-图7示出了用于将自然语言查询转换为数据库查询的各种过程。本领域技术人员将认识到,其他实施例可以以不同于流程图中所示的顺序执行图4-图7的步骤。而且,其他实施例可以包括与本文描述的步骤不同的步骤和/或另外的步骤。被指示为由某些模块执行的步骤可以由其他模块执行。

[0044] 图4示出了根据一实施例的用于将自然语言查询转换成数据库查询的总体过程。自然语言与数据库查询转换器140接收输入的自然语言查询(410)。输入预处理模块210通过串接来自数据库模式的列名、输入的自然语言查询和数据库查询语言的词汇表(例如SQL语言的各种关键字,例如SELECT, FROM, WHERE等等)来生成令牌序列(420)。例如,等式(1)示出了令牌序列包含列名称 x_i^c 、表示SQL词汇的项(term) x^s 、表示输入的自然语言查询的项 x^q :

$$[0045] \quad x = [< col >; x_1^c; x_2^c; \dots; x_N^c; < sql >; x^s; < question >; x^q] \quad (1)$$

[0046] 在等式(1)中,序列a与b的串接表示为[a;b]。此外,组合的序列x包括相邻序列之间的哨兵(sentinel)令牌以划分边界。例如,令牌<col>标识列名称,令牌<sql>标识表示SQL词汇表的项,令牌<question>标识输入的自然语言查询的项。

[0047] 输入预处理模块210生成令牌序列的输入表示(430)。在实施例中,输入预处理模块210生成多个输入表示,多个模型中的每个模块一个输入表示。

[0048] 自然语言与数据库查询转换器140访问多个神经机器学习模型,每个模型被配置为生成输出数据库查询的一部分。在一个实施例中,自然语言与数据库查询转换器140将多个基于经训练的神经网络的模型从存储设备加载到存储器。自然语言与数据库查询转换器140向多个基于机器学习的模型中的每一个提供输入表示(450)。多个基于机器学习的模型

中的每一个都生成数据库查询的一部分。

[0049] 在一些实施例中,输入预处理模块210生成多个输入表示,自然语言与数据库查询转换器140可以向每个基于机器学习的模型提供不同的输入表示。每个基于机器学习的模型都生成数据库查询的一部分,并将其提供给查询合成模块270。查询合成模块270组合数据库查询的多个部分以生成完整的数据库查询(460)。查询执行引擎230执行数据库查询以生成结果集(470)。

[0050] 聚合分类器

[0051] 图5示出了聚合分类器基于自然语言查询来确定输出数据库查询的聚合算子的过程。聚合分类器260基于在输入的自然语言查询中指定的问题的类型来确定输出数据库查询的聚合算子。例如,聚合分类器260可以将包括字符串“多少(how many)”的输入问题映射到聚合算子COUNT,聚合分类器260可以将包括“什么是最高的(what is the highest)”的输入问题映射到聚合算子最大值,聚合分类器260可以将包括“什么是最小的(what is the smallest)”的输入问题映射到聚合算子的最小值等等。

[0052] 聚合分类器260确定输入令牌序列的输入表示(510)。聚合分类器260计算输入序列中的每个第 t 个令牌的标量注意力分数 $\alpha_t^{\text{inp}} = w^{\text{inp}} * h_t^{\text{enc}}$ 。聚合分类器260生成分数的向量 $\alpha^{\text{inp}} = [\alpha_1^{\text{inp}}, \alpha_2^{\text{inp}}, \dots]$ 。聚合分类器260归一化分数向量 α^{inp} ,以通过对向量 α^{inp} 应用softmax函数确定 $\beta^{\text{inp}} = \text{softmax}(\alpha^{\text{inp}})$,来产生输入编码的分布。聚合分类器260产生输入编码的分布。聚合分类器260将输入表示 κ^{agg} 确定为由归一化的分数 β^{inp} 加权的输入编码 h^{enc} 的总和,如下等式所示。

$$[0053] \quad \kappa^{\text{agg}} = \sum_t \beta_t^{\text{inp}} h_t^{\text{enc}} \quad (2)$$

[0054] 聚合分类器260包括应用于所生成的输入表示 κ^{agg} 的多层感知器,以生成与各种聚合运算相对应的分数 α^{agg} ,聚合运算例如COUNT、MIN、MAX、指示没有聚合的NULL算子等。聚合分类器260基于所生成的分数来识别针对数据库查询的聚合运算(530)。

[0055] 在一个实施例中,聚合分类器260使用以下方程确定 α^{agg} 。

$$[0056] \quad \alpha^{\text{agg}} = w^{\text{agg}} \tanh(V^{\text{agg}} \kappa^{\text{agg}} + b^{\text{agg}}) + c^{\text{agg}} \quad (3)$$

[0057] 项 $w^{\text{agg}}, V^{\text{agg}}, b^{\text{agg}}$ 和 c^{agg} 表示对应于多层感知器的权重。聚合分类器260应用softmax函数以获得在可能的聚合运算 $\beta^{\text{agg}} = \text{softmax}(\alpha^{\text{agg}})$ 的集合上的分布。基于交叉熵损失 L^{agg} 训练聚合分类器。

[0058] 结果列预测器

[0059] SELECT子句也称为选择列或结果列。结果列预测器270基于数据库模式中的表列以及自然语言查询来确定选择列。例如,给定自然语言查询“有多少个CFL团队(How many CFL teams)...”,结果列预测器270确定选择列包括CFLDraft表中的CFL_Teams列。因此,结果列预测器270解决了作为匹配问题的SELECT列预测问题。在一个实施例中,结果列预测器270使用指针来标识SELECT列。给定列表示的列表和自然语言查询的表示,结果列预测器270选择与自然语言查询最匹配的列。

[0060] 图6示出了结果列预测器执行的用于基于自然语言查询来确定输出数据库查询的SELECT子句的列的过程。结果列预测器270通过使用LSTM(长短期记忆网络)对每个列名称进行编码(610)来使用列的输入表示。输入预处理模块210使用以下等式生成特定列 j 的输

入表示 e_c^j (620) :

$$[0061] \quad h_{j,t}^c = \text{LSTM}(\text{emb}(x_{j,t}^c), h_{j,t-1}^c) \quad e_j^c = h_{j,Tj}^c \quad (4)$$

[0062] 在该等式中, $h_{j,t}^c$ 表示第j列的第t个编码器状态, emb是返回嵌入的函数。输入预处理模块210使最后的编码器状态为 e_c^j , 列j的表示。

[0063] 输入预处理模块210使用类似于以上针对 κ^{agg} 所描述的架构来构建用于自然语言查询 κ^{sel} 的表示。结果列预测器270以输入表示为条件, 在列表示上应用多层感知器(630), 以使用以下等式计算每列j的分数:

$$[0064] \quad \alpha_j^{\text{sel}} = W^{\text{sel}} \tanh(V^{\text{sel}} \kappa^{\text{sel}} + V^c e_j^c) \quad (5)$$

[0065] 在等式中 W^{sel} , V^{sel} , 和 V^c 是多层感知器的权重。结果列预测器270使用softmax函数对分数进行归一化(640), 以产生可能的SELECT列 $\beta^{\text{sel}} = \text{softmax}(\alpha^{\text{sel}})$ 的分布。在CFLDraft表的上述示例中, 分布是在“编号(Pick_number)”、“CFL团队(CFL_Team)”、“玩家(Player)”、“位置(Position)”和“大学(College)”列上。结果列预测器270基于归一化的分数来选择输出数据库查询的结果列(650)。基于交叉熵损失 L^{sel} 来训练聚合分类器。

[0066] 条件子句预测器

[0067] 在一个实施例中, 条件子句预测器使用指针解码器生成WHERE子句。但是, 查询的WHERE条件可以互换, 查询将产生相同的结果。例如, 给定自然语言查询“which males are older than 18(年龄大于18岁的男性)”, 输出数据库查询可以是“SELECT name FROM insurance WHERE age>18AND gender=“male””或者“SELECT name FROM insurance WHERE gender=“male”and age>18”。即使这两个数据库查询并不基于两个查询字符串之间的字符串匹配而匹配, 但这两次数据库查询都获得正确的执行结果。如果在训练神经网络时将第一次数据库查询作为地面实况(ground truth)提供, 并且使用交叉熵损失来监督训练, 则第二次数据库查询将被错误地处罚, 因为它与基于字符串匹配的第一次数据库查询不匹配。因此, 实施例应用强化学习来学习策略, 以直接优化数据库查询的执行结果的预期正确性。

[0068] 图7示出了根据一个实施例的训练条件子句预测器以确定输出数据库查询的条件子句的过程。条件子句预测器280接收自然语言查询710和数据库模式720作为输入, 以生成数据库查询730。条件子句预测器280使用数据库160发送数据库查询以执行, 以获得奖励度量。查询执行引擎230执行所生成的数据库查询730, 以获得预测的查询结果750。计算系统130将地面实况的查询结果750存储在训练数据存储215中。条件子句预测器280将预测的查询结果750与地面实况的查询结果750进行比较以确定奖励750。提供奖励作为条件子句预测器280的输入, 同时作为训练条件子句预测器280的反馈。

[0069] 由WHERE子句中的条件子句预测器280生成的令牌序列由 $y = [y^1, y^2, \dots, y^T]$ 表示。令 $q(y)$ 表示由模型生成的查询, 而 q_g 表示与自然语言查询相对应的地面实况的数据库查询。条件子句预测器280使用以下等式作为奖励度量 $R(q(y), q_g)$:

$$[0070] \quad R(q(y), q_g) = \begin{cases} \dots 2, & \text{如果 } q(y) \text{ 不是有效的 SQL 查询。} \\ -1, & \text{如果 } q(y) \text{ 是有效的 SQL 查询并执行得到不正确的结果} \\ +1, & \text{如果 } q(y) \text{ 是有效的 SQL 查询并执行得到正确的结果。} \end{cases}$$

[0071] 因此,如果所生成的数据库查询的执行结果与作为地面实况提供的预期结果相匹配,则条件子句预测器280分配正奖励。如果所生成的数据库查询的执行结果未能与作为地面实况提供的预期结果相匹配,或者如果所生成的数据库查询不是有效的数据库查询,则条件子句预测器280分配负奖励。

[0072] 条件子句预测器280将损失 L^{whe} 确定为可能的WHERE子句的负期望奖励。训练模块使用梯度下降来训练条件子句预测器280,以最小化目标函数 $L=L^{agg}+L^{sel}+L^{whe}$ 。因此,条件子句预测器280将总梯度确定为从预测SELECT列的交叉熵损失、预测聚合运算的交叉熵损失和条件子句的策略学习得到的梯度的加权和。

[0073] 自然语言与数据库查询转换器140的结构合并可减少可生成的无效的数据库查询。列名称导致的大量无效查询结果-生成的查询引用了表中不存在的选择列。当列名称包含许多令牌(例如“Miles

[0074] (km)”,其有4个令牌)时,这特别有用。为聚合引入分类器还可以降低错误率。聚合分类器的使用提高了预测COUNT算子的准确性和查全率(recall)。使用表示学习来生成条件子句会导致生成更高质量的WHERE子句,该子句的排序可能不同于地面实况。即使条件的顺序与地面实况查询不同,使用基于策略的表示学习来进行训练也会得到正确的结果。

[0075] 计算机架构

[0076] 图8是说明用于实现图1的客户端设备和/或计算系统的示例计算机的高级框图。计算机800包括耦合到芯片组804的至少一个处理器802。芯片组804包括存储器控制器集线器820和输入/输出(I/O)控制器集线器822。存储器806和图形适配器812耦合到存储器控制器集线器820,并且显示器818耦合到图形适配器812。存储设备808、输入设备814和网络适配器816耦合到I/O控制器集线器822。计算机800的其他实施例具有不同的架构。

[0077] 存储设备808是非暂时性计算机可读存储介质,例如硬盘驱动器、光盘只读存储器(CD-ROM)、DVD或固态存储器设备。存储器806保存处理器802使用的指令和数据。输入接口814可以是触摸屏接口、鼠标、跟踪球或其他类型的指示设备、键盘或其某种组合,并且用于将数据输入到计算机800中。在一些实施例中,计算机800可以被配置为经由来自用户的手势从输入接口814接收输入(例如,命令)。图形适配器812在显示器818上显示图像和其他信息。网络适配器816将计算机800耦合到一个或多个计算机网络。

[0078] 计算机800适于执行计算机程序模块以提供本文描述的功能。如本文所使用的,术语“模块”是指用于提供指定功能的计算机程序逻辑。因此,可以用硬件、固件和/或软件来实现模块。在一个实施例中,程序模块被存储在存储设备808上,被加载到存储器806中,并由处理器802执行。

[0079] 图1中实体使用的计算机800的类型可根据实施例和实体所需的处理能力而有所不同。计算机800可能缺少上面描述的一些组件,例如图形适配器812和显示器818。例如,计算系统130可以由多个刀片服务器组成,这些服务器通过网络(如服务器农场)进行通信。

[0080] 本申请的主题的实现尤其可以是以下示例1至21:

[0081] 1.一种方法,包括:

[0082] 基于使用数据库模式存储的数据,接收输入的自然语言查询;

[0083] 从多个项生成令牌序列,所述多个项包括:

[0084] 所述输入的自然语言查询的项,

- [0085] 所述数据库模式的一组列,以及
- [0086] 数据库查询语言的词汇表;
- [0087] 生成一个或更多个输入表示,每个输入表示通过对所述令牌序列进行编码而获得;
- [0088] 访问基于机器学习的多个模型,每个模型配置为预测与所述输入自然语言查询相对应的数据库查询的部分;
- [0089] 对于所述多个模型中的每一个,基于所述输入表示执行模型以生成所述数据库查询的部分;
- [0090] 组合所生成的数据库查询的部分,以获得所述数据库查询;
- [0091] 执行所述数据库查询,以获得结果集。
- [0092] 2. 根据示例1所述的方法,其中从客户端设备接收所述输入自然语言查询,并且所述方法还包括将所述结果集发送到所述客户端设备的步骤。
- [0093] 3. 根据示例1所述的方法,其中所述多个模型包括用于确定所述数据库查询中的聚合算子的聚合分类器模型,其中所述聚合分类器模型包括多层感知器。
- [0094] 4. 根据示例1所述的方法,其中所述多个模型包括用于确定所述数据库查询的结果列的结果列预测器模型,其中所述结果列预测器模型包括多层感知器。
- [0095] 5. 根据示例1所述的方法,其中所述多个模型包括用于确定所述数据库查询的条件子句的条件子句预测器模型,其中所述条件子句预测器模型是基于强化学习的。
- [0096] 6. 根据示例5所述的方法,还包括:
- [0097] 基于地面实况数据库查询接收结果集;
- [0098] 基于从生成的查询获得的结果与从地面实况查询获得的结果的比较来确定奖励值;以及
- [0099] 基于所述奖励值调整所述条件子句预测器模型的权重。
- [0100] 7. 根据示例5所述的方法,还包括:
- [0101] 确定与所述序列中的每个令牌相对应的列编码;
- [0102] 针对输入序列的每个令牌,确定包含标量注意力分数的向量;
- [0103] 使用softmax函数对所述向量进行归一化;以及
- [0104] 确定所述输入表示,作为由相应归一化的分数加权的列编码之和。
- [0105] 8. 根据示例1所述的方法,还包括:
- [0106] 使用梯度下降对所述多个模型进行训练,以基于所述多个模型中每一个的结果最小化表示损失的目标函数。
- [0107] 9. 根据前述示例之一所述的方法,其中生成所述一个或更多个输入表示包括:通过以下步骤计算输入表示 κ^{agg} :
- [0108] 针对令牌序列中的每个第 t 个令牌,计算标量注意力分数 $\alpha_t^{\text{in}} = W^{\text{inp}} h_t^{\text{enc}}$,其中 h_t^{enc} 是与输入序列中的第 t 个词相对应的编码器的状态,
- [0109] 归一化分数向量 $\alpha^{\text{in}} = [\alpha_1^{\text{inp}}, \alpha_2^{\text{inp}}, \dots]$,以产生所述令牌序列中的令牌的分布,
- [0110] 获得所述输入表示 κ^{agg} 为

[0111]
$$\kappa^{\text{agg}} = \sum_t \beta_t^{\text{inp}} h_t^{\text{enc}}$$

[0112] 其中 $\beta^{\text{agg}} = \text{softmax}(\alpha^{\text{agg}})$, 且 $\alpha^{\text{agg}} = W^{\text{agg}} \tanh(V^{\text{agg}} \kappa^{\text{agg}} + b^{\text{agg}}) + c^{\text{agg}}$ 。

[0113] 10. 根据前述示例之一所述的方法, 还包括:

[0114] 使用具有指针网络的长短期记忆 (LSTM), 所述长短期记忆 (LSTM) 用一个或多个列名称来公式化所述SQL查询的选择子句, 包括:

[0115] 给出列表示和问题表示的列表, 选择与问题最匹配的列, 其中通过使用LSTM对每个列名称进行编码来获得列表示的列表, 其中特定列j的表示 e_j^c 通过 $h_{j,t}^c = \text{LSTM}(\text{emb}(x_{j,t}^c), h_{j,t-1}^c)$ $e_j^c = h_{j,T_j}^c$ 给出。

[0116] 11. 根据前述示例之一所述的方法, 还包括:

[0117] 使用强化学习通过对数据库执行生成的SQL查询来公式化SQL查询的where条件, 以获得奖励 $R(q(y), q_y)$, $R(q(y), q_y)$ 被定义为:

[0118]
$$R(q(y), q_y) = \begin{cases} \dots 2, & \text{如果 } q(y) \text{ 不是有效的 SQL 查询。} \\ -1, & \text{如果 } q(y) \text{ 是有效的 SQL 查询并执行得到不正确的结果} \\ +1, & \text{如果 } q(y) \text{ 是有效的 SQL 查询并执行得到正确的结果。} \end{cases}$$

[0119] 其中 $q(y)$ 表示模型生成的查询, q_y 表示与所述输入自然语言查询相对应的地面实况查询。

[0120] 12. 一种方法, 包括:

[0121] 处理自然语言请求以生成SQL查询的部分, 包括:

[0122] 使用具有指针网络的长短期记忆 (LSTM), 所述长短期记忆 (LSTM) 用一个或多个列名称来公式化SQL查询的选择子句; 和

[0123] 使用通过强化学习训练的增强指针解码器来公式化SQL查询的where条件。

[0124] 13. 根据示例12所述的方法, 还包括:

[0125] 使用多层感知器确定适用于指定条件下选择的列的SQL查询的聚合运算。

[0126] 14. 根据示例12或13所述的方法, 其中使用具有指针网络的长短期记忆 (LSTM), 所述长短期记忆 (LSTM) 用一个或多个列名称来公式化所述SQL查询的选择子句, 包括:

[0127] 给定列表示和问题表示的列表, 选择与问题最匹配的列。

[0128] 15. 根据示例14所述的方法, 其中所述列表示的列表通过使用LSTM对每个列名称进行编码来获得, 其中特定列j的表示 e_j^c 通过 $h_{j,t}^c = \text{LSTM}(\text{emb}(x_{j,t}^c), h_{j,t-1}^c)$ $e_j^c = h_{j,T_j}^c$ 给出。

[0129] 16. 根据示例14-15之一所述的方法, 其中生成所述一个或多个输入表示包括通过以下步骤计算输入表示 κ^{agg} :

[0130] 针对令牌序列中的每个第t个令牌, 计算标量注意力分数 $\alpha_t^{\text{inp}} = W^{\text{inp}} h_t^{\text{enc}}$, 其中 h_t^{enc} 是与输入序列中的第t个词相对应的编码器的状态,

[0131] 归一化分数向量 $\alpha^{\text{inp}} = [\alpha_1^{\text{inp}}, \alpha_2^{\text{inp}}, \dots]$ 以产生所述令牌序列中的令牌的分布,

[0132] 获得所述输入表示 κ^{agg} 为

$$[0133] \quad \kappa^{\text{agg}} = \sum_t \beta_t^{\text{inp}} h_t^{\text{enc}}$$

[0134] 其中 $\beta^{\text{agg}} = \text{softmax}(\alpha^{\text{agg}})$ 且 $\alpha^{\text{agg}} = W^{\text{agg}} \tanh(V^{\text{agg}} \kappa^{\text{agg}} + b^{\text{agg}}) + c^{\text{agg}}$ 。

[0135] 17. 根据示例14-16之一所述的方法, 还包括:

[0136] 使用强化学习通过对数据库执行生成的SQL查询来公式化SQL查询的where条件, 以获得奖励 $R(q(y), q_g)$, $R(q(y), q_g)$ 被定义为:

$$[0137] \quad R(q(y), q_g) = \begin{cases} -2, & \text{如果 } q(y) \text{ 不是有效的 SQL 查询。} \\ -1, & \text{如果 } q(y) \text{ 是有效的 SQL 查询并执行得到不正确的结果} \\ +1, & \text{如果 } q(y) \text{ 是有效的 SQL 查询并执行得到正确的结果。} \end{cases}$$

[0138] 其中 $q(y)$ 表示模型生成的查询, q_g 表示与输入自然语言查询相对应的地面实况查询。

[0139] 18. 一种非暂时性计算机可读存储介质, 包括计算机可执行代码, 所述计算机可执行代码在由一个或多个计算机执行时, 使一个或多个计算机执行根据示例1-17中任一项的步骤。

[0140] 19. 一种用于生成循环神经网络 (RNN) 体系架构的计算机系统, 所述计算机系统包括:

[0141] 一个或多个计算机处理器; 以及

[0142] 非暂时性计算机可读存储介质, 包括计算机可执行代码, 该计算机可执行代码在由一个或多个计算机执行时, 使得该一个或多个计算机执行根据权利要求18所述的方法的步骤。

[0143] 20. 一种计算机系统, 包括一个或多个计算机处理器和至少一个非暂时性存储介质, 并且还包括:

[0144] 用于基于使用数据库模式存储的数据接收输入自然语言查询的装置;

[0145] 用于从多个项生成令牌序列的装置, 所述多个项包括:

[0146] 所述输入自然语言查询的项,

[0147] 所述数据库模式的一组列, 以及

[0148] 数据库查询语言的词汇表;

[0149] 用于生成一个或多个输入表示的装置, 每个输入表示通过对所述令牌序列进行编码而获得;

[0150] 用于访问基于机器学习的多个模型的装置, 每个模型配置为预测与所述输入自然语言查询相对应的数据库查询的部分;

[0151] 用于针对所述多个模型中的每一个, 基于输入表示执行模型以生成所述数据库查询的部分的装置;

[0152] 用于组合所生成的所述数据库查询的部分以获得所述数据库查询的装置; 以及

[0153] 用于执行所述数据库查询以获得结果集的装置。

[0154] 21. 一种计算机系统, 包括一个或多个计算机处理器和至少一个非暂时性存储介质, 并且还包括:

[0155] 处理自然语言请求以生成SQL查询的部分的装置, 包括:

[0156] 用于使用具有指针网络的长短期记忆 (LSTM) 的装置, 所述长短期记忆 (LSTM) 用一

个或更多个列名称来公式化SQL查询的选择子句;以及

[0157] 用于使用通过强化学习训练的增强指针解码器以公式化SQL查询的where条件的装置。

[0158] 替代实施例

[0159] 尽管所公开的实施例基于关系数据库并且使用SQL进行了说明,但是所公开的技术可应用于其他类型的数据库,例如,基于对象的数据库、对象关系数据库等。如果用于特定类型数据库的数据库查询语言支持等效于结果列、聚合子句或条件子句的特征,则所公开的技术适用。例如,如果数据库查询语言支持条件子句,则条件子句预测器可用于基于输入自然语言查询来预测输出数据库查询的条件子句。

[0160] 应当理解,本发明已经简化了附图和描述,以示出对于清楚地理解本发明相关的元件,同时为了清楚起见,省去了在典型的分布式系统中发现的许多其他元件。本领域技术人员可以认识到在实现实施例中其他元件和/或步骤是期望的和/或需要的。然而,由于这样的元件和步骤在本领域中是公知的,因为它们并不会促进对实施例的更好的理解,因此这里不再对这样的元件和步骤进行讨论。本文的公开内容针对本领域技术人员已知的对这样的元件和方法的所有这样的变型和修改。

[0161] 上面的描述的某些部分在算法和信息操作的符号表示方面来描述实施例。这些算法的描述和表示通常由数据处理领域的技术人员用来将其工作的实质有效地传达给本领域的其他技术人员。这些操作虽然在功能上、计算上或逻辑上进行了描述,但应理解为由计算机程序或等效电路、微代码等实现。此外,在不失一般性的情况下,有时将这些操作布置称为模块也是方便的。所描述的操作及其相关的模块可以用软件、固件、硬件或其任何组合来体现。

[0162] 如本文所使用的,对“一个实施例”或“一实施例”的任何引用均表示结合该实施例描述的特定元件、特征、结构或特性包括在至少一个实施例中。说明书中各个地方出现的短语“在一个实施例中”不一定都指的是同一实施例。

[0163] 可以使用表达“耦合”和“连接”及其派生词来描述一些实施例。应该理解的是,这些术语并不旨在彼此等同。例如,可以使用术语“连接”来描述一些实施例,以表示两个或更多个元件彼此直接物理或电接触。在另一个示例中,可以使用术语“耦合”来描述一些实施例,以指示两个或更多个元件是直接物理或电接触。然而,术语“耦合”还可以表示两个或更多元件彼此不直接接触,但是仍然彼此协作或相互作用。实施例不限于此上下文。

[0164] 如本文所使用的,术语“包括”、“包含”、“含有”、“具有”或其任何其他变型旨在覆盖非排他性包含。例如,包括一系列元素的过程、方法、物品或设备不必仅限于那些元素,而是可以包括未明确列出或此类过程、方法、物品或设备固有的其他元素。此外,除非明确指出相反的意思,否则“或”是指包含性的或而不是排他性的或。例如,条件A或B由以下任一条件满足:A为真(或存在)且B为假(或不存在),A为假(或不存在)且B为真(或存在),并且A和B均为真(或存在)。

[0165] 另外,使用“一个”或“一种”来描述本文的实施例的要素和组分。这样做仅仅是为了方便并给出本发明的一般意义。该描述应被理解为包括一个或至少一个,并且单数也包括复数,除非显而易见的是另有说明。

[0166] 在阅读了本公开之后,本领域技术人员将理解通过本文公开的原理,使用失真区

域显示图表的系统和过程的另外的替代结构和功能设计。因此,尽管已经示出和描述了特定的实施例和应用,但是应当理解,所公开的实施例不限于本文所公开的精确构造和组件。对本领域技术人员而言显而易见的是,在不脱离所附权利要求书所限定的精神和范围的情况下,可以对本文公开的方法和装置的布置、操作和细节进行各种修改、改变和变化。

[0167] 附录

[0168] 以下标题为“Seq2SQL:使用强化学习从自然语言生成结构化查询”的文档公开了“Seq2SQL”,它是此附录的一部分,因此也是本申请公开内容的一部分,其中公开了“Seq2SQL”-用于将自然语言问题转换成相应SQL查询的深度神经网络。Seq2SQL利用SQL查询的结构来显著减少所生成查询的输出空间。Seq2SQL包含三个组件,这些组件利用SQL查询的结构来大大减少所生成查询的输出空间。特别是,它使用指针解码器和策略梯度来生成查询的条件,由于它们的无序性质,我们证明该条件不适合使用交叉熵损失进行优化。Seq2SQL使用混合目标进行训练,该目标结合了交叉熵损失和数据库实时查询执行中的强化学习奖励。这些特征允许该模型在查询生成上可以实现大大改善的结果。

[0169] Seq2SQL:使用强化学习从自然语言生成结构化查询

[0170] 匿名作者

[0171] 附属

[0172] 地址

[0173] 埃内尔

[0174] 摘要

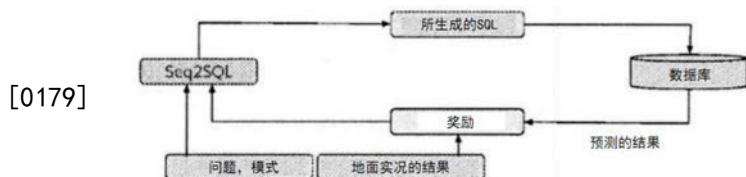
[0175] 关系数据库存储了大量的世界知识。然而,由于缺少对查询语言(如SQL)的理解,所以限制了用户从数据库中检索事实的能力。我们提出了Seq2SQL,这是一种将自然语言问题转换为相应的SQL查询的深度神经网络。我们的模型利用SQL查询的结构来显著减少生成的查询的输出空间。而且,我们使用循环查询执行将策略梯度强化学习来学习生成SQL查询条件的策略。查询的因素是无序的而且不太适合通过交叉熵损失进行优化。此外,我们将发布WikiSQL,这是由37000个手工注释的问题示例和分布在来自wikipedia的9500个表中SQL查询组成的数据集。这个数据集用来训练我们的模型并且比可比数据集大一个数量级。通过应用使用来自对WikiSQL执行的循环查询的奖励的策略梯度方法,我们发现Seq2SQL优于注意力序列到序列模型,执行精度从21.8%提高到52.2%,逻辑形式精度从20.4%提高到46.6%。

[0176] 1. 介绍

[0177] 关系数据库存储了当下大量的信息,其为医疗记录[Hillestad等人,2005]、金融市场[Beck等人,2000]和客户关系管理[Ngai等人,2009]等关键应用程序提供了基础。然而,访问关系数据库需要理解SQL等查询语言(诸如SQL),尽管这些语言功能强大,但是很难掌握。自然语言接口(NLI)[Androutsopoulos等人,1995]是自然语言处理和人机交互的交叉研究领域,旨在为人类提供通过使用自然语言与计算机交互的手段。我们研究了自然语言接口应用于关系数据库的一个特定方面:将自然语言问题转换为SQL查询。

[0178] 我们在这项工作中的主要贡献有两方面。首先,我们引入了Seq2SQL,这是用于将自然语言问题转换成相应SQL查询的深度神经网络。如图1所示,Seq2SQL由三个组件组成,它们利用SQL查询的结构来显著减少所生成的查询的输出空间。尤其是,它使用指针解码器

和策略梯度来生成查询条件,由于其无序性,该查询条件不适合使用交叉熵损失进行优化。Seq2SQL使用混合目标对Seq2SQL进行训练,该混合目标组合交叉熵损失和由对数据库执行的实时查询而得到的强化学习奖励。这些特性允许该模型实现对查询的改进的结果生成。



[0180] 图1:Seq2SQL将问题和表格的列表作为输入。它生成相应的SQL查询,在训练期间对数据库执行该查询。以执行结果作为奖励,训练强化学习算法。

[0181] 第二,我们发布了WikiSQL,这是一个由37000个手工注释的自然语言问题实例、SQL查询和从来自Wikipedia的9500个HTML表中提取的SQL表组成的语料库。WikiSQL还提供逻辑形式和自然语言表达的可比数据集要大一个数量级。我们以原始JSON格式和SQL数据库的形式发布了WikiSQL中使用的表。与WikiSQL一起,我们还为数据库发布了查询执行引擎,在策略学习期间我们使用它在循环中执行查询。在WikiSQL上,我们表明Seq2SQL的性能显著优于注意力序列到序列基线,其获得了20.3%的执行精度,以及指针网络基线,其获得了43.2%的执行精度。通过利用SQL查询的固有结构,并在训练期间使用来自实时查询执行的奖励信号应用策略梯度方法,Seq2SQL实现了对WikiSQL的显著改进,获得了52.2%的执行精度。

[0182] 2. 相关工作

[0183] 从自然语言表达学习逻辑形式。在问答(QA)的语义分析中,自然语言问题被解析为逻辑形式,然后在知识图谱[Zelle&Mooney,1996,Wong&Mooney,2007,Zettlemoyer&Collins,2005,2007.]上执行。语义分析领域的其他工作关注的是学习解析器,而不依靠注释的逻辑形式,而是利用会话日志[Artzi&Zettlemoyer,2011]、演示[Artzi&Zettlemoyer,2013]、远程监督[Cai&Yates,2013,Reddy等人,2014]和问答对[Liang等人,2011]。最近,Pasupat&Liang[2015]引入了浮动解析器来处理大量的表达以及在web表上语义解析的表模式的变化。我们的方法不同于这些语义分析技术,因为它不依赖于高质量的语法和词汇,而是依赖于表示学习。

[0184] 语义解析数据集。以前的语义解析系统通常设计成在封闭域、固定模式数据集(如GeoQuery[Tang&Mooney,2001]和ATIS[Price,1990])上以回答复杂的组合问题。研究人员还研究了大型知识图谱子集(如DBPedia[Starc&Mladenec,2017]和Freebase[Cai&Yates,2013;Berant等人,2013])的问答。与这些数据集相比,WikiSQL是最大的,并且还有很大的差距——它在示例的数量和域/表模式的数量上要大一个数量级。“overnight”数据集,Wang等人[2015]使用类似的众包(crowd-sourcing)过程来构建一个(自然语言问题、逻辑形式)成对的数据集,但只有8个域。WikiTableQuestions[Pasupat&Liang,2015](其和WikiSQL一样)是一个问题和答案的集合,也是从维基百科中提取的大量表格,在问题域的广度和问题复杂性的深度之间取得平衡。但是,WikiTableQuestions不提供逻辑形式,而WikiSQL提供。此外,WikiSQL侧重于为关系数据库表上的问题生成SQL查询,意图为真实世界构建自然语言接口,产生数据集。

[0185] 序列生成的表示学习。我们的基线模型是基于Bahdanau等人[2015]提出的注意力

序列到序列模型(Seq2Seq)。Seq2Seq在各种自然语言处理任务(例如:机器翻译[Sutskever等人,2014,Bahdaoui等人,2015]、总结[Nallapati等人,2016,Paulus等人,2017]、语义分析[Dong&Lapata,2016])上具有非常大的进步。Seq2Seq的灵感来自指针网络[Vinyals等人,2015],它不是从固定的词汇表(如序列到序列模型)生成词汇,而是从输入序列中选择词汇而生成。这类模型已成功地应用于诸如语言建模[Merity等人,2017]、总结[Gu等人,2016]、组合优化[Bello等人,2017]和问题回答[Seo等人,2017,Xiong等人,2017]之类的任务。神经方法也被应用于语义解析[Dong&Lapata,2016,Neelakantan等人,2017]。但是,与[Dong&Lapata,2016]不同的是,我们使用基于指针的生成方法而不是Seq2Seq生成方法,并证明了前一种方法工作得更好,特别是对于不可见的词汇和列名称。与[Neelakantan等人,2017]]不同的是,我们的模型在推理期间不需要访问表内容。与这两种方法相反,我们使用策略梯度来训练我们的模型,这再次显著地提高了性能。

[0186]

编号	CFL团队	玩家	位置	大学
27	Hamilton Tiger-Cats	Connor Healy	DB	Wilfrid Laurier
28	Calgary Stampeders	Anthony Forgone	OL	York
29	Ottawa Renegades	L.P. Ladouceur	DT	California
30	Toronto Argonauts	Frank Hoffman	DL	York

问题:
有多少个CFL团队来自约克的大学?

SQL:
"SELECT COUNT (CFL_Team) FROM CFLDraft WHERE College = "York" ;"

结果:
2

[0187] 图2:WikiSQL中的示例。输入包括表格和问题。输出包括地面实况SQL查询和相应的执行结果。

[0188] 数据库的自然语言接口。WikiSQL的实际应用与自然语言接口(NLI)有很大关系。该领域最著名的工作之一是《PRECISE》[Popescu等人,2003],它将问题转换为SQL查询并识别不确定的问题。Giordani和Moschitti[2012]提出了另一种将问题转换成SQL的系统,该模型首先从语法生成候选查询,然后使用树内核对查询进行排序。这两种方法都依赖于高质量的语法,不适合需要泛化到新模式的任務(例如WikiSQL)。Iyer等人[2017]也提出了翻译成SQL的系统,但使用了Seq2Seq模型,该模型进一步改进了人类反馈。与该模型相比,我们表明了Seq2SQL显著优于Seq2Seq,并且在训练中使用强化学习代替人工反馈。

[0189] 3. 模型

[0190] Seq2SQL模型从自然语言问题和表模式¹生成SQL查询。一个强大的基线模型是用于机器翻译的Seq2Seq模型[Babdanau等人,2015]。但是,Seq2Seq模型中的标准softmax的输出空间对于这个任务来说太大了。特别是,我们注意到,通过将生成的序列的输出空间限制为表模式、问题表达和SQL关键字的联合,可以极大地简化问题。得到的模型类似于带有增强的输入的指针网络[Vinyals等人,2015]。在我们对Seq2SQL的描述中,我们首先描述了该增强的指针网络模型,然后解决了它的局限性,特别是在生成无序查询条件方面。

¹ 为了简化,我们将表模式定义为表中列的名称。

[0191] 3.1增强的指针网络

[0192] 增强的指针网络通过从输入序列中进行选择来逐个令牌地生成SQL查询。在我们的例子中,输入序列是选择列所需的列名称和查询条件所需的查询条件列、问题以及有限的SQL语言词汇表(如SELECT、COUNT等)的串接。在图2所示的例子中,列名令牌包括“Pick”、“#”、“CFL”、“Team”等;问题令牌包括“How”、“many”、“CFL”、“teams”等;SQL令牌包括SELECT、WHERE、COUNT、MIN、MAX等。使用这个增强的输入序列,指针网络能够通过从输入中

专门选择来生成SQL查询。

[0193] 如图2所示,假设我们有包含N个表列的列表和问题,并被要求生成相应的SQL查询。令 $x_j^c = [x_{j,1}^c, x_{j,2}^c, \dots, x_{j,T_j}^c]$ 表示第j列名称中的词序列,其中 $x_{j,i}^c$ 表示第j列中的第i个词, T_j 表示第j列中的词的总数。同样,让 x^q 和 x^s 分别表示问题中的词序列和SQL词汇表中的所有唯一词的集合。

[0194] 我们将输入序列x定义为所有列名、问题和SQL词汇表的串接:

$$[0195] \quad x = [\langle \text{col} \rangle; x_1^c; x_2^c; \dots; x_N^c; \langle \text{sql} \rangle; x^s; \langle \text{question} \rangle; x^q] \quad (1)$$

[0196] 其中 $[a; b]$ 表示序列a和b之间的串接,我们在所有三个序列之间添加特殊哨兵令牌以划分边界。

[0197] 该网络首先使用两层、双向的长短期记忆网络[Hochreiter&Schmidhuber,1997]对x进行编码。编码器的输入是与输入序列中的词相对应的嵌入。我们用h表示编码器的输出,其中 h_t 是编码器对应于输入序列中第t个词的状态。为了简单起见,我们没有写出LSTM等式,该等式曾由Hochreiter和Schmidhuber[1997]描述。然后,我们将与Vinyals等人[2015]提出的类似的指针网络应用于输入编码h。

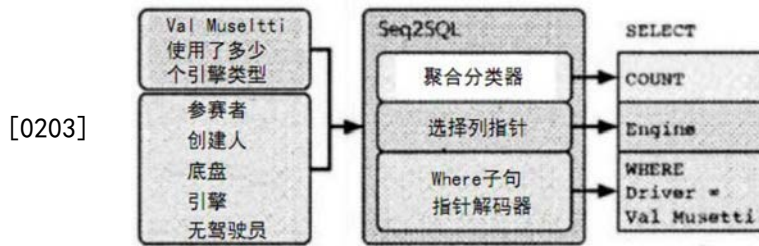
[0198] 解码器网络使用两层、不定向的LSTM。在每个解码器步骤s中,解码器LSTM将前一个解码步骤中生成的查询令牌 y_{s-1} 作为输入,输出状态 g_s 。接下来,解码器对输入序列x的每个位置t产生一标量注意力分数 $\alpha_{s,t}^{ptr}$:

$$[0199] \quad \alpha_{s,t}^{ptr} = W^{ptr} \tanh(U^{ptr} g_s + V^{ptr} h_t) \quad (2)$$

[0200] 解码器选择得分最高的输入令牌作为SQL查询的下一个令牌,即

$$[0201] \quad y_s = \operatorname{argmax}(\alpha_s^{ptr}).$$

[0202] 3.2 Seq2SQL



[0204] 图3:Seq2SQL模型有三个成分,对应于SQL查询(右)的三个部分。模型的输入是问题(左上)和表的列名称(左下)。

[0205] 虽然增强的指针模型能够解决SQL生成问题,但它没有利用SQL查询固有的结构。通常,如图3所示的SQL查询由三个部分组成。第一个成分是聚合算子(在本例中为COUNT),它生成SQL查询选择的行的摘要。或者查询可不请求汇总统计信息,在这种情况下不提供聚合算子。第二个成分是SELECT列(在本例中是Engine),它标识返回结果中包含的列。第三个成分是查询的WHERE子句,在本例中,Driver=Val Musetti,其包含筛选行所需的条件。这里,我们只想要driver为“Val Musetti”的行。

[0206] 为了利用SQL查询中存在的结构,我们引入了Seq2SQL。如图3所示,Seq2SQL有三个部分,分别对应于聚合算子、SELECT列和WHERE子句。首先,网络对查询的聚合运算进行分

类,并添加了与无聚合对应的空操作。接下来,网络指向输入表中与SELECT列对应的列。最后,网络使用指针网络为查询生成条件。前两个成分使用交叉熵损失进行监督,而第三个生成成分使用策略梯度强化学习进行训练,以解决查询条件的无序性(我们在第3.2节对此进行了详细解释)。类似于从Seq2seq移动到增强的指针模型,利用SQL查询的结构使得Seq2SQL可以进一步削减SQL查询的输出空间。我们表明了,与Seq2Seq模型和指针模型相比,这导致了显著更高的性能。

[0207] 聚合运算

[0208] 聚合运算取决于问题。对于图3所示的示例,我们认为正确的算子是COUNT,因为问题要求“How many”。

[0209] 为了计算聚合运算,我们首先计算输入表示 k^{agg} ,和等式(2)类似,我们首先针对输入序列中的每个第 t 个令牌计算标量注意力分数 $\alpha_t^{inp} = W^{inp} h_t^{enc}$ 。然后将所有分数的向量 $\alpha^{inp} = [\alpha_1^{inp}, \alpha_2^{inp}, \dots]$ 归一化,以产生输入序列 x 中所有令牌的分布 $\beta^{inp} = \text{softmax}(\alpha^{inp})$ 。输入表示 k^{agg} 是由相应的归一化的注意力权重加权的列编码的和:

$$[0210] \quad k^{sel} = \sum_t \beta_t^{inp} h_t^{enc} \quad (3)$$

[0211] 让 α^{agg} 表示聚合运算(如COUNT、MIN、MAX)和空聚合运算NULL的得分。我们通过将多层感知器应用于输入表示 k^{sel} 来计算 α^{agg} :

$$[0212] \quad \alpha^{agg} = W^{agg} \tanh(V^{agg} k^{agg} + b^{agg}) + c^{agg} \quad (4)$$

[0213] 然后通过应用softmax函数得到可能的聚合运算集合上的分布 $\beta^{agg} = \text{softmax}(\alpha^{agg})$ 。我们使用交叉熵损失 L_{agg} 进行聚合运算。

[0214] 选择列

[0215] 选择列取决于表列和问题。即对于图3中的示例,我们从问题的“多少个引擎类型”部分可以看到查询需要“Engine(引擎)”列。选择列预测是一个匹配问题,可以使用指针解决:给定列表示和问题表示的列表,我们选择最匹配问题的列。为了产生列的表示,我们首先用LSTM对每个列名称进行编码。特定列 j 的表示 e_j^c 由下式给出:

$$[0216] \quad h_{j,t}^c = LSTM(\text{emb}(x_{j,t}^c), h_{j,t-1}^c) \quad e_j^c = h_{j,T_j}^c \quad (5)$$

[0217] 这里, $h_{j,t}^c$ 表示第 j 列的第 t 个编码器状态。列 j 的表示 e_j^c 被认为是最后一个编码器状态。

[0218] 为了构造问题的表示,我们使用关于 k^{agg} (方程3)的但具有不固定的权重的相同的体系架构计算另一个输入表示 k^{sel} 。最后,我们在列表示上应用多层感知器,根据输入表示,来计算每一列 j 的分数:

$$[0219] \quad \alpha_j^{sel} = W^{sel} \tanh(V^{sel} k^{sel} + V^c e_j^c) \quad (6)$$

[0220] 我们使用softmax函数对分数进行归一化,以产生可能的选择列 $\beta^{sel} = \text{softmax}(\alpha^{sel})$ 上的分布。对于图3所示的示例,分布是在“Entrant”、“Constructor”、“Chassis”、“Engine”、“No”和地面实况SELECT列“Driver”上的分布。利用交叉熵损失 L^{sel} 对SELECT网络进行训练。

[0221] WHERE子句

[0222] 可以使用类似于3.1节中描述的指针解码器来训练WHERE子句。然而,利用交叉熵

损失优化网络存在关键的局限性：可以任意交换查询的WHERE条件，并且查询仍产生相同的结果。假设我们有一个问题“年龄大于18岁的男性的名字”，查询语句：SELECT name FROM insurance WHERE age>18AND gender=“male”和SELECT name FROM insurance WHERE gender=“male”AND age>18。尽管没有精确的字符串匹配，这两个查询都获得正确的执行结果。假设将前一个查询作为地面实况提供，利用交叉熵损失来监督生成，则会错误地惩罚后一个查询。为了解决这个问题，我们使用强化学习来学习一个策略，以直接优化执行结果的期望“正确性”（公式7）。

[0223] 在每个步骤中，我们从输出分布中取样，以获得下一个令牌，而不是老师强制这样做。在生成过程的最后，我们对数据库执行生成的SQL查询以获得奖励。让 $q(y)$ 表示由模型生成的查询， q_g 表示与问题相对应的地面实况查询。奖励 $R(q(y), q_g)$ 定义为：

$$[0224] \quad R(q(y), q_g) = \begin{cases} -2, & \text{如果 } q(y) \text{ 不是有效的 SQL 查询。} \\ -1, & \text{如果 } q(y) \text{ 是有效的 SQL 查询并执行得到不正确的结果} \\ +1, & \text{如果 } q(y) \text{ 是有效的 SQL 查询并执行得到正确的结果。} \end{cases} \quad (7)$$

[0225] 让 $y = [y^1, y^2, \dots, y^T]$ 表示在SQL查询 $q(y)$ 的WHERE子句中生成的令牌序列，其导致执行奖励为 $R(q(y), q_g)$ 。损失 $L^{\text{whe}} = -E_y[R(q(y), q_g)]$ 是可能的WHERE子句上的负的期望奖励。

[0226] 如Schulman等人[2015]所示，在网络的前向传递期间采样的行为导致反向传递期间合成梯度的相应注入，该合成梯度是奖励的函数。具体地，让 $p_t(y^t)$ 表示在第 t 步中选择令牌 y^t 的概率，对应的合成梯度为 $R(q(y), q_g) \log P_t(y^t)$ 。

[0227] 混合目标函数

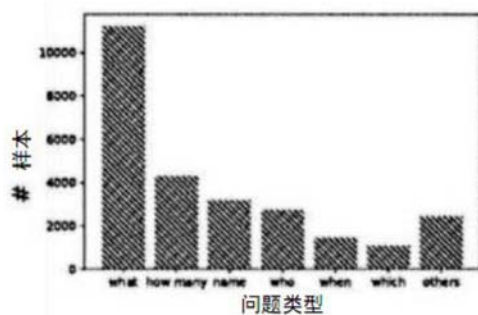
[0228] 利用梯度下降法对模型进行训练，以使目标函数 $J^{\text{obj}} = L^{\text{agg}} + L^{\text{sel}} + L^{\text{whe}}$ 最小化。因此，反向传递期间的总梯度是来自预测SELECT列时的交叉熵损失、预测聚合运算时的交叉熵损失和策略学习时的交叉熵损失的梯度的等同加权和。

[0229] 4. WikiSQL

[0230] WikiSQL是自然语言表达和相应SQL查询和SQL表的集合。据我们所知，这是目前所能找到的最大的语料库。如图2所示的WikiSQL中的单个示例，包含SQL表、SQL查询以及与SQL查询对应的自然语言表达。这项工作中，我们不使用SQL表的内容，只使用它的模式（例如，列名称）。这意味着在WikiSQL上训练的系统不需要在推理期间访问最终用户的数据库内容。

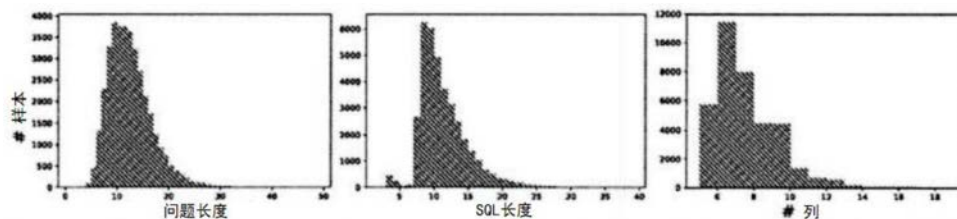
[0231] 表1示出了WikiSQL如何与相关数据集进行比较。也就是说，我们注意到，WikiSQL是迄今为止最大的手工注释语义解析数据集——无论是从示例的数量还是表模式的数量来看，它都比其他具有逻辑形式的数据集大一个数量级。而且，WikiSQL中的查询跨越了大量的表模式并覆盖多种数据。大量的模式相比其他数据集提出了挑战：模型必须不仅能够泛化到新的查询，而且能够泛化到新的模式。最后，WikiSQL包含从web提取的挑战性的真实数据。这在列数、问题长度和查询长度的分布中很明显（分别如图5所示）。数据集中各种问题的另一个指标是问题类型的分布（如图4所示）。

[0232]



[0233] 图4:WikiSQL中问题类型的分布

[0234]



[0235] 图5:WikiSQL中列数、每问题令牌和每查询令牌的分布。

[0236] WikiSQL是使用Amazon Mechanical Turk (AMT) 上的众包收集的,分为两个阶段。首先,工作人员将生成的问题改写为表格。生成的问题本身使用模板形成,使用随机生成的SQL查询填充。由于HTML表的变化很大,我们通过只保留那些合法的数据库表和足够大的行数和列数,从而确保了表的有效性和复杂性。在第二阶段,另外两名工作人员被要求验证意译与生成的问题具有相同的含义。我们丢弃那些没有显示出足够变化的意译,如通过与生成的问题具有字符编辑距离所测量的,以及那些在验证阶段期间被工作人员认为不正确的。关于如何收集WikiSQL的更多详细信息显示在附录的A部分。我们提供了意译阶段²和验证阶段³使用的接口示例。接口的HTML页面也包含在补充资料中。

² WikiSQL 使用的意译接口的示例: <https://tableqa.herokuapp.com/example/2562572-25>

³ WikiSQL 使用的验证接口的示例

例: <https://paraphrase.herokuapp.com/example/3VHP9MDGR03FQ3XNGYAFKJDNCFZ>

[0237] 这些表和它们相应的意译和SQL查询被随机地插入到训练、偏移和测试分支中,这样每个表都以一个分支的形式出现。除了原始表内容、查询、结果和自然表达之外,我们还为查询系统发布了相应的SQL数据库和执行引擎。

[0238] 4.1评价

[0239]

数据集	大小	LF	模式
WikiSQL	37000	yes	9500
Geoquery	880	yes	8
ATIS	5871	yes*	141
Freebase917	917	yes	81*
Overnight	26098	yes	8
WebQuestions	5810	no	2420
WikiTableQuestions	22033	no	2108

[0240] 表1:WikiSQL与现有数据集之间的比较。数据集包括GeoQuery880[Tang&Mooney, 2001]、ATIS[Price,1990]、Free917[Cai&Yates,2013]、Overnight[Wang等人,2015]、WebQuestions[Berant等人,2013]和WikiTableQuestions[Pasupat&Liang,2015]。“大小

(Size)”表示数据集中的示例的总数量。“LF”表示数据集是否有带注释的逻辑形式。“模式 (Schema)”表示数据集中域或模式的数量。ATIS的格式被呈现为插槽填充任务。每个 Freebase API页面都被视为一个单独的域。

[0241] 评估WikiSQL的一种自然方法是使用执行精度度量。设 N 为数据集中的示例总数， N_{ex} 为执行后得到正确结果的查询总数。执行精度定义为 $Acc_{ex} = N_{ex} / N$ 。

[0242] 另一个度量是逻辑形式的准确性。设 N_{lf} 表示查询总数与用于收集意译的地面实况查询的字符串完全匹配。逻辑形式精确度定义为 $Acc_{lf} = N_{lf} / N$ 。

[0243] 上述两个度量都有优点和缺点。正如我们将在第3.2节中所示的，逻辑形式的精确性不正确地惩罚了实现正确结果但与地面实况查询没有精确字符串匹配的查询。替代地，可以构造一个与自然语言问题相对应但不会得到相同的结果的SQL查询。两个不同的查询产生相同结果的示例为：SELECT COUNT (name) WHERE SSN=123和SELECT COUNT (SSN) WHERE SSN=123。由于上述这些观察，我们使用这两个度量来评估模型。

[0244] 5实验

[0245] 数据集使用Stanford CoreNLP分词，我们使用归一化的令牌进行训练，并在输出查询之前恢复为原始注释。通过这种方式，查询由来自原始注释的令牌组成，因此可以在数据库中执行。

[0246] 我们使用来自Glove [Pennington等人, 2014]的预先训练的词嵌入和来自 [Hashimoto等人, 2016]的字符n-gram嵌入。设 w_x^g 为GloVe嵌入， w_x^c 为词 x 的字符嵌入。其中 w_x^c 为 x 中所有字符n-gram的嵌入的均值。对于一个给定的，我们将其词嵌入 $w_x = [w_x^g; w_x^c]$ 定义为其GloVe嵌入及其字符嵌入的串接。对于没有预先训练的嵌入的词，我们将嵌入赋值零向量。我们不训练嵌入，而是固定预先训练的嵌入。

[0247] 所有网络的运行时间最大为100个时期 (epoch)，并提前停止使用dev分离器上的执行精度标准。我们使用ADAM [Kingma&Ba, 2014]进行训练，并使用dropout [Srivastava等人, 2014]进行调节。我们使用Pytorch⁴来实现所有的模型。为了训练Seq2SQL，我们首先预先训练一个版本，其中WHERE子句是通过教师强迫进行监督的 (例如，采样程序不是从零开始训练)，然后使用强化学习继续训练。为了获得第3.2节中描述的奖励，我们使用了第4节中描述的查询执行引擎。

⁴ <https://pytorch.org>

[0248] 5.1结果

[0249] 我们将结果与Berant等人 [2013]使用的用于机器翻译的注意力序列到序列基线进行比较。这个模型已经在附件B部分详细地描述。我们还提出了消除Seq2SQL的方法，在这个方法中，SQL查询的WHERE子句使用交叉熵损失 (例如教师强迫) 进行训练，其方式与增强的指针网络相同。三种模型的性能如表2所示。

模型	Dev Acc _{lf}	Dev Acc _{ex}	Test Acc _{lf}	Test Acc _{ex}
注意力Seq2seq	20.3%	22.4%	20.4%	21.8%
增强的指针网络	38.5%	43.5%	37.7%	43.2%
Seq2SQL (无 RL)	46.2%	49.9%	44.4%	47.3%
Seq2SQL	48.3%	54.2%	46.6%	52.2%

[0251] 表2: WikiSQL上的性能。这两个度量都在4.1节中定义。当与Seq2SQL相比时，除了

WHERE子句与强化学习相比时通过教师强迫是受监督的以外,Seq2SQL (无RL) 模型在架构中是相同的。

[0252] 我们注意到,通过利用增强的指针网络来减少输出空间,对注意力序列到序列模型显著地改进了21.4%。而且,利用SQL查询的结构可以带来另外的4.1%的显著改进,这是与增强的指针网络相比,没有RL的Seq2SQL的性能所示的。最后,完整的Seq2SQL模型,使用基于数据库中实时查询执行的奖励的强化学习进行训练,导致了另外显著的性能4.9%。

[0253] 5.2分析

[0254] 限制输出空间获得更精确的条件。当我们比较注意力序列到序列模型和增强的指针网络的输出时,我们发现后者产生的WHERE子句条件的质量更高。例如,对于问题“1850年3月4日有多少个选区的继任者就职?”,前者产生“SELECT COUNT District WHERE Date successor seated=seated march 4”,但是后者产生“SELECT COUNT District WHERE Date successor seated=seated march 4.1850”。类似地,对于“道格·巴塔利亚的编号是多少?”,前者产生“SELECT pick WHERE player=doug”,但是后者产生“SELECT pick WHERE player=doug battalia”。一个关键的原因是条件往往是由罕见的词组成(例如,1850出现在语料库非常罕见)。但是Seq2SQL模型随后倾向于生成一个条件,该条件包含训练语料库中经常出现的令牌,比如日期为“3月4日”或玩家姓名为“doug”。指针网络不会面对该问题,因为输出分布从该输入序列专门构建。

[0255] 合并结构减少了无效的查询。正如预期的那样,将简单的选择和聚合结构合并到模型中可以显著降低生成的无效SQL查询的百分比,从39.8%降低到7.1%。通常,我们发现大量无效的查询结果来自无效的列名称。也就是说,模型生成引用表中不存在的列的查询。当列名包含许多令牌时这尤其有用,例如“Miles (km)”,其经过分词后有4个令牌。为聚合引入专门的分类器也可以减少由于错误的聚合算子而导致的错误率。表3示出了,添加聚合分类器显著提高了预测COUNT算子的查全率。对于不同模型产生的更多查询,请参见附录的C部分。

[0256] 强化学习产生更高质量的WHERE子句,它的顺序有时与地面实况(groud truth)不同。如预期的,采用策略学习训练的模型获得了正确的结果,其中条件的顺序不同于地面实况查询的顺序。例如,对于问题“1992年民主党候选人在哪个地区首次当选?”地面实况条件是“WHERE First elected=1992AND Party=Democratic”,而Seq2SQL生成“WHERE Party=Democratic AND First elected=1992”。更一般地说,我们注意到,在大多数情况下,如果Seq2SQL是正确的,而没有策略学习的Seq2SQL不是正确的,会产生错误的WHERE子句。例如,对于相当复杂的问题“新泽西特伦顿第12轮比赛中,a.j.福伊特获得杆位的比赛叫什么名字?”,不经过强化学习训练的Seq2SQL产生WHERE子句“WHERE rnd=12and track=a.j.foyt AND pole position=a.j.foyt”,而基于策略梯度的Seq2SQL可以正确地生成子句“WHERE rnd=12AND pole position=a.j.foyt.”。

	模型	精确度	查全率	F1
[0257]	增强的指针	87.0%	47.8%	61.7%
	Seq2SQL	85.9%	79.7%	82.7%

[0258] 表3:对COUNT聚合算子的性能。Seq2SQL模型包括COUNT算子与地面实况查询包括

COUNT算子相比。

[0259] 困难的表模式带来了挑战。在许多HTML表中,列名称不能准确地描述列的内容。例如,在开发集中的一个表中,来自布朗克斯区的投票数被列在标题为“the bronx”的列中。当我们问“有多少来自布朗克斯的选民投票给了社会党”这个问题时,由于短语“有多少”,模型错误地生成了聚合算子COUNT。然而,由于此表的组织方式,投票者数量实际上在列中列出,COUNT运算是多余的。另一个困难的例子是问题“Plymouth的一辆车在2月9日赢得了哪一年的比赛”,因为模型不明白“Plymouth”指的是汽车制造。

[0260] 6. 结论

[0261] 我们提出了Seq2SQL,这是一个将自然语言问题转换成相应SQL查询的深度神经网络,它利用SQL查询的结构来显著减少生成的查询的输出空间。作为Seq2SQL的一部分,我们提出了使用循环查询执行来学习生成SQL查询条件的策略,该策略的性质是无序的,不适合通过交叉熵损失进行优化。除了Seq2SQL,我们还引入了WikiSQL,这是一个由自然语言问题和SQL查询组成的数据集,它比可比数据集大一个数量级。最后,我们证明了Seq2SQL显著优于WikiSQL上的注意力序列到序列模型,执行精度从21.8%提高到52.2%,逻辑形式精度从20.4%提高到46.6%。在未来的工作中,我们计划投资更复杂的查询,如嵌套的查询和更复杂的奖励函数。

[0262] 参考文献

[0263] I.Androutsopoulos,G.D.Ritchie,and P.Thanisch.Natural language interfaces to databases-an introduction.1995.

[0264] Y.Artzi and L.S.Zettlemoyer.Bootstrapping semantic parsers from conversations.In EMNLP,2011.

[0265] Y.Artzi and L.S.Zettlemoyer.Weakly supervised learning of semantic parsers for mapping instructions to actions.TACL,1:49-62,2013.

[0266] D.Bahdanau and K.Cho.and Y.bengio.Neural machine translation by jointy learning to align and translate.ICLR,2015

[0267] T.Beck,A.Demirguc,-Kunt,and R.Levine.A new database on the structure and developmentof the financial sector.The World Bank Economic Review,14(3): 597-605,2000.

[0268] I.Bello,H.Pham,Q.V.Le,M.Norouzi,and Samy Bengio.Neural combinatorialoptimization with reinforcement learning.ICLR,2017.

[0269] J.Berant,A.Chou,R.Frostig,and P.Liang.Semantic parsing on freebase from question-answer pairs.In EMNLP,2013.

[0270] C.Bhagavatula,T.Noraset,and D.Downey.Methods for exploring and mining tables on wikipedia.In IDEA@KDD,2013.

[0271] Q.Cai and A.Yates.Large-scale semantic parsing via schema matching and lexiconextension.In ACL,2013.

[0272] L.Dong and M.Lapata.Language to logical form with neural attention.ACL,2016.

[0273] A.Giordani and A.Moschitti.Translating questions to SQL queries with

generative parsers discriminatively reranked. In COLING, 2012.

[0274] J. Gu, Z. Lu, H. Li, and V. O. K. Li. Incorporating copying mechanism in sequence-to-sequence learning. ACL, 2016.

[0275] K. Hashimoto, C. Xiong, Y. Tsuruoka, and R. Socher. A Joint Many-Task Model: Growing a Neural Network for Multiple NLP Tasks. arXiv, cs.CL 1611.01587, 2016.

[0276] R. Hillestad, J. Bigelow, A. Bower, F. Girosi, Robin Meili, Richard Scoville, and R. Taylor. Can electronic medical record systems transform health care? potential health benefits, savings, and costs. Health affairs, 24(5):1103-1117, 2005.

[0277] S. Hochreiter and J. Schmidhuber. Long short-term memory. Neural computation, 1997.

[0278] S. Iyer, I. Konstas, A. Cheung, J. Krishnamurthy, and L. Zettlemoyer. Learning a neural semantic parser from user feedback. In ACL, 2017.

[0279] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. arXiv, abs/1412.6980, 2014.

[0280] G. Klein, Y. Kim, Y. Deng, J. Senellart, and A. M. Rush. OpenNMT: Open-Source Toolkit for neural machine translation, Arxiv. abs, 2017

[0281] P. Liang, M. I. Jordan, and D. Klein. Learning dependency-based compositional semantics. Computational Linguistics, 39:389-446, 2011.

[0282] T. Luong, H. Pham, and C. D. Manning. Effective approaches to attention-based neural machine translation. In EMNLP. 2015.

[0283] S. Merity, C. Xiong, J. Bradbury, and R. Socher. Pointer sentinel mixture models. ICLR, 2017.

[0284] R. Nallapati, B. Zhou, C. dos Santos, C. Glanz, and J. R. Glaser. Compositional semantics. Computational linguistics. In EMNLP. 2015.

[0285] A. Neelakantan, Q. V. Le, M. Abadi, Andrew McCallum, and Dario Amodei. Learning a natural language interface with neural programmer. In ICLR, 2017.

[0286] E. W. Ngai, L. Xiu, and D. C. Chau. Application of data mining techniques in customer relationship management: A literature review and classification. Expert systems with applications, 36(2):2592-2602, 2009.

[0287] P. Pasupat and P. Liang. Compositional semantic parsing on semi-structured tables. In ACL, 2015.

[0288] R. Paulus, C. Xiong, and R. Socher. A deep reinforced model for abstractive summarization. arXiv preprint arXiv:1705.04304, 2017.

[0289] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In EMNLP, 2014.

[0290] A. -Maria Popescu, O. Etzioni, and H. Kautz. Towards a theory of natural language interfaces to databases. In Proceedings of the 8th International

Conference on Intelligent User Interfaces, pp.149-157. ACM, 2003.

[0291] P.J.Price.Evaluation of spoken language systems:The ATIS domain.1990.

[0292] S.Reddy,M.Lapata,and M.Steedman.Large-scale semantic parsing without questionanswer pairs.TACL,2:377-392,2014.

[0293] J.Schulman,N.Heess,Theophane Weber,and Pieter Abbeel.Gradient estimation using stochastic computation graphs.In NIPS,2015.

[0294] M.J.Seo,A.Kembhavi,A.Farhadi,and H.Hajishirzi.Bidirectional attention flow for machine comprehension.ICLR,2017.

[0295] N.Srivastava,G.E.Hinton,A.Krizhevsky,I.Sutskever,and R.Salakutdinov.Dropout:a simple way to prevent neural networks from overfitting.Journal of Machine Learning Research,15:1929-1958,2014.

[0296] J.Starc and D.Mladenec.Joint learning of ontology and semantic parser from text.Intelligent Data Analysis,21:19-38,2017.

[0297] I.Sutskever,O.vinyals.and Q.V.Le.sequence to sequence learning with neural networks.In NIPS.2014

[0298] L.R.Tang and R.J.Mooney.Using multiple clause constructors in inductive logic programming for semantic parsing.In ECML,2001.

[0299] O.Vinyals,M.Fortunato,and N.Jaitly.Pointer networks.In NIPS,2015.

[0300] Y.Wang,J.Berant,and P.Liang.Building a semantic parser overnight.In ACL,2015.

[0301] Y.W.Wong and R.J.Mooney.Learning synchronous grammars for semantic parsing with lambda calculus.In ACL,2007.

[0302] C.Xiong,V.Zhong,and R.Socher.Dynamic coattention networks for question answering.ICLR,2017.

[0303] J.M.Zelle and R.J.Mooney.Learning to parse database queries using inductive logic programming.In AAAI/IAAI,Vol.2,1996.

[0304] L.S.Zettlemoyer and M.Collins.Learning to map sentences to logical form:Structured classification with probabilistic categorial grammars.In Uncertainty in Artificial Intelligence,2005.

[0305] L.S.Zettlemoyer and M.Collins.Online learning of relaxed ccg grammars for parsing to logical form.In EMNLP-CoNLL,2007.

[0306] A WikiSQL的集合

[0307] WikiSQL在意译阶段和验证阶段收集。在释义阶段,我们使用Bhagavatula等人[2013]从Wikipedia中提取的表,并根据以下标准删除小表:

[0308] • 每行中的单元格数不相同

[0309] • 单元格中的内容超过50个字符

[0310] • 标题单元格为空

[0311] • 该表少于5行或5列

[0312] • 一行中超过40%的单元格包含相同的内容

[0313] 我们还删除了表的最后一行,因为大量HTML表的最后一行往往有汇总统计信息,因此最后一行不符合标题行定义的表模式。

[0314] 对于每个通过上述标准的表,我们按照以下规则随机生成6个SQL查询:

[0315] • 该查询遵循格式:SELECT agg_op agg_col from table where cond1_col cond1_op cond1AND cond2_col cond2_op cond2...

[0316] • 聚合算子agg_op可以为空,也可以为COUNT。如果聚合列agg_col是数值型的,则agg_op可以是最大值和最小值之一。

[0317] • 条件算子cond_op是=。如果相应的条件列cond_col是数值型的,则cond_op可以附加地是>和<中的一个。

[0318] • 条件cond可以是表中对应的cond_col下出现的任何可能的值。如果cond_col是数值,则cond可以是列中的最小值到列中的最大值范围内采样的任何数值。

[0319] 我们只生成产生非空结果集的查询。为了强制执行简洁的查询,如果这样做不会改变执行结果,我们将从生成的查询中删除条件。

[0320] 对于每个查询,我们使用模板生成一个粗糙的问题,并通过Amazon Mechanical Turk上的众包获得人工意译。在每个Amazon Mechanical Turk HIT中,都会向一个工作者显示表格以及其生成的问题,并要求每个问题的释义。

[0321] 在从意译阶段获得自然语言释义后,我们将每个问题意译对交给另外两个处于验证阶段的工作者,以验证意译和原问题是否包含相同的意思。

[0322] 然后,我们使用以下标准过滤意译的初始集合:

[0323] • 在验证阶段,至少有一名工作者认为意译是正确的

[0324] • 意译必须与产生的问题有足够的不同,并有一定的字符级别编辑距离大于10

[0325] 图2展示了WikiSQL中的一个示例。其中,对应产生的问题是“约克大学所在的CFL团队的总数是多少”。

[0326] B注意力Seq2Seq基线

[0327] 对于我们的基线,我们使用注意力序列到序列模型,这导致机器翻译[Babdanan等人,2015]和神经语义解析[Dong和Lapata,2016]有了显著的改进。该模型基于luong等人[2015]描述的全局注意力编码器-解码器模型(具有输入馈送)。我们使用基于OpenNMT[Klein等人]的实现。

[0328] 我们使用相同的两层双向堆叠LSTM编码器,如前所述。解码器是两层无方向性的堆叠LSTM。解码器LSTM与方程2所描述的解码器几乎相同,唯一的区别在于输入馈送。

$$[0329] \quad g_s = LSTM([emb(y_{s-1}); (k_{s-1}^{dec})], g_{s-1}) \quad (8)$$

[0330] 其中 d_{hid} 维度 k_s^{dec} 是sth解码步骤期间输入序列上的注意力上下文,计算为:

$$[0331] \quad \alpha_{s,t}^{dec} = h_s^{dec} (W^{dec} h_t^{enc})^T \quad \beta_s^{dec} = \text{soft max}(\alpha_s^{dec}) \quad (9)$$

$$[0332] \quad k_s = \sum_t \beta_{s,t} h_t^{enc} \quad (10)$$

[0333] 为了在第s个解码器步骤中产生输出令牌,将解码器状态和注意力上下文的串接给到最后一个线性层,从而产生目标词汇表中词的分布 α^{dec} 。

$$[0334] \quad \alpha^{dec} = \text{soft max}(U^{dec}[h_s^{dec}; k_s^{dec}]) \quad (11)$$

[0335] 其中, U^{dec} 具有维度 $N^{vocab} \times d_{hid}$, N^{vocab} 是输出词汇表的大小。

[0336] 在训练期间, 教师强迫被使用。在推理过程中, 使用大小为5的波束, 将生成的未知词替换为注意力权重最高的输入词。

[0337] C由Seq2SQL的预测

Q当康涅狄格和维拉诺瓦是常规赛冠军时, 有多少比赛场地(城市)?

P SELECT COUNT tournament player (city) WHERE regular season winner city) = connecticut & villanova

S' SELECT COUNT tournament venue (city) WHERE tournament winner = connecticut & villanova

S SELECT COUNT tournament venue (city) WHERE regular season winner = connecticut & villanova

G SELECT COUNT tournament venue (city) WHERE regular season winner = connecticut & villanova

Q第一回合成绩为0-1的比赛的总分数是多少?

P SELECT aggregate WHERE 1st. = 0-1

S' SELECT COUNT agg. score WHERE 1st leg = 0-1

S SELECT agg. score WHERE 1st leg = 0-1

[0338] G SELECT agg. score WHERE 1st leg = 0-1

Q在新泽西特伦顿举行的第12轮比赛中, a.j.福伊特获得杆位的比赛叫什么名字?

P SELECT race name WHERE location = 12th AND round position = a.j. foyt, new jersey AND

S' SELECT race name WHERE rnd = 12 AND track = a.j. foyt AND pole position = a.j. foyt

S SELECT race name WHERE rnd = 12 AND pole position = a.j. foyt

G SELECT race name WHERE rnd = 12 AND pole position = a.j. foyt

Q哪个城市在89.9?

P SELECT city WHERE frequency = 89.9

S' SELECT city of license WHERE frequency = 89.9

S SELECT city of license WHERE frequency = 89.9

G SELECT city of license WHERE frequency = 89.9

Q布朗克斯有多少选民投给了社会党?

P SELECT MIN % party = socialist

S' SELECT COUNT the bronx where the bronx = socialist

S SELECT COUNT the bronx WHERE the bronx = socialist

G SELECT the bronx WHERE party = socialist

Q哪一年2月9日普利茅斯的一辆车赢了?

[0339]

P SELECT MIN year (km) WHERE date = february 9 AND race time
= plymouth 9

S' SELECT year (km) WHERE date = plymouth 9 AND race time =
february 9

S SELECT year (km) WHERE date = plymouth 9 AND race time=
february 9

G SELECT year (km) WHERE manufacturer = plymouth AND date =
february 9

[0340] 表4:模型在开发部分的预测示例。Q表示自然语言问题,G表示相应的地面实况查询。P、S' 和S分别表示由增强的指针网络Seq2SQL产生的查询,而不使用强化学习Seq2SQL。为了简洁,我们省略了查询的FROM表部分。

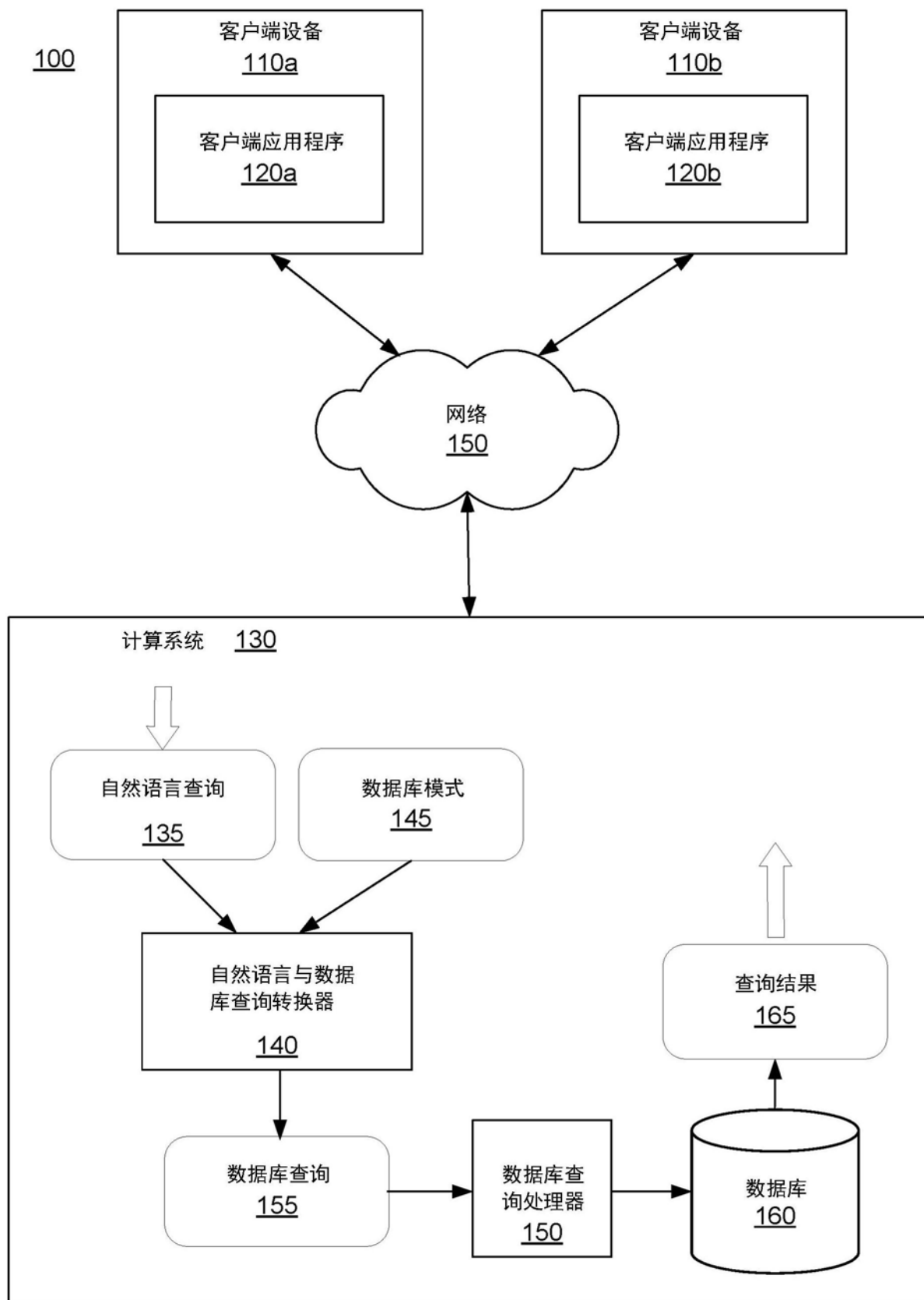


图1

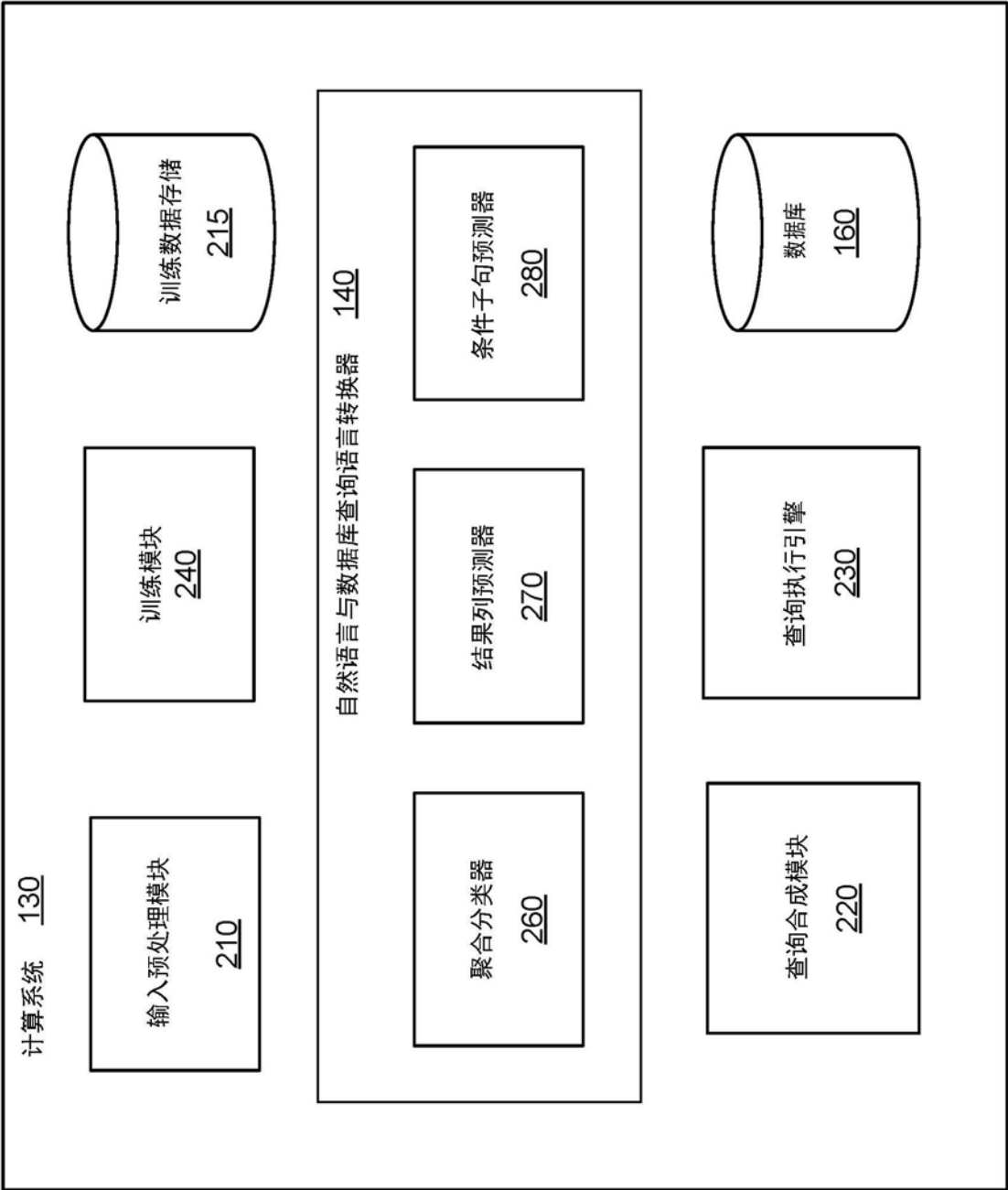


图2

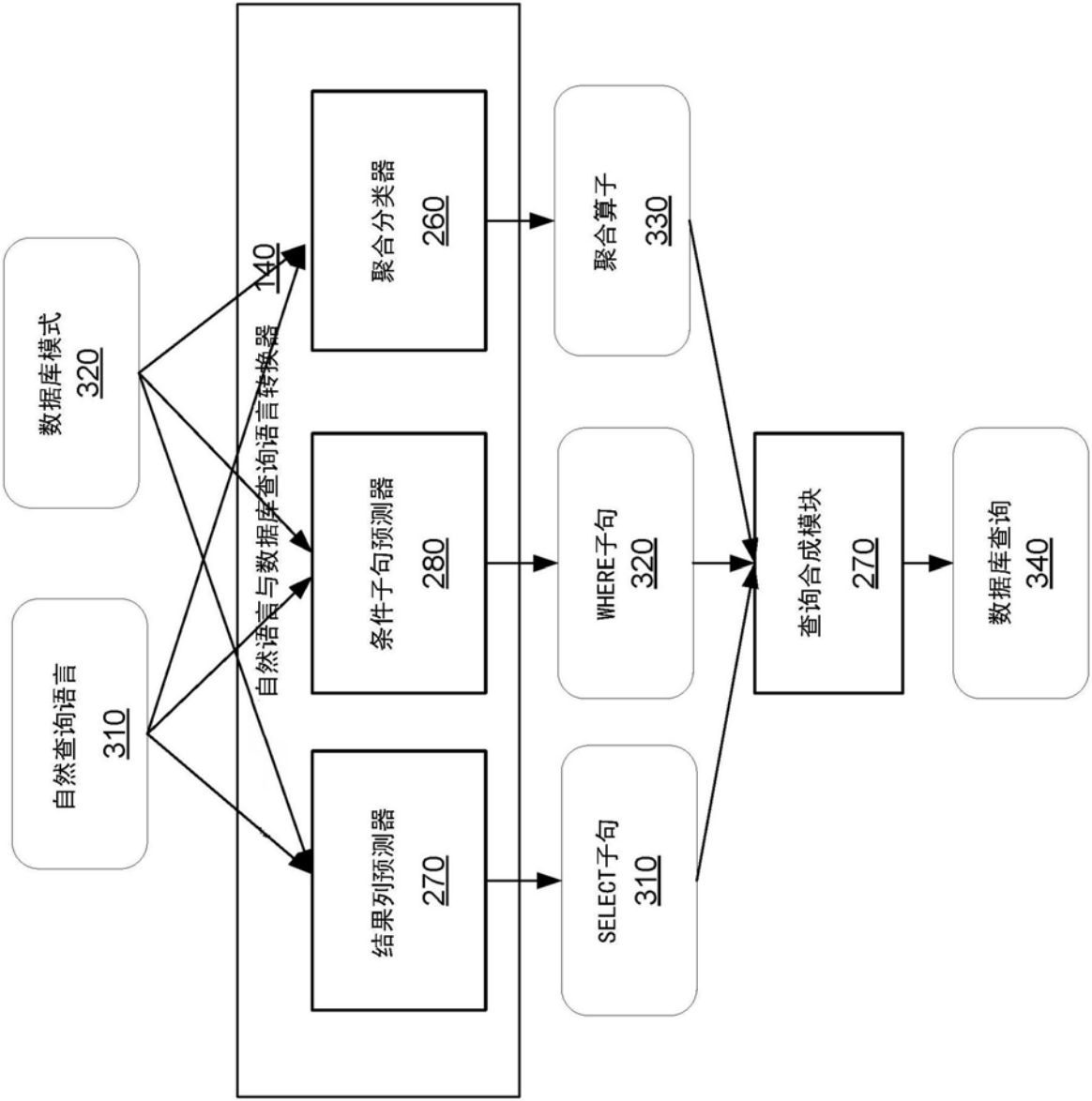


图3

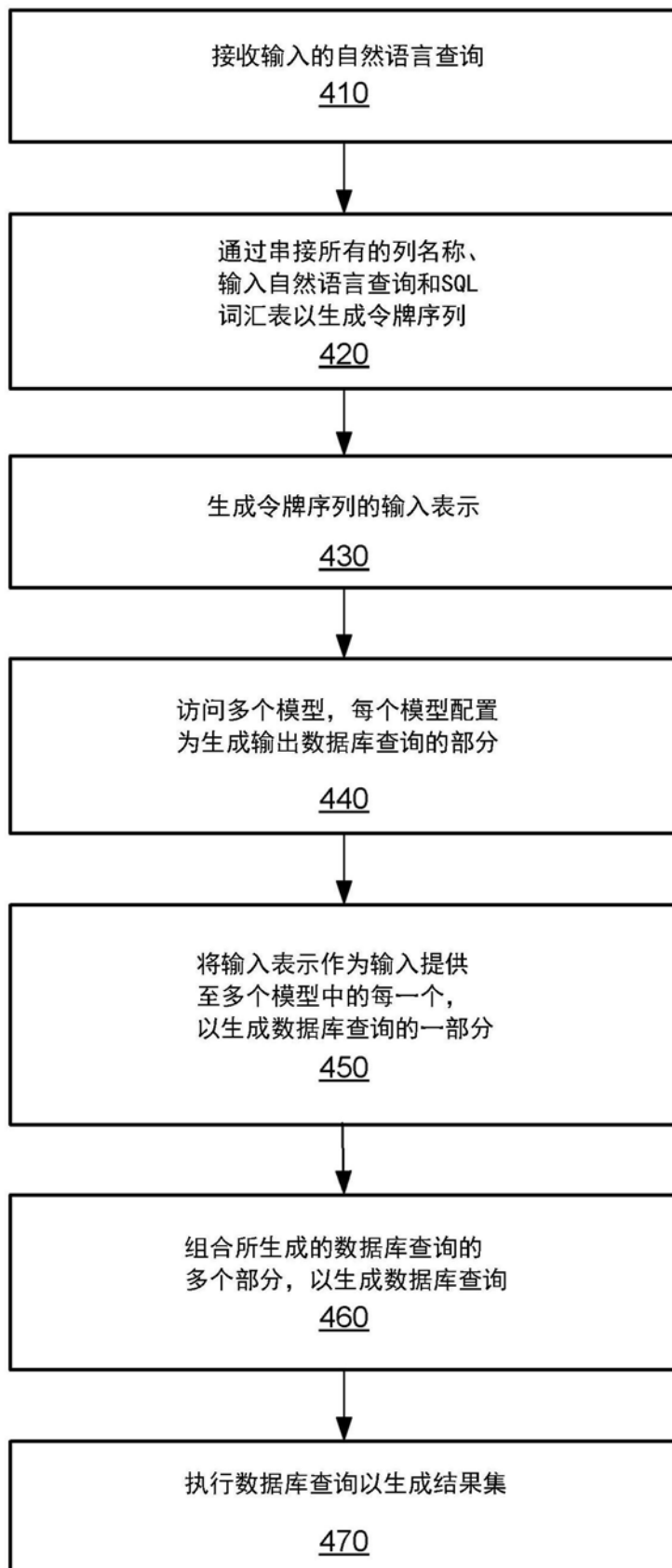


图4

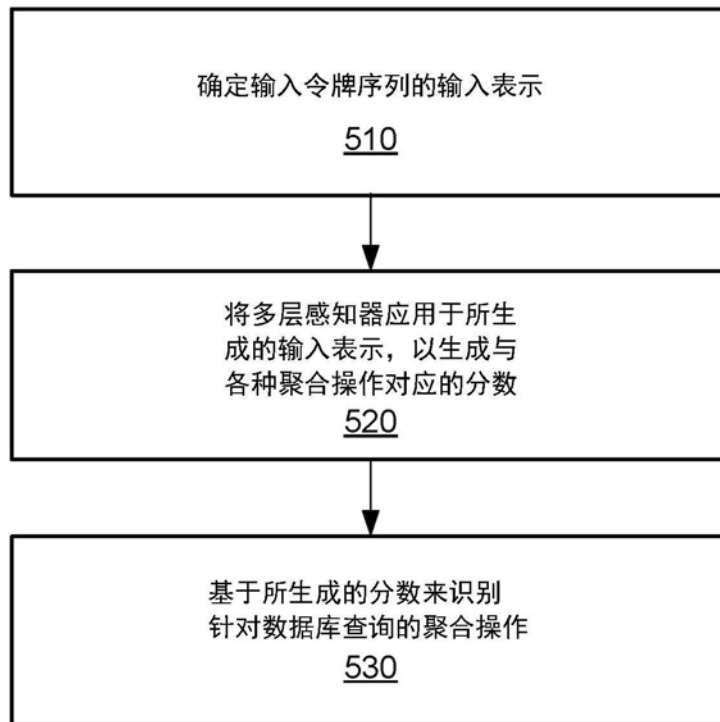


图5

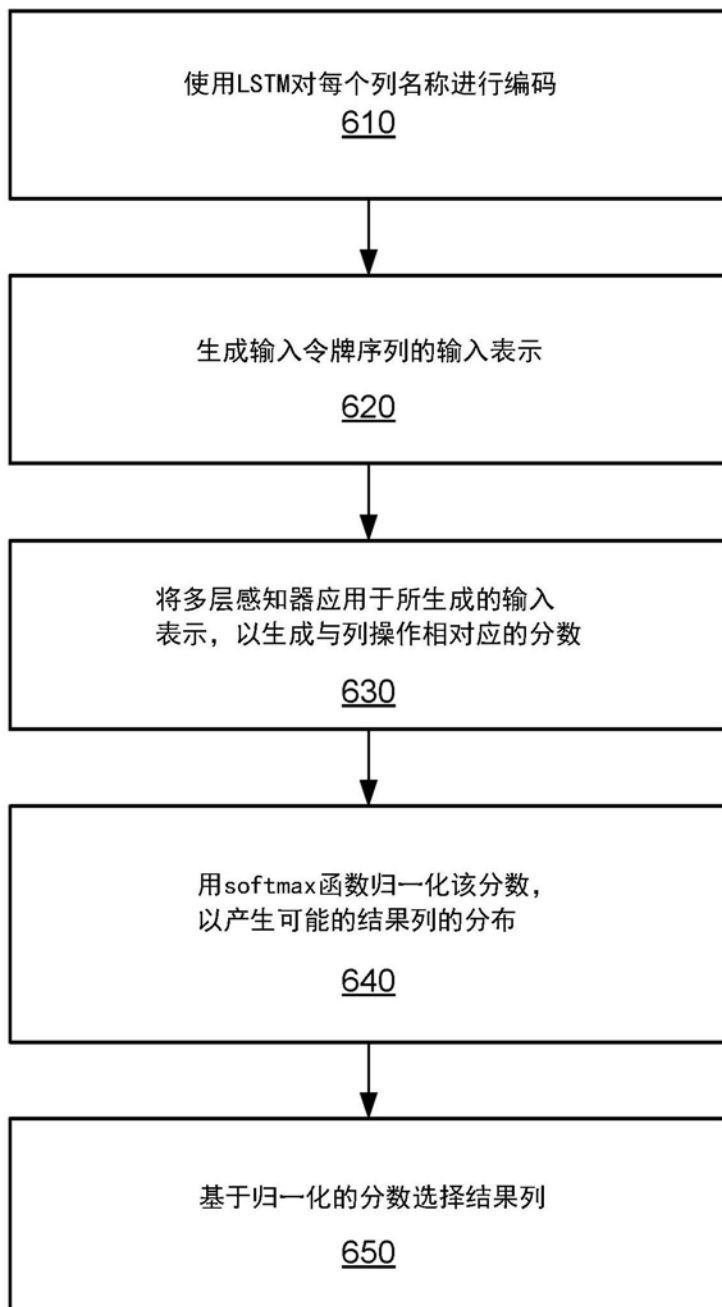


图6

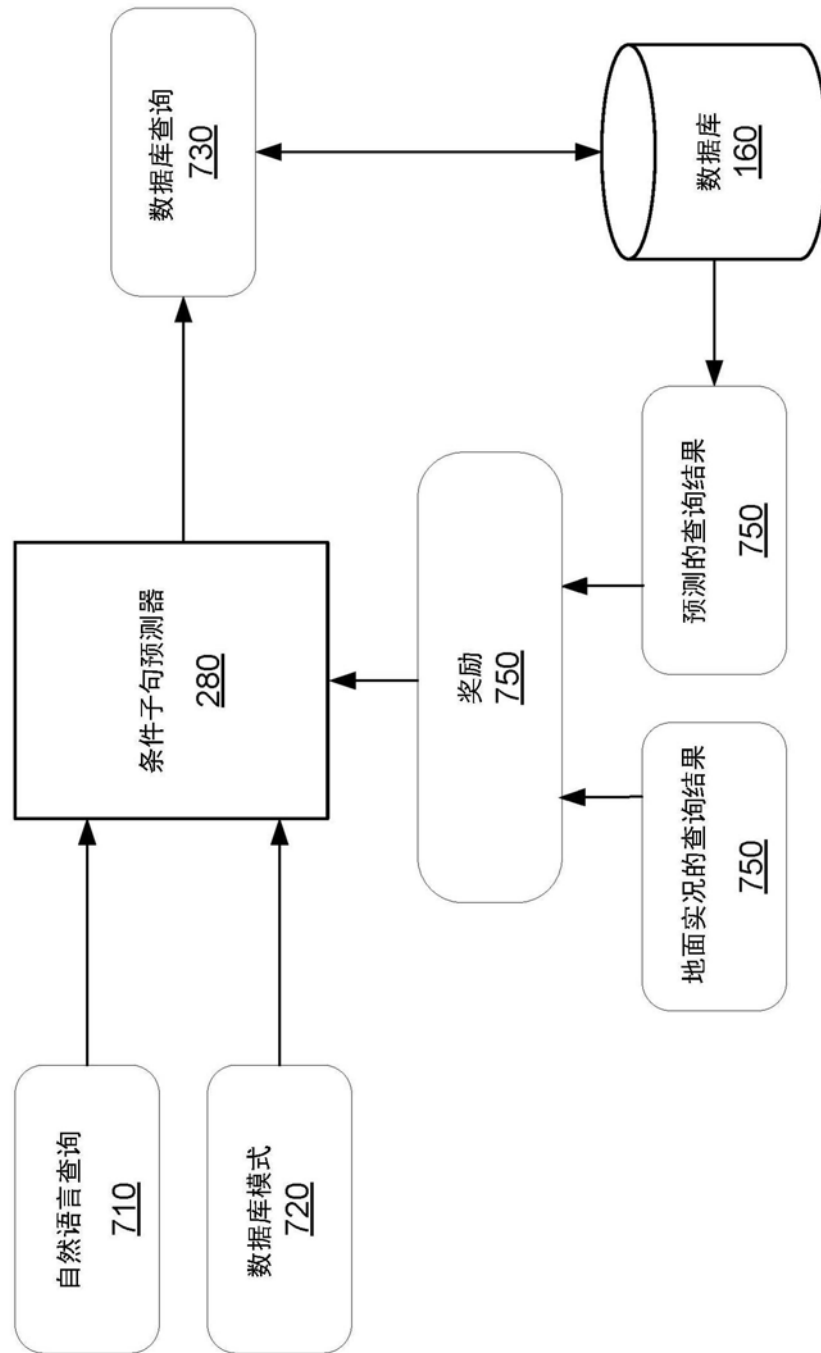


图7

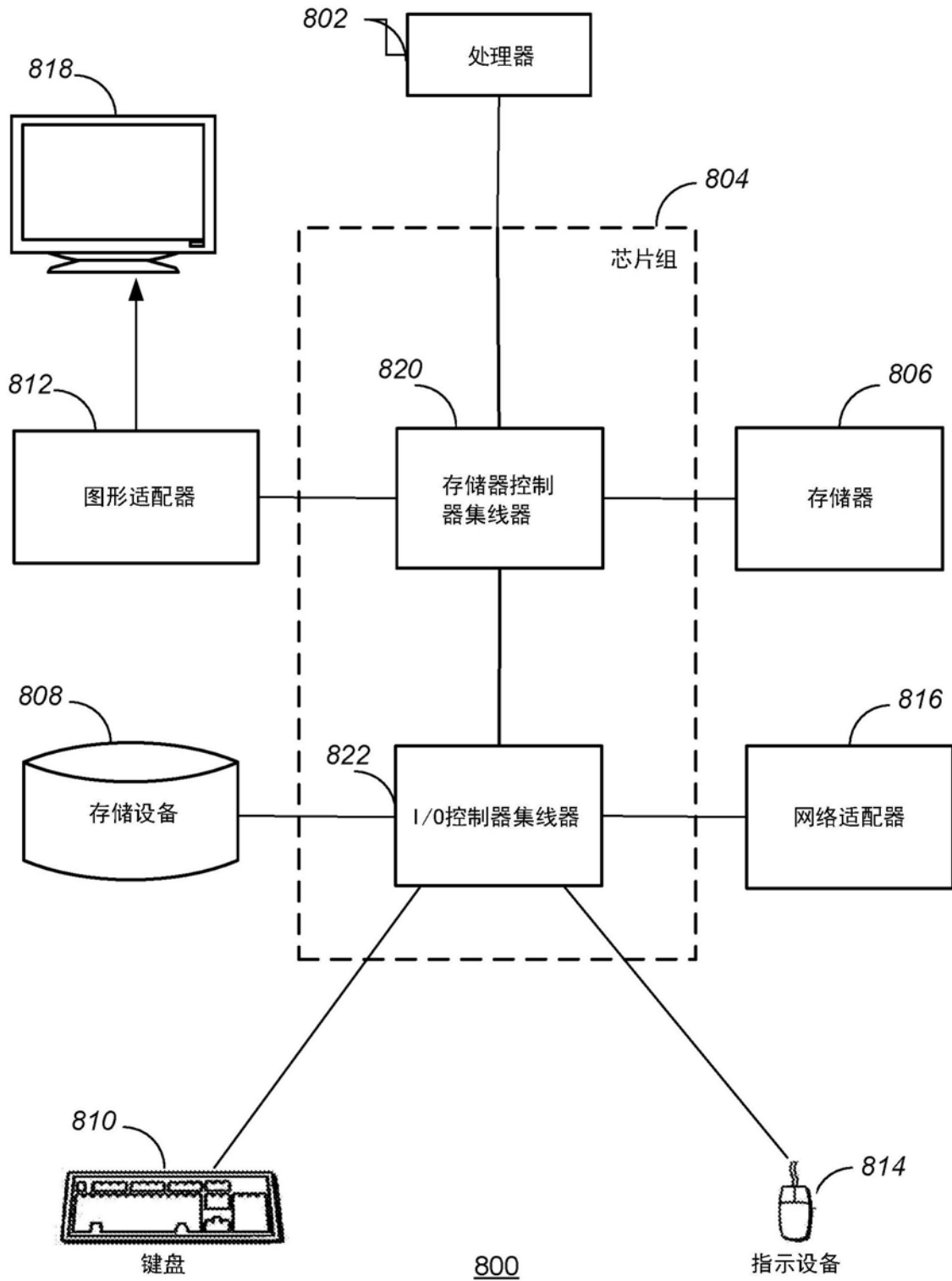


图8