软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, 2022,33(3):797−813 [doi: 10.13328/j.cnki.jos.006448]

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

基于树型门控循环单元的基数和代价估计器*

乔少杰[1], 杨国平[1], 韩 楠[2], 屈露露[1], 陈 浩[3], 毛 睿[4], 元昌安[5], Louis Alberto GUTIERREZ[6]

[1](成都信息工程大学 软件工程学院, 四川 成都 610225)

[2](成都信息工程大学 管理学院, 四川 成都 610225)

[3](北京华为数字技术有限公司, 北京 100085)

[4](深圳大学 计算机与软件学院, 广东 深圳 518060)

[5](广西教育学院, 广西 南宁 530023)

[6](Department of Computer Science, Rensselaer Polytechnic Institute, New York, USA)

通信作者: 韩楠, E-mail: hannan@cuit.edu.cn

摘 要: 基数估计和代价估计可以引导执行计划的选择, 估计准确性对查询优化器至关重要. 然而, 传统数据库的代价和基数估计技术无法提供准确的估计, 因为现有技术没有考虑多个表之间的相关性. 将人工智能技术应用于数据库(artificial intelligence for databases, AI4DB)近期得到广泛关注, 研究结果表明, 基于学习的估计方法优于传统方法. 然而, 现有基于学习的方法仍然存在不足: 首先, 大部分的方法只能估计基数, 但忽略了代价估计; 其次, 这些方法只能处理一些简单的查询语句, 对于多表查询、嵌套查询等复杂查询则无能为力; 同时, 对字符串类型的值也很难处理. 为了解决上述问题, 提出了一种基于树型门控循环单元, Tree-GRU (tree-gated recurrent unit)的基数和代价估计方法, 可以同时对基数和代价进行估计. 此外, 采用了有效的特征提取和编码技术, 在特征提取中兼顾查询和执行计划, 将特征嵌入到Tree-GRU中. 对于字符串类型的值, 使用神经网络自动提取子串与整串的关系, 并进行字符串嵌入, 从而使具有稀疏性的字符串变得容易被估计器处理. 在JOB、Synthetic等数据集上进行了大量实验, 实验结果表明, 所提模型的各方面性能优于主流算法.

关键词: AI4DB; 基数估计; 代价估计; 查询优化器; Tree-GRU; 执行计划

中图法分类号: TP311

中文引用格式: 乔少杰, 杨国平, 韩楠, 屈露露, 陈浩, 毛睿, 元昌安, Gutierrez LA. 基于树型门控循环单元的基数和代价估计器. 软件学报, 2022, 33(3): 797–813. http://www.jos.org.cn/1000-9825/6448.htm

英文引用格式: Qiao SJ, Yang GP, Han N, Qu LL, Chen H, Mao R, Yuan CA, Gutierrez LA. Cardinality and Cost Estimator Based on Tree Gated Recurrent Unit. Ruan Jian Xue Bao/Journal of Software, 2022, 33(3): 797–813 (in Chinese). http://www.jos.org.cn/1000-9825/6448.htm

Cardinality and Cost Estimator Based on Tree Gated Recurrent Unit

QIAO Shao-Jie[1], YANG Guo-Ping[1], HAN Nan[2], QU Lu-Lu[1], CHEN Hao[3], MAO Rui[4], YUAN Chang-An[5], Louis Alberto GUTIERREZ[6]

[1](School of Software Engineering, Chengdu University of Information Technology, Chengdu 610225, China)

* 基金项目: 国家自然科学基金(61772091, 61802035, 61962006, 61962038, U1802271, U2001212, 62072311); CCF-华为数据库创新研究计划(CCF-HuaweiDBIR2020004A); 四川省科技计划(2021JDJQ0021, 2020YJ0481, 2020YJ0430); 成都市重大科技创新项目(2021-YF08-00156-GX); 成都市技术创新研发项目(2021-YF05-00491-SN); 四川音乐学院数字媒体艺术四川省重点实验室资助项目(21DMAKL02); 广西自然科学基金(2018GXNSFDA138005); 广东省基础与应用基础研究基金(2020B1515120028)

本文由“数据库系统新型技术”专题特约编辑李国良教授、于戈教授、杨俊教授和范举教授推荐.

收稿时间: 2021-06-30; 修改时间: 2021-07-31; 采用时间: 2021-09-13; jos在线出版时间: 2021-10-21

[2](School of Management, Chengdu University of Information Technology, Chengdu 610225, China)
[3](Beijing Huawei Digital Technologies Co., Ltd, Beijing 100085, China)
[4](College of Computer Science and Software Engineering, Shenzhen 518060, China)
[5](Guangxi College of Education, Nanning 530023, China)
[6](Department of Computer Science, Rensselaer Polytechnic Institute, New York, USA)

Abstract: Cardinality estimation and cost estimation can guide the selection of execution plan, and cardinality accuracy is very important for query optimizer. However, the cost and cardinality estimation techniques in traditional databases cannot provide accurate estimations because they do not consider the correlation across multiple tables. Recently, the application of artificial intelligence technology to databases (artificial intelligence for databases, AI4DB) has attracted wide attention, and the results show that the learning-based estimation method is superior to the traditional methods. However, the existing learning-based methods have some drawbacks. Firstly, most methods can only estimate the cardinality, but ignore the cost estimation. Secondly, these methods can only deal with some simple query statements, while do not work for complex queries, such as multi-table query and nested query. At the same time, it is also difficult for them to deal with string type of values. In order to solve the above problems, a novel method of estimating the cardinality and cost based on Tree-GRU (tree-gated recurrent unit), which can estimate the cardinality as well as the cost. In addition, an effective feature extraction and coding technology is applied, both query and execution plan are considered in feature extraction. These features are embedded into Tree-GRU. For the value of string type, the neural network is used to automatically extract the relationship between the substring and the whole string, embedding the string, so that the sparse string can be easily processed by the estimator. Extensive experiments were conducted on the JOB and Synthetic datasets and the results show that the performance of the proposed model outperforms the state-of-the-art algorithms in all aspects.

Key words: AI4DB; cardinality estimation; cost estimation; query optimizer; Tree-GRU; execution plan

大数据时代，一个性能优良、功能强大、高效的数据库系统能给我们带来巨大便利. 当数据量猛增和业务需求变复杂时，如果用户对数据库发起查询操作，SQL 语句一般会涉及多表连接以及嵌套查询等复杂语句和操作. 这时，如果应用开发人员编写的 SQL 语句不佳，那么传统查询优化器可能不会优化得较好. 随着人工智能的发展，将神经网络应用到查询优化器是当前的研究热点，目的在于利用神经网络全方位、多角度学习数据与数据之间、数据与 SQL 之间以及 SQL 与数据库之间的关系，从而使优化器更加精确，最终实现无需要求数据库管理员或应用开发人员编写高性能 SQL 语句. 查询优化器将查询转换为执行计划，并具有最佳的估计性能. 为了能够在不同的执行计划备选方案中进行选择，查询优化器必须对中间结果大小有准确的估计. 然而，传统数据库中的优化器产生的执行计划并不好，因为无法对基数和代价进行准确的估计. 在基数/代价估计中，最大的挑战是获得多表之间的相关性[1,2]. 近期，数据库专家尝试使用机器学习模型去解决上述问题. 目前较为先进的估计模型是 MSCN (multi-set convolutional network)[3]和端到端的代价估计器[4]. MSCN 具有以下局限性：该模型只能估计基数，忽略了对代价的估计；该模型使用平均池化的方法去处理查询语句中的所有结构，但对于复杂的谓词并未考虑，只能处理一些简单的数值型查询；该模型只考虑了查询语句本身，但并未考虑与估计结果息息相关的执行计划. 端到端的代价估计器虽然解决了 MSCN 的局限性，但它本身也带来了一些新的问题. 比如，对于字符串类型的值，需要使用 skip-gram 模型[5]进行词嵌入，但使用这种方法需要提前规定子串的提取规则；对于估计器本身，由于使用了树型结构的长短期记忆细胞(Tree-LSTM[6])，训练的速度非常的慢.

本文研究遇到的主要挑战包括：1) 获取不同表、不同列之间的关系十分困难；2) 现有的技术很少同时兼顾基数/代价估计；3) 字符串类型的值较为稀疏，很难处理；4) 业务需求复杂且不同，SQL 包含复杂谓词，难以处理；5)现有的基于学习的技术的训练虽然速度快，但只局限于处理简单 SQL，有的能够处理复杂 SQL，但训练速度非常慢.

针对上述挑战性问题，亟需提出一种新型的基数与代价估计器. 本文的贡献如下：

1) 对查询所涉及的表/列进行采样，可以使所提模型更好地学习数据分布，并且所提模型可以很好地学习表与表、列与列之间的高维特征关系，进而使获取不同表/列之间的关系不再困难；
2) 所提模型的训练数据包含执行计划以及所对应的真实基数和真实代价，可以同时输出估计的基数

和估计的代价, 所提模型可以很好地学习执行计划与基数和代价之间的高维关系, 进而给出精确的估计;

3) 对于字符串类型, 较为稀疏, 很难处理, 所提模型使用一种词嵌入方法去学习字符串中子串与原串的关系, 可以在一定程度上对未出现过的字符串给出合理的向量表示;

4) 所提模型根据复杂而不同的业务需求, 选择了一种树型结构的神经网络作为核心部分, 它完全符合执行计划树的结构, 任何复杂的查询语句最终都转化成计划树, 所以都能被所提模型处理, 泛化性很强;

5) 所提模型具有很强的泛化性, 可以处理任何结构的查询, 并且考虑到训练速度, 选用了 GRU 而不是 LSTM 作为学习单元, 因为 GRU 只有两个门, 训练速度更快.

本文第 1 节主要介绍目前已有的代表性基数估计和代价估计的技术, 包括传统方法和基于学习的方法, 总结了各种方法的优点以及所存在的局限性. 第 2 节主要对代价和基数估计进行概述, 分别对比了传统数据库中的代价/基数估计和基于学习的代价/基数估计, 并介绍了所提模型的基本工作原理. 第 3 节对所提模型进行详细介绍, 包括数据生成器、特征抽取器以及 Tree-GRU 输出器. 第 4 节介绍了一种高效的字符串嵌入技术. 第 5 节对比不同模型的实验结果并进行分析. 第 6 节总结全文, 并对未来工作进行展望.

1 相关工作

表 1 例举了代表性基数估计和代价估计的相关方法.

表 1 不同基数/代价估计方法

方法	具体分类
传统基数估计方法	直方图法[7] 概率法[8] 采样法[9]
传统代价模型法	因子调整法
基于学习的基数估计法	查询分类法[10] 深度学习法[3] 强化学习法[11] 深度概率模型法[12]
基于学习的代价估计法	深度学习法[4,13]

传统基数估计方法大致分为 3 类, 包括直方图法[7]、概率法[8]和采样法[9]. 直方图法会按照某一列不同值出现数量多少, 以及出现的频率高低来绘制数据的分布情况, 以便能够指导优化器根据数据的分布做出正确的选择. 目前, 主流的关系型数据库, 如 PostgreSQL 等使用该方法. 然而, 此方法没有考虑不同列之间的联系, 从而导致不准确的估计. 概率法的核心思想是: 将数据库中的行映射成位图, 然后统计 0 或 1 的数量, 最终近似地估计基数的值, 但此方法不能处理范围查询. 采样法是指利用数据样本去估计基数, 有人提出了基于索引的采样, 该方法可以解决在多表连接时样本无效的问题, 但是这种方法过于依赖于索引, 如果用户建立的索引不合适或者没有构建索引, 那么对最终结果的影响十分巨大.

传统代价模型中使用的策略是给定一个数据库配置文件, 配置文件中可以配置多种代价因子, 包括 Seq_page_cost、Random_page_cost、CPU_tuple_cost、CPU_index_tuple_cost 以及 CPU_operator_cost. 上述代价因子因为涉及数据页、CPU、索引以及操作类型, 所以与查询的基数密切相关. 然而传统的代价模型需要数据库管理员根据经验手动去调整上述代价因子, 这样消耗了大量的人力及时间.

基于学习的基数估计是指利用机器学习、深度学习甚至强化学习等手段使数据库具有人工智能的特性, 能自动产生准确的基数估计. Malik 等人[10]首次使用机器学习来做基数估计任务, 他们根据连接条件、谓词中的属性等结构将查询分类. 这种方法局限性太大, 因为它并不能处理训练集中不存在的查询结构. Leis 等人[3]使用了一种多集卷积神经网络, 此方法性能优良、准确度高, 但是它的编码方式较为简单, 无法处理树型结构的查询, 并且也不能处理字符串类型的值. Ortiz 等人[11]利用强化学习技术来处理查询状态, 从而进行查

询优化. Yang 等人[12]提出一种深度概率模型去获取多个属性之间的数据分布, 但是它只关注了单表, 并未考虑多表之间的关联. 一种基于索引的连接采样方式(index-based join sampling, IBJS)[14]能够处理多表之间的联系, 因为它通过索引采集每张基表的样本, 但是这种方式过于依赖索引结构, 如果索引不合适甚至没有索引, 那么性能将会降低. 这种方法的另一个限制是可能产生 0-tuple 问题, 即根据合格的样本数来推断基数, 当查询稀疏时, 样本的位图很有可能等于 0, 则样本失效, 因此不能简单地从样本数来进行推断. 而 TGRU 通过其树型结构神经网络强大的高维空间学习能力, 可以从执行计划中学到特定的表/列以及谓词的特征信息, 即使位图为 0, TGRU 也能很好地学习, 并给出合理的估计值.

基于学习的代价估计是指利用神经网络去学习查询或者执行计划与代价之间的高维特征关系. Marcus 等人[13]提出一种基于 RNN 的代价模型, 该模型建立在传统关系数据库中的估计基数和其他统计参数的基础上. 该模型将查询计划编码为树状结构的深层神经网络. 然而, 它不编码谓词, 依赖于关系型数据库估计的基数, 不准确的基数估计会给代价估计带来很大的误差. Sun 等人[4]提出了一种基于树结构 LSTM 的端到端代价估计器, 它学习每个子计划的表示, 并使用另一个估计层同时输出估计基数和代价. 此外, 该方法还通过词嵌入对谓词中的关键字进行编码.

为了能让用户使用 SQL 的门槛降低, Fan 等人[15]提出了一种用户友好的交互式 SQL 查询建议模板. Li 等人[16]针对众包数据库系统, 提出了基于人群的查询优化方案. 当前, 机器学习技术已经开始应用于数据库领域[17], 因为传统的人工优化方法既耗时, 又费力, 最终的效果可能也不好. 基于学习的数据库优化是数据库领域未来研究的方向与重点, 目前已有不少的研究成果, 例如连接顺序选择[18]、节点协调[19,20]以及代价估计器[4]等一系列成果.

2 代价和基数估计

代价估计的作用是估计执行计划的执行代价, 查询优化器使用估计的代价来选择最低代价的执行计划. 基数估计是估计查询(包括子查询)结果中的元组数. 在本节中, 将对代价和基数估计的工作原理进行介绍.

2.1 传统数据库的代价与基数估计

传统的数据库使用数据统计的方法来估计代价和基数. 对于过滤操作, 传统数据库使用直方图来估计基数; 对于连接操作, 以参与连接的表节点的选择性为变量, 通过经验函数估计基数. 在图 1 中, 每个节点顶部的数字是估计基数和真实基数. 可以发现, 传统方法存在较大的误差, 而基于学习的方法可以将每个节点的信息保存到向量中.

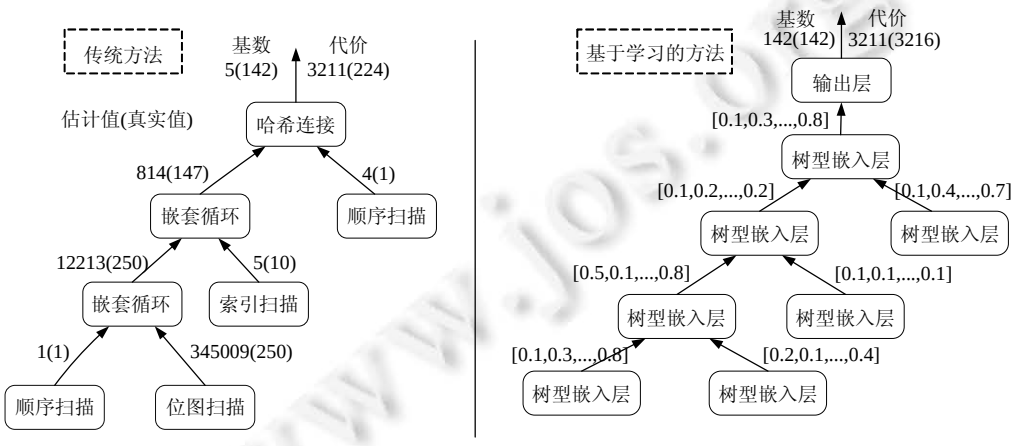


图 1 传统代价估计与基于学习的代价估计方法比较图

一般来说, 利用直方图可以有效地估计叶节点的基数. 但是对于连接操作来说, 由于表之间的相关性, 将

会导致误差非常大. 与传统的代价估计方法不同, 本文提出的基于Tree-GRU的模型能够学习多个列和表之间的相关性, 即便是最复杂的业务所需的查询, 这种表示也能准确地保留执行计划的信息.

此外, 执行计划本身是树形结构, 并且以自底向上的方式执行. 众所周知, 一个计划的代价以及基数应该根据它的子计划来估计. 为此, 本文设计了一个与执行计划吻合的树型结构模型, 其中每个模型可以由子模型组成, 用这些子模型去模拟数据库优化器处理子计划的过程. 本文使用树形结构模型, 以自下而上的方式估计计划的代价以及基数.

2.2 基于学习的代价和基数估计

本文提出的估计器来源于一种端到端的代价估计器[4], 如图2所示. 其整体框架包括3个组件: 数据生成器、特征抽取器以及Tree-GRU输出器.

(1) 数据生成器用于生成训练数据. 首先, 它会根据数据集中的表之间的主外键关系, 生成一张连接图, 如图3所示. 图中的实线表示主外键关系(1:n), 箭头指向主键; 虚线表示外键与外键之间的关系(n:m). 生成器利用连接图和工作负载中的谓词, 产生不同的查询. 对于第 i 个查询 q_i, 生成器都会提取数据库优化器产生的执行计划 p_i, 这个执行计划包含了真实的代价 $cost_i^r$ 以及真实的基数 $card_i^r$. 最后, 数据生成器会生成一个三元组 T_1, 包括该执行计划 p_i、真实的代价 $cost_i^r$ 以及真实的基数 $card_i^r$, 记作 $T_1\langle p_i, cost_i^r, card_i^r\rangle$.

(2) 特征抽取器主要提取执行计划中有用的信息, 比如操作符类型和谓词. 执行计划树的每一个节点都被编码成特征向量, 然后由这些特征向量组成一种树型结构的张量. Tree-GRU把树型结构的张量作为树结构模型的输入. 其中, 简单的特征用0-1向量表示. 对于复杂的特征, 例如节点中的谓词特征, 使用一个三元组 T_2 来表示, 这个三元组由列名 $column_i$、操作符 $operator_i$ 以及操作数 $operand_i$ 组成, 该三元组记作 T_2:$\langle column_i, operator_i, operand_i\rangle$, 最后将该三元组转化为特征向量.

(3) Tree-GRU输出器用于输出最终的代价以及基数. Tree-GRU能够学习执行计划树的各个节点的特征信息, 然后利用这些特征信息去估计代价和基数. 该模型基于训练数据进行训练, 将更新后的参数存储在模型中, 并为新的执行计划估计代价和基数.

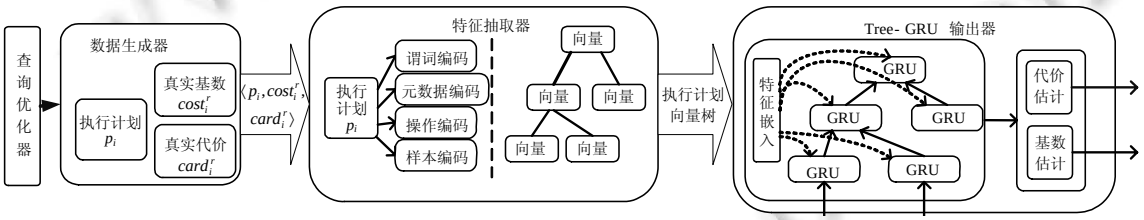


图2 基于树型门控单元的基数/代价估计器结构图

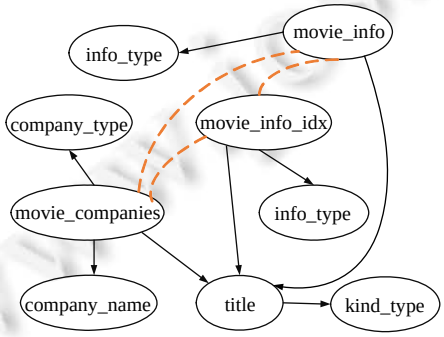


图3 IMDB数据库的表连接关系图

3 基于 Tree-GRU 的代价估计器

本节将介绍基于 Tree-GRU 的代价估计器的具体工作流程及内部结构. 首先介绍了数据生成器和字符串谓词嵌入技术, 然后介绍了特征抽取器, 最后给出 Tree-GRU 输出器.

3.1 数据生成器

如图 3 所示, 以 IMDB 数据库为例, 根据每张表的主键和外键关系生成一个连接图[2], 其中, 节点表示表, 边表示两个表之间的主外键关系. 根据连接图, 选择带有连接关系的表, 在这些连接表中加入谓词. 为了产生谓词, 生成器生成了两种类型的表达式: 一种是数值类型表达式, 另一种是字符串类型表达式. 对于数值类型的表达式, 在连接表中选择数值类型的列, 并在这列随机选取一个数值, 并随机在操作符集合中选择一个操作符插入到列名与该数值之间, 该操作符集合为 $operator_1$={“>”, “<”, “=”, “!=”}. 对于字符串类型的表达式, 选择操作符集合 $operator_2$={“=”, “!=”, “LIKE”, “IN”, “NOT LIKE”, “NOT IN”}, 然后在连接表的字符串类型的列上选择一个值, 与 $operator_2$ 中的操作符做组合. 通过上述步骤, 对每一张连接表都产生了两类表达式, 然后使用“AND/OR”将这两类表达式组合成复杂谓词. 在生成的复杂谓词前加上“SELECT * FROM TABLE_NAME WHERE”, 形成了完整的 SQL 语句, 最后使用 PostgreSQL 数据库将这些语句批量执行”EXPLAIN”命令, 并收集产生的执行计划.

3.2 特征抽取器

首先将查询节点编码为节点向量, 然后将节点向量转换为树状结构向量. 4 种影响基数和代价的因素分别为: 物理查询操作 Operation、查询谓词 Predicate、元数据 Metadata 和样本数据 Sample Bitmap. 接下来讨论如何提取上述特征并将它们编码为向量.

(1) 执行计划的物理查询操作主要包含连接操作、扫描操作、排序操作以及合并操作. 如图 4 所示, 可以清楚地观察到图中各节点的节点类型, 对应着上述 4 种具体的物理查询操作. 需将节点类型用 one-hot 向量表示, 即对物理查询操作使用 one-hot 编码.

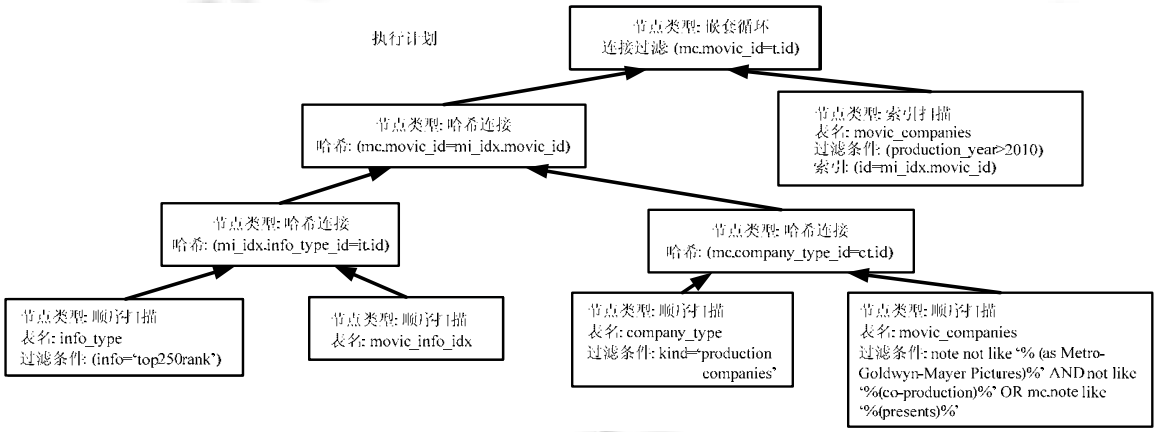


图 4 执行计划树的节点信息

(2) 对于谓词来讲, 复杂谓词通常包含了若干个原子谓词. 原子谓词即“production_year>2010” “t.id=100”等诸如此类的过滤条件. 可以发现, 每一个原子谓词由 3 部分组成: 列名、操作符以及操作数. 对于操作符和列名, 使用 one-hot 向量即可. 对于“AND/OR”, 将其编码为 one-hot 向量. 对于操作数, 如果它是数值类型, 那么使用归一化技术将其映射到范围[0,1]之间; 如果它是字符串类型, 使用 fastText 模型[21]去做字符串嵌入, 具体操作在第 3.3 节介绍. 最后, 将列名、操作符以及操作数所得的向量拼接成一个向量.

对于带有“AND/OR”的复合谓词, 如“(((production_year>2010) AND (production_year<2020)) OR

((production_year>2000) AND (production_year<2005))) AND (season_nr>4)”, 可以将其转换为一棵二叉树, 叶子节点表示原子谓词, 非叶子节点为“AND/OR”, 如图 5 所示. 使用层序遍历(按照各节点右上角的数字标识), 得到一个节点序列, 每个节点都包含一个向量, 再将这些向量拼接成一个长向量, 而这个长向量就包含了上述谓词的所有信息. 当然, 也可以使用其他遍历方式得出节点序列.

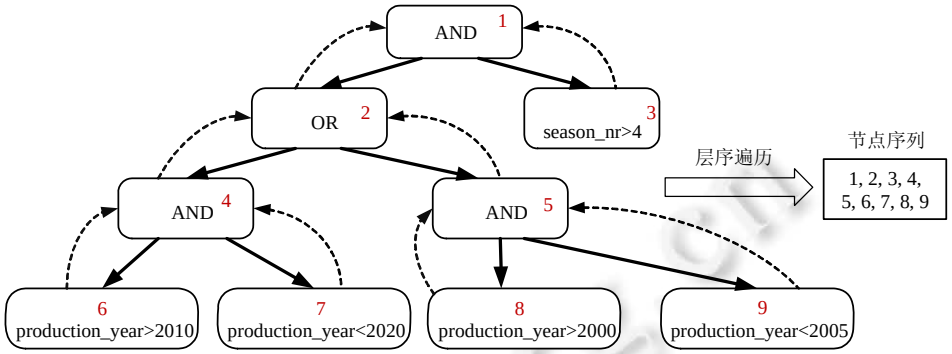


图 5 复合谓词树编码

(3) 元数据编码包括对表名、列名以及索引的编码, 我们使用 one-hot 向量对表名、列名以及索引进行编码. 元节点向量是列向量、表向量和索引向量的拼接.

(4) 对于样本数据的编码, 采用位图作为最终的表示方式. 对每一张表都进行采样, 满足查询节点的谓词的样本就记为 1; 反之为 0. 通过采样的方式, 可以一定程度上对表中的数据分布进行描述.

当处理一个查询时, 对于子计划, 先将其映射为表示向量, 再将表示向量暂时存储起来. 举个例子, 当优化器使用动态规划算法优化 $A⋈B⋈C⋈D$ 时(其中, A, B, C, D 表示 4 张表, ⋈表示连接), 优化器必须知道{A, B, C, D, $A⋈B$, $B⋈C$, $A⋈C$, $(A⋈B)⋈C$, $A⋈(B⋈C)$, ...}的代价. 当估计$(A⋈B)⋈C$ 时, 子计划$(A⋈B)$和 C 的表示向量可以直接从内存中取出, 不需要重新计算, 这样可以节省计算资源. 对于子计划的存储, 只存储当前查询的子计划, 当前查询执行完毕后, 释放子计划的存储空间.

3.3 Tree-GRU输出器

Tree-GRU 输出器由 3 层组成: 嵌入层, 表示层和估计层. 首先, 每个计划的节点特征向量是大而稀疏的, 需要对其进行压缩, 提取出高维的特征信息, 进而, 嵌入层对每个节点嵌入向量; 其次, 表示层采用树状结构模型, 每个节点都是一个表示模型, 树状结构与计划相吻合. 每个表示模型学习对应的子计划的向量信息. 每个表示模型的节点学习其两个子节点的向量. 最后, 根节点向量传入估计层对代价和基数进行估计.

(1) 嵌入层将稀疏向量压缩为稠密向量. 在第 3.2 节中得到了物理查询操作 Operation、元数据 Metadata 以及样本数据 Sample Bitmap 的 one-hot 向量, 在嵌入层中, 需要对以上 one-hot 向量进行压缩. 本文使用了多级卷积神经网络, 用不同参数的多层感知机(multilayer perceptron, MLP)[22]压缩 one-hot 向量. 所用多层感知机包含 4 层, 包含两层全连接层, 激活函数都使用 ReLU[23]. 而对于谓词 Predicate 向量, 由于它太复杂, 不能简单地用多层感知机去处理.

对于原子谓词, 可以直接使用这个向量, 因为其比较简单. 包含“AND/OR”的复合谓词所对应的向量较长, 而且很稀疏, 使用多层感知机去处理的效果较差. 考虑一个带有两个原子谓词的复合谓词, 假如使用“AND”语义, 可以用满足原子谓词的估计结果的最小数目来估计满足谓词的结果数目. 因此, 使用最小池化层来合并“AND”语义的两个原子谓词. 进一步地, 考虑一个带有两个原子的复合谓词, 假如使用“OR”语义, 可以用满足原子谓词的估计结果的最大数目来估计满足谓词的结果数目. 因此, 本文使用最大池化层来组合“OR”语义的两个原子谓词. 图 6 显示了一个复合谓词及其嵌入模型. 对于叶节点, 使用全连接神经网络. 对于非叶节点, 使用最大池化层表示“OR”, 最小池化层表示“AND”, 满足“AND”和“OR”的语义. 使用最小/最大

池化不是唯一的处理节点序列的方案，也可以使用 LSTM[24]等能够处理序列的模型.

将节点 t 的物理操作 Operation、元数据 Metadata、谓词 Predicate 和样本数据 Sample Bitmap 分别表示为 Ope_t, Met_t, Pre_t, Bit_t, 将特征谓词的每个节点表示为 Pre_t, 其中, Pre_t^l 为其左子节点, Pre_t^r 为其右子节点. 嵌入的公式如下所示, 其中, E 是最终的嵌入向量, W_o, W_m, W_b 分别是 Operation, Metadata 以及 Sample Bitmap 所对应的 MLP 权重, W_p^1 表示谓词全连接神经网络的权重, W_p^2 表示 Predicate 所对应的 MLP 权重, b 是偏移项. 对于谓词嵌入, 需要使用递归的方法从根节点开始进行嵌入, 具体参见公式(1)–公式(6):

$$E=[embed(Ope_t),embed(Met_t),embed(Pre_t),embed(Bit_t)] \tag{1}$$

$$embed(Ope_t)=MLP(W_o\cdot Ope_t+b_o) \tag{2}$$

$$embed(Met_t)=MLP(W_m\cdot Met_t+b_m) \tag{3}$$

$$embed(Bit_t)=MLP(W_b\cdot Bit_t+b_b) \tag{4}$$

$$embed'(Pre_t)=\begin{cases}\min(embed'(Pre_t^l),embed'(Pre_t^r)), & Pre_t=\text{AND}\\ \max(embed'(Pre_t^l),embed'(Pre_t^r)), & Pre_t=\text{OR}\\ W_p^1\cdot P_t+b_p^1, & Pre_t=\text{EXPR}\end{cases} \tag{5}$$

$$embed(Pre_t)=MLP(W_p^2\cdot embed'(Pre_t)+b_p^2) \tag{6}$$

其中, $type(Pre_t)$表示节点的类型, 包括“AND”、“OR”和谓词表达式.

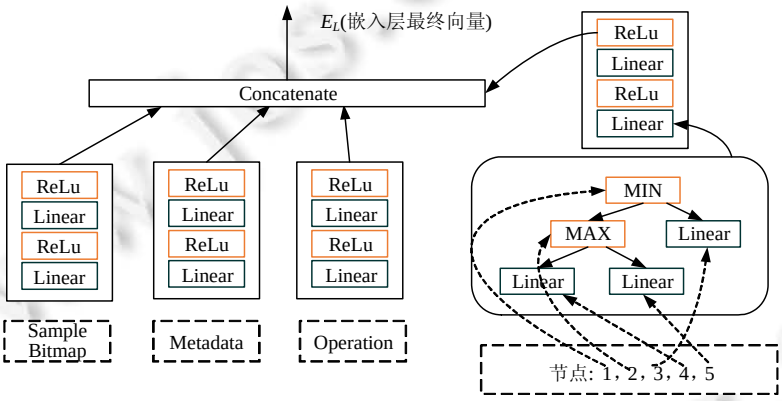


图 6 嵌入层编码

(2) 表示层主要解决信息消失和空间爆炸的问题. 通常很容易估计简单操作的代价, 例如估计单个表上过滤谓词的代价, 但很难估计连接多个表的成本, 因为连接空间很大, 连接的元组是稀疏的. 对于计划中的叶子节点, 是容易处理的; 但对于非叶子节点, 尤其是更高层的节点, 很难获取低层节点传入的信息. 其次, 想要获得足够的信息量, 需要从大量的中间结果中去寻找, 进而花费大量的存储空间. 表示层旨在解决这两个问题: 通过捕获从叶子节点到根节点的全局代价信息以及避免信息丢失.

如图 7 所示, 每个表示层包含了一个门控单元 GRU. 这一层的所有单元都是具有相同结构和共同参数的神经网络, 将这些单元称为表示模型. 每个表示模型有 3 个输入: 嵌入向量 E、左孩子向量的表示向量 H_{t-1}^l、右孩子向量的表示向量 H_{t-1}^r. 对于叶节点, 使用零向量作为其子向量. 每个单元输出为 H_t. 与深度神经网络相比, GRU 细胞可以利用额外的信息通道有效地解决梯度消失以及底层节点信息丢失问题.

与 Tree-LSTM 类似, Tree-GRU 模型是对顺序 GRU 模型的扩展[25]. Tree-GRU 和 Tree-LSTM 之间的区别是它们如何调节单元内部的信息流. 具体来说, Tree-GRU 单元去掉了单独的存储单元, 仅使用两个门来模拟信息收集中的重置和更新过程. 每个 Tree-GRU 单元的计算如公式(7)–公式(10)所示:

$$r_t=\sigma(W^{(r)}\cdot E_t+U^{(r)}\cdot[H_{t-1}^l,H_{t-1}^r]) \tag{7}$$

$$z_t = \sigma(W^{(z)} \cdot E_t + U^{(z)} \cdot [H_{t-1}^l, H_{t-1}^r]) \tag{8}$$

$$H_t^* = \tanh(W^{(h)} \cdot E_t + U^{(h)} \cdot [H_{t-1}^l \odot r_t, H_{t-1}^r \odot r_t]) \tag{9}$$

$$H_t = z_t \odot H_t^* + (1 - z_j) \odot (H_{t-1}^l + H_{t-1}^r) \tag{10}$$

其中, r_t 是重置门, z_t 是更新门. 重置门控制前一状态有多少信息被写入到当前的候选集 H_t^* 上, 重置门越小, 前一状态的信息被写入的越少. 而更新门决定更新到隐藏状态的信息. 在 Tree-GRU 中没有记忆单元, 只有 H_{t-1}^l 和 H_{t-1}^r 作为左、右子节点的隐藏状态.

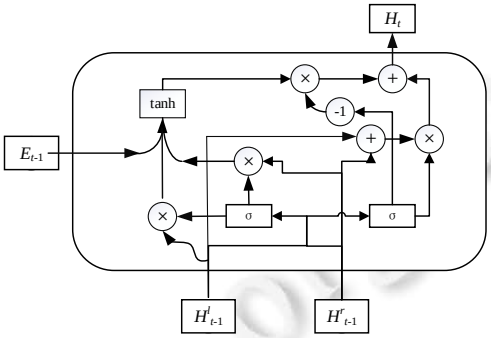


图 7 树型结构 GRU 单元结构

(3) 估计层包含两层全连接层, 激活函数使用 ReLU 和 Sigmoid 函数, 如图 8 所示, 使用 Sigmoid 函数预测归一化后的基数和代价. 输出层能够通过其表示向量来估计任何子计划的代价或基数. 归一化基数和代价的公式如下:

$$card_{out}=(card_i-card_{\min})/(card_{\max}-card_{\min}) \tag{11}$$

$$cost_{out}=(cost_i-cost_{\min})/(cost_{\max}-cost_{\min}) \tag{12}$$

其中, $card_{out}$ 与 $cost_{out}$ 分别表示输出的基数和输出的代价归一化后的结果. $card_i$ 表示第 i 个查询的基数, $card_{\min}$ 表示训练集中最小的基数, $card_{\max}$ 表示训练集中最大的基数. 同理, $cost_i$、$cost_{\min}$ 与 $cost_{\max}$ 分别表示第 i 个查询的代价、训练集中最小和最大的代价值. 通过上述公式, 将数值较大的基数和代价映射到 0–1 之间, 然后再输入到模型中. 最后的 Sigmoid 输出值是一个 0–1 的概率值, 再次借助上述公式, 将这个概率值还原为基数和代价值. 对于估计层的最后一层不继续使用 ReLU, 是因为 ReLU 函数在负半区的值恒为 0, 而正半区的取值范围(0,+∞), 取值范围无法与公式(11)、公式(12)对应; 而 Sigmoid 函数取值范围在[0,1], 恰好能与公式(11)、公式(12)的取值范围对应, 故而选用 Sigmoid 函数.

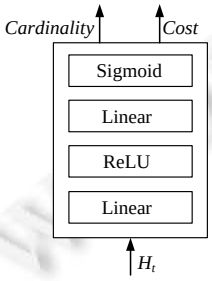


图 8 估计层结构

4 字符串类型的谓词嵌入

上一节详细描述了所提模型的数据生成方式、特征抽取方式以及模型内部结构, 但是对于谓词的处理, 依

然很困难, 因为谓词的形式多种多样, 特别是字符串类型的谓词. 首先, 数值的分布是可以学习的, 即使有些值没有出现在训练数据中, 因为大多数数值都在一个连续的空间中. 例如, 谓词 production_year∈[1990,2000)可以从 production_year∈[1990,1995)和 production_year∈[1995,2000)中推断出来. 然而, 字符串值是稀疏和离散的, 因此很难学习其分布. 比如, 谓词“NOT LIKE ‘%(user_comment)%’”, 如果不知道哪些值包含模式‘(user_comment)’, 则很难支持带有字符串值的谓词. 模糊查询所包含字段的值可能含有多个子词, 如何学习子词与其他完整词之间的关系是一个挑战. 本节使用一种基于学习的字符串嵌入方法去学习字符串中的模式, 进而使所提模型能够很好地支持字符串类型的谓词.

4.1 字符串嵌入

给定一个数据集和一个查询工作负载, 目标是对当前工作负载中使用的或将在未来工作负载中使用的所有关键字进行编码. 因此, 不仅需要对查询工作负载中出现的普通字符串进行编码, 而且需要对字符串/子字符串进行泛化, 并生成一些重要的规则来提取所有关键字在数据集的查询工作负载中. Sun 等人[4]提出了一种基于规则的字符串子串提取, 用满足规则的子串去训练 skip-gram 模型来学习每个子串的表示. 这种提取子串的方式需要人工提取规则, 而本文使用 fastText 模型去自动提取子串和整词之间的关系, 速度非常快, 将在第 4.2 节详细介绍.

4.2 fastText训练字符串

为了更好地对字符串进行嵌入, 本文使用 fastText 进行嵌入. fastText 的子词嵌入在 word2vec 的基础上, 引入了子词这个因素, 从而使得词的微变形关系也能映射到嵌入空间中. 在 fastText 中, 每个词被看作是 *n*-gram 字母串包. 为了区分前后缀情况, “〈”与“〉”符号被加到了词的前后端. 除了词的子串外, 词本身也被包含进了 *n*-gram 字母串包. 以 where 为例, 在 *n*=3 的情况下, 其子串分别为(〈wh)、(whe)、(her)、(ere)、(re〉)以及其本身(〈where〉). 注意, 这里的 her 与单词〈her〉是不同的.

本文使用负例采样的方式加速 fastText 的训练速度. 对比 skip-gram 与 fastText 经过负采样后的损失函数, 其中, L_1(skip-gram)表示某一中心词的损失函数, L_2(fastText)表示某一中心词及其子词的损失函数. 其中, (·)表示 Sigmoid 激活函数, u_o^T 表示中心词 v_c 的周围词向量的转置, u_i^T 表示第 i 个中心词 v_c 的非周围词向量的转置, C_w 表示中心词 v_c 的子词并集, z_g 表示某一子词向量:

$$L_1=\log\sigma(u_o^T\cdot v_c)+\sum_{i=1}^{i}\log\sigma(-u_i^T\cdot v_c) \tag{13}$$

$$L_2=\log\sigma(u_o^T\cdot\sum_{g\in C_w}z_g)+\sum_{i=1}^{i}\log\sigma(-u_i^T\cdot\sum_{g\in C_w}z_g) \tag{14}$$

通过公式(13)、公式(14)可以发现, skip-gram 并不能学习子词之间的关系, 而 fastText 可以学习子词之间的关系. 在 fastText 中, 原中心词向量被替换成了中心词子向量之和, 与整词学习(skip-gram、CBOW[5]、Glove[26])不同, 词典以外新词的词向量可以使用新词中的子词向量之和. fastText 对于一些语言十分重要, 例如阿拉伯语、德语和俄语等. 比如在德语中有很多复合词, 例如乒乓球(英文为 table tennis)在德语中叫 Tischtennis. 可以通过 fastText 的子词学习进而表示两个词的相关性, 例如 tennis 和 Tischtennis.

5 实验结果与分析

实验中使用真实的数据集 IMDB 和 JOB[2]. 互联网电影资料库(Internet movie database, IMDB)是一个关于电影演员、电影、电视节目、电视明星、电子游戏和电影制作的在线数据库, 该数据集收集了超过 250 万部电影, 这些电影由 234 997 家不同的公司生产, 演员超过 400 万. 由于 IMDB 数据集的相关性和倾斜分布, 估计 IMDB 数据集的基数和代价要比 TPC-H[27] (TPC-H 是一个决策支持基准, 它由一套面向点对点查询和并发数据修改的业务组成, 选择的查询和数据具有广泛的行业相关性)困难得多. IMDB 数据集包括 22 个表, 表通过主键和外键连接, 在主键和外键上建立索引. JOB 数据集基于真实 IMDB 数据集生成, 包含 113 个多表连接

查询, JOB 数据集具有多样性和真实性的特点. 实验中使用第 5.1 节介绍的两种类型的查询工作负载进行测试.

5.1 实验设置

第 1 种工作负载(workload)类型仅包含具有数值的谓词. 使用了两个被广泛使用的工作负载: Synthetic 工作负载是基于 IMDB 数据集生成的 SQL 语句, JOB-light 工作负载是基于 IMDB 数据集生成的 SQL 语句. Synthetic 包含 5 000 个查询, 每个查询含有 0–2 个连接. JOB-light 包含 70 个查询, 每个查询含有 1–4 个连接. 利用本文设计的数据生成器, 基于 IMDB 数据集生成了 100 000 个查询, 其中, 90 000 个查询作为训练集, 10 000 个查询作为验证集, 上述两个工作负载作为测试集.

第 2 种工作负载类型包含带有字符串属性的复杂谓词. 基于 IMDB 的主外键连接关系图和 JOB 工作负载中使用的谓词来生成训练数据. 生成 30 000 个单表查询, 30 000 个 0–4 个连接的查询, 30 000 个超过 5 个连接的查询. 从 PostgreSQL 的优化器中获取这些查询的执行计划, 并使用它们来训练模型. 将生成的查询的 90% 作为训练集, 10%作为验证集, JOB 工作负载作为测试集.

实验中选用最流行的关系数据库进行对比实验, 即 PostgreSQL v9.7.21、MySQL v5.7.19、SQL Server 2014 和 Oracle 12c. 本文也实现一些主流的模型作为比较方法: (1) IBJS[14]是基于索引的基数估计, 其查询性能依赖于索引结构; (2) MSCN 使用多集卷积神经网络来学习 SQL 查询的基数, 但它不支持字符串谓词, 而且不支持代价估计; (3) Tree-LSTM 估计器需要人为规定子词或子串的规则, 在业务多变的大数据场景下, 基于规则的提取不适用了, 用神经网络自动学习规则是一种可取的方法, 但是 Tree-LSTM 有 3 个门, 训练速度相对较慢. 下述实验的 TLSTM 表示使用基于规则的字串提取方法的 Tree-LSTM 模型, 并且使用 LSTM 处理复合谓词节点序列. 实验使用 PyTorch[28]框架和 CUDA[29]对提出的模型进行编码. 实验硬件环境为: Intel(R) Xeon(R) CPU i7-6700k, 16 GB 内存, 128 GB SSD, Ubuntu 18.04 操作系统的机器上进行.

5.2 评价指标

实验中模型同时训练代价和基数, 损失函数可以是代价损失和基数损失的线性组合, 权重可以视为超参数. 实验中, 在{0.001,0.01,0.1,0.2,0.5,1.0,5.0,10.0}中尝试不同的损失权重, 并通过交叉验证选择验证误差最小的损失权重. 为了达到高质量的估计效果和加快收敛速度, 本文以归一化的真实基数/代价为目标. 损失函数形式化定义如公式(15)所示, 公式(16)、公式(17)分别表示代价和基数 *q-error*(误差).

$$L=\frac{1}{n}\sum_{i=1}^{n}(\omega\cdot qerror(cost_r-cost_e)+qerror(card_r-card_e)) \tag{15}$$

$$q\text{-}error(cost_e,cost_r)=\max(cost_e,cost_r)/\min(cost_e,cost_r) \tag{16}$$

$$q\text{-}error(card_e,card_r)=\max(card_e,card_r)/\min(card_e,card_r) \tag{17}$$

其中, L 表示基数和代价相组合的损失函数, n 为训练查询数, $cost_r$ 和 $card_r$ 分别是实际代价和基数, $cost_e$ 和 $card_e$ 分别表示估计的代价和基数. 图 9 展示了训练时各个模型的损失值下降情况.

图 9 展示了 3 种基于学习的基数/代价估计器在验证集上技术损失和代价损失变化情况. 特别地, 在基数估计中, 由于各模型经历第 1 次或前两次 epochs 训练所得到损失较大, 不利于在图中表示, 因此对基数的损失取 $\log_2(\cdot)$函数做处理, 使其映射到更小的范围里, 便于观察. 图 9(a)中的 log(TGRU)表示对 TGRU 的基数估计损失做 log 操作, 其他两种模型同理表示.

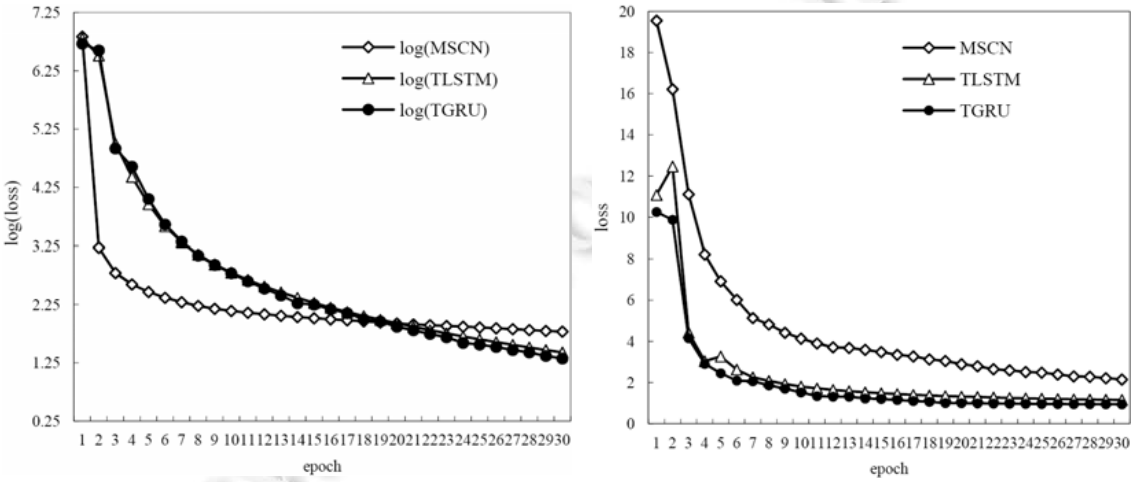
图 9(a)展示了模型在验证集的基数损失, 可以发现, MSCN 下降最快, 在第 10 个 epoch 就接近于 2 了, 但此后收敛较慢. 而 TGRU 与 TLSTM 的曲线几乎重合, 损失匀速下降, 虽然在第 15 个 epoch 前下降较慢, 但之后继续下降, 而 MSCN 几乎收敛了. TGRU 从第 29 个 epoch 之后, 损失比 TLSTM 下降得更快. 造成上述情况的原因如下.

1) MSCN 的编码方式比较巧妙, 恰好能用结构简单的多级卷积模型去训练, 故而前期损失下降得快. 但在后期, 在数据集中复杂查询导致 MSCN 的损失无法再下降了.
2) TLSTM 与 TGRU 采用树型结构模型去处理执行计划树, 这是一种新方法, 因为执行计划本身就是

树型结构，前期损失下降慢是因为模型结构较为复杂，需要大量的 epoch 让模型变得更好.

3) 由于对真实的基数损失做了 log 处理，导致图中 TGRU 与 TLSTM 的曲线几乎重合. 实际上，TGRU 要比 TLSTM 稍好，从第 29 个 epoch 开始，就能明显看到 TGRU 曲线的下降趋势比 TLSTM 的大. 这是因为 TLSTM 的每个单元含有 3 个门，而 TGRU 只含有两个门，并且 TGRU 含有最小/最大池化的方案，所以速度与精确度都均得到兼顾.

图 9(b)展示了模型在验证集的代价损失. MSCN 从一开始就不如 TGRU 与 TLSTM 模型，主要原因是其设计的初衷仅仅是为了基数估计，它并未考虑执行计划对代价的影响. TGRU 与 TLSTM 的代价损失最终收敛于 1.0 左右，在第 10 个 epoch 之前, TGRU 的损失下降比 TLSTM 稍快. TGRU 与 TLSTM 充分考虑了执行计划对代价估计的影响，而 TGRU 对 TLSTM 进行了优化.



(a) 数值类型验证集基数损失　　(b) 数值类型验证集代价损失

图 9　数值类型验证集上不同模型损失对比

5.3 数值类型工作负载测试

由于现有方法几乎只支持带有数值类型的谓词，本节实验使用带有数值谓词的工作负载对不同方法进行测试. 即模型收敛后，在 Synthetic 和 JOB-light 工作负载下进行测试. 表 2 和表 3 中，mid，90th，99th，max 和 mean 分别表示不同估计器的中值、第 90 百分位数、第 99 百分位数、最大和平均 *q-error* 值. 加粗数字表示每组实验中最小的误差值.

从表 2 和表 3 可以发现，基于学习的方法(MSCN、TLSTM、TGRU)优于传统的方法(PostgreSQL、MySQL、Oracle). 因为传统的方法依赖于不同列之间的独立假设，而基于学习的方法可以捕获列与表之间的相关性. IBJS 在基数估计方面获得了最好的误差中位数. TLSTM 和 TGRU 在基数/代价估计的表现上优于 MSCN，因为树状结构模型比卷积网络可以更好地捕获查询谓词并学习它们的语义. 而 TGRU 优于 TLSTM，因为 TLSTM 在处理复杂谓词序列中使用的是线性的 LSTM，一旦谓词比较复杂，就不能很好地捕获“AND/OR”等关键字的语义. 本文所提模型使用最小/最大池化树去处理节点序列，自然地结合语法树本身结构去处理复杂谓词. 此外可以观察到，在 Synthetic 上的基数/代价的误差是小于在 JOB-light 上的，因为 Synthetic 中包含的查询比较简单. 但是如果将模型应用于像 JOB-light 这样的带有复杂查询的工作负载上，样本位图的偏差或 0-tuple 问题将导致估计基数的巨大误差. 在 JOB-light 工作负载下，TLSTM 在基数估计的平均误差上比 MSCN 小 50.6%，在代价估计上比 MSCN 少 33.8%. 而 TGRU 在 JOB-light 基数估计的平均误差比 TLSTM 少 6.6%，在代价估计上少 67.6%. 在 Synthetic 工作负载上, TGRU 和 TLSTM 在基数估计的最大误差上比 MSCN 低 4 倍左右，代价估计上低 3 倍左右. 主要原因有以下 3 点.

1) 树结构模型善于表示复杂的计划和谓词.
2) 对于复杂的查询, 树结构模型能够捕获更多的关系.
3) MSCN 使用的代价模型难以调整.

表 2 在数值工作负载中的基数测试误差

方法	Synthetic					JOB-light				
	mid	90th	99th	max	mean	mid	90th	99th	max	mean
PostgreSQL	1.69	9.57	456	373 901	154	7.93	164	2 912	3 477	174
MySQL	2.07	22.6	625	458 835	353	9.55	303	2 256	2 578	149
Oracle	1.97	12.4	473	545 912	378	8.32	374	2 761	3 331	157
SQL Server	2.15	17.4	396	512 591	384	9.03	364	2 550	3 421	151
IBJS	**1.11**	10.36	299	293 290	127	**1.59**	157	14 309	15 775	590
MSCN	1.19	3.32	30.51	1 322	2.95	3.85	78.4	927	1 110	57.9
TLSTM	1.25	3.41	26.12	364	2.92	3.79	53.8	266	298	28.6
TGRU	1.21	**3.30**	**25.8**	**345**	**2.89**	3.55	**50.1**	**256**	**285**	**26.7**

表 3 在数值工作负载中的代价测试误差

方法	Synthetic					JOB-light				
	mid	90th	99th	max	mean	mid	90th	99th	max	mean
PostgreSQL	15.1	65.1	1 200	8 040	62.7	26.8	332	2 740	3 020	173
MySQL	4.51	37.6	451	7 250	35.3	9.47	102	1 293	2 228	84.5
Oracle	6.72	41.4	793	6 672	56.5	12.3	157	1 366	1 825	102.1
SQL Server	9.15	57.4	596	6 691	58.4	10.1	164	1 550	2 421	121
MSCN	3.14	7.43	65.51	738	10.7	4.75	11.4	567	987	28.1
TLSTM	1.59	4.49	59.7	695	4.52	3.77	33.8	445	585	18.6
TGRU	**1.54**	**4.39**	**48.2**	**559**	**4.21**	**1.94**	**12.2**	**104**	**127**	**6.02**

5.4 字符串类型工作负载测试

本节利用 100 000 个使用多表连接的查询训练表示层和输出层. 将 90%的查询作为训练集, 余下的 10%作为验证集. 为了研究本文提出的字符串编码技术, 在包含多表连接和复杂谓词的训练集上训练所提模型, 并在含有 113 个查询的 JOB 工作负载上测试, 从而可以比较不同谓词嵌入技术对复杂查询估计的影响.

可以从表 4 和表 5 得到以下结论: 传统的关系型数据库在 JOB 工作负载下对复杂查询的基数估计存在较大的误差. 原因是使用统计或抽样方法传递到查询计划树的根节点的分布信息不准确, 传统方法将大多数查询的基数估计为 1(真正的值从 0 到百万). 代价估计误差小于基数估计, 且基于学习的方法仍比传统方法优越. TLSTM 基数估计的平均误差要比 PostgreSQL 小 184.6 倍, 代价估计的平均误差要比 Oracle 小 5.8 倍. 而 TGRU 基数估计的平均误差比 TLSTM 减少了 11.1%, 代价估计的平均误差减少了 12.4%. 因为 TGRU 模型既能学习列之间的关系, 也能学习表之间的关系, 并使用 fastText 嵌入技术对字符串子串进行自动学习, 所以 TGRU 模型在多表连接时表现很好. TGRU 模型带有最小/最大池化的树模型, 可以更好地表示复合谓词, 并为基数和代价估计训练一个更健壮的模型. 在代价估计的 99th 和最大误差中, TGRU 的性能比 TLSTM 分别高 1.65 倍和 1.89 倍, 因为最小/最大池化结构训练的谓词表示对于复杂查询较为准确.

表 4 不同方法在 JOB 工作负载中的基数测试误差

方法	JOB					
	mid	90th	95th	99th	max	mean
PostgreSQL	184	8 303	34 204	106 000	670 000	10 416
MySQL	104	28 157	213 471	1 630 689	2 487 611	60 229
Oracle	119	55 446	179 106	697 790	927 648	34 493
SQL Server	152	37 925	195 870	853 657	1 131 894	46 257
TLSTM	11.7	145	236	**682**	934	56.4
TGRU	**10.9**	**87.4**	**201**	687	**827**	**50.1**

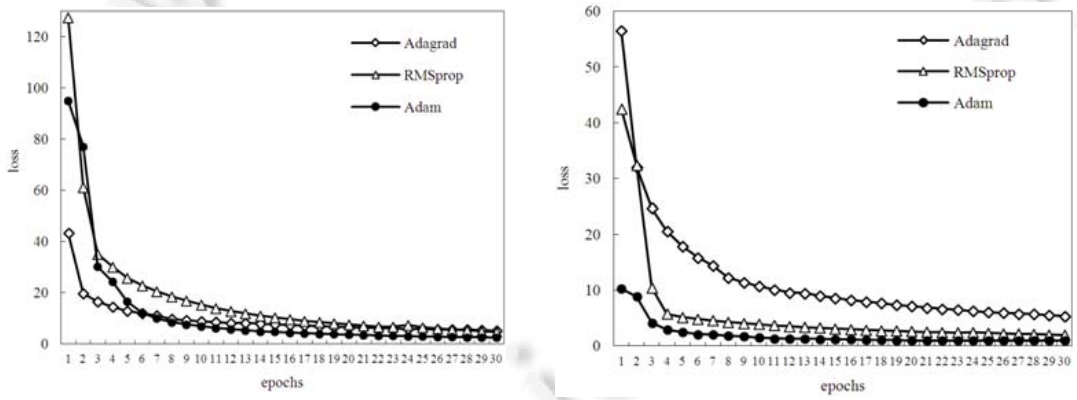
表 5 不同方法在 JOB 工作负载中的代价测试误差对比

方法	JOB					
	mid	90th	95th	99th	max	mean
PostgreSQL	4.90	80.8	104	3 577	4 920	105
MySQL	7.94	691	1 014	15 68	1 943	173
Oracle	6.63	149	246	630	1 274	55.3
SQL Server	6.72	502	627	1 251	1 571	72.3
TLSTM	4.38	14.5	23.7	108	135	9.5
TGRU	**4.21**	**13.7**	**16.7**	**65.8**	**71.2**	**8.32**

5.5 不同梯度下降算法在基数/代价估计时的损失比较

TGRU 模型选择了 Adam[30]算法进行梯度下降, 本节对 Adam 和其他梯度下降算法在数值类型的验证集上分别对基数和代价进行损失测试, 实验结果如图 10 所示.

实验中比较选用了 Adam、RMSprop[31]和 Adagrad[32]. 在基数估计中, Adagrad 在前 6 个 epochs 上的损失下降速度比 Adam 和 RMSprop 快; 但从第 7 个 epochs 开始, Adam 的下降趋势就超过 Adagrad 和 RMSprop 了. 在代价估计中, Adam 从一开始就比 Adagrad 与 RMSprop 下降速度快, 且损失一直保持在较低的水平. 值得注意的是: 在基数估计中, RMSprop 的曲线在 Adagrad 的上方; 而在代价估计中, RMSprop 的曲线在 Adagrad 的下方. 造成上述情况的原因是: TGRU 模型对于基数估计的损失较大, 对于代价估计的损失较小, 而 Adagrad 能够根据损失值的大小自动调节梯度下降速率, 当损失越大时, Adagrad 下降越快, 反之越小. 因为 Adagrad 优化过程中梯度下降速率骤变, 即摆动幅度(所谓的摆动幅度就是在优化中经过更新之后参数的变化范围)较大, 所以 Adagrad 善于处理稀疏梯度. 不过, 随着时间步的增加, 可能会导致学习率降到太小无法进行有效更新. 而 RMSprop 解决了 Adagrad 的速率摆动幅度较大的问题, RMSProp 通过引入一个衰减系数, 让学习率每回合都衰减一定比例, 相当于规定了一个摆动幅度范围, 所以 RMSprop 善于处理非平稳目标. 而 Adam 综合了 Adagrad 和 RMSprop 的优点, 它利用梯度的一阶矩估计和二阶矩估计动态调整每个参数的学习率. Adam 的优点主要在于经过偏置校正后, 每一次迭代学习率都有个确定范围, 使得参数比较平稳, 对内存需求较小, 这也是 Adam 在基数/代价估计的表现优于 Adagrad 和 RMSprop 的原因.



(a) 不同梯度下降算法在基数估计时的性能 (b) 不同梯度下降算法在代价估计时的性能

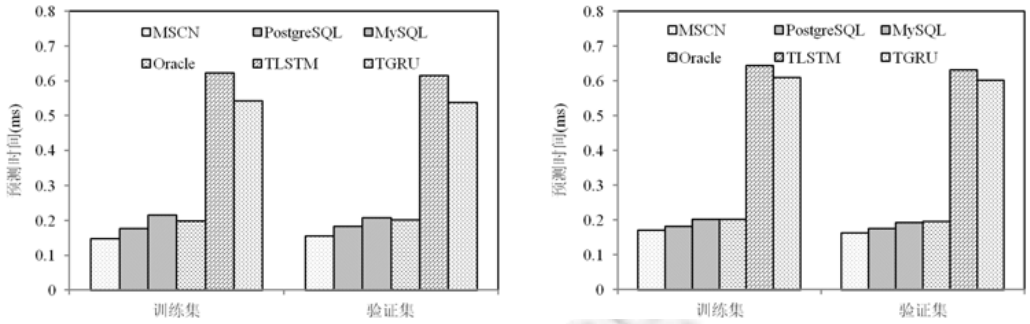
图 10 不同梯度下降算法在基数/代价估计时的损失比较

5.6 时间性能分析

图 11 展示了不同方法的基数和代价估计的平均预测时间.

TGRU 预测基数所需的平均时间是 0.53ms. 在训练集和验证集中, 比较了每个模型的预测时间, 可以发现, TGRU 比 MSCN 和传统数据库需要更多的时间, 但比 TLSTM 需要更少的时间. 训练集上, TLSTM 的预测时间比 TGRU 高 12.7%; 验证集上, TLSTM 的预测时间比 TGRU 高 13.1%. 可以发现, 树结构能够处理复杂的

谓词, 功能强大, 但是所需时间更长. 由于 GRU 本身比 LSTM 的结构简单, 所以 TGRU 的时间比 TLSTM 更少. 对于代价估计来说, 所有模型所消耗的平均估计时间要比基数估计略微高一些, 因为代价估计本身比基数估计复杂. MSCN 的代价估计平均时间最少, 因为它的结构比较简单. 而 TLSTM 与 TGRU 的代价估计平均时间依然较大, 因为树型结构比较复杂.



(a) 基数估计平均时间 (b) 代价估计平均时间

图 11 不同估计器在训练集和验证集上的平均时间比较

6 结论与展望

本文提出了一种新的基于树型门控单元的代价估计器, 用于同时估计基数和代价. 将查询操作、元数据、查询谓词和样本编码到模型中, 然后利用树型结构模型去表示整个执行计划树, 同时又使用了最小/最大池化去处理复合谓词. 为了提高模型的泛化能力, 本文使用 fastText 技术去学习子词或子串之间的关系, 并在实际数据集上进行了大量的实验. 实验结果表明, 所提出的基数和代价估计模型优于现有的各种方法. 未来工作的重点是充分利用树型结构的优点, 尝试优化树型结构模型, 使其时间效率得到提升.

致谢 感谢所有参与本课题研究和本文工作没有在文中署名的课题组成员及对本项目提供技术支持的专家和学者; 感谢所有评阅本文的匿名评审人及对本文提出的宝贵修改意见.

References:

[1] Leis V, Gubichev A, Mirchev A, Boncz PA, Kemper A, Neumann T. How good are query optimizers, really? Proc. of the VLDB Endowment, 2015, 9(3): 204−215. [doi: 10.14778/2850583.2850594]

[2] Leis V, Radke B, Gubichev A, Mirchev A, Boncz PA, Kemper A, Neumann T. Query optimization through the looking glass, and what we found running the join order benchmark. Proc. of the VLDB Endowment, 2018, 27(5): 643−668. [doi: 10.14778/2850583.2850594]

[3] Kipf A, Kipf T, Radke B, Leis V, Boncz PA, Kemper A. Learned cardinalities: Estimating correlated joins with deep learning. In: Proc. of the 9th Biennial Conf. on Innovative Data Systems Research. New York: ACM, 2019. 1−8.

[4] Sun J, Li GL. An end-to-end learning-based cost estimator. Proc. of the VLDB Endowment, 2019, 13(3): 307−319. [doi: 10.14778/3368289.3368296]

[5] Mikolov T, Chen K, Corrado G, Dean J. Efficient estimation of word representations in vector space. In: Proc. of the 1st Int'l Conf. on Learning Representations. New York: ACM, 2013. 1−9.

[6] Dai YF, Guo WZ, Chen X, Zhang ZW. Relation classification via LSTMs based on sequence and tree structure. IEEE Access, 2018, 6: 64927−64937. [doi: 10.1109/ACCESS.2018.2877934]

[7] Ioannidis YE. The history of histograms (abridged). In: Proc. of 29th Int'l Conf. on Very Large Data Bases. Berlin: Morgan Kaufmann, 2003. 19−30. [doi: 10.1016/B978-012722442-8/50011-2]

[8] Flajolet P, Martin GN. Probabilistic counting algorithms for data base applications. Journal of Computer and System Sciences, 1985, 31(2): 182−209. [doi: 10.1016/0022-0000(85)90041-8]

[9] Wu WT, Naughton JF, Singh H. Sampling-based query re-optimization. In: Proc. of the 2016 Int'l Conf. on Management of Data. San Francisco, New York: ACM, 2016. 1721−1736. [doi: 10.1145/2882903.2882914]

[10] Malik T, Burns RC, Chawla NV. A black-box approach to query cardinality estimation. In: Proc. of the 3rd Biennial Conf. on Innovative Data Systems Research. New York: ACM, 2007. 56−67.

[11] Ortiz J, Balazinska M, Gehrke J, Keerthi SS. Learning state representations for query optimization with deep reinforcement learning. In: Proc. of the 2nd Workshop on Data Management for End-to-end Machine Learning. New York: ACM, 2018. [doi: 10.1145/3209889.3209890]

[12] Yang ZH, Liang E, Kamsetty A, Wu CG, Duan Y, Chen X, Abbeel P, Hellerstein JM, Krishnan S, Stoica I. Deep unsupervised cardinality estimation. Proc. of the VLDB Endowment, 2019, 13(3): 279−292. [doi:10.14778/3368289.3368294]

[13] Marcus R, Papaemmanouil O. Plan-structured deep neural network models for query performance prediction. Proc. of the VLDB Endowment, 2019, 12(11): 1733−1746. [doi:10.14778/3342263.3342646]

[14] Leis V, Radke B, Gubichev A, Kemper A, Neumann T. Cardinality estimation done right: Index-based join sampling. In: Proc. of the 8th Biennial Conf. on Innovative Data Systems Research. New York: ACM, 2017. 1−8.

[15] Fan J, Li GL, Zhou LZ. Interactive SQL query suggestion: Making databases user-friendly. In: Proc. of the 27th Int'l Conf. on Data Engineering. Washington: IEEE, 2011. 351−362. [doi: 10.1109/ICDE.2011.5767843]

[16] Li GL, Chai CL, Fan J, Weng XP, Li J, Zheng YD, Li YB, Yu X, Zhang XH, Yuan HT. CDB: Optimizing queries with crowd-based selections and joins. In: Proc. of the 2017 ACM Int'l Conf. on Management of Data. New York: ACM, 2017. 1463−1478. [doi: 10.1145/3035918.3064036]

[17] Li GL, Zhou XH. XuanYuan: An AI-native database systems. Ruan Jian Xue Bao/Journal of Software, 2020, 31(3): 831−844 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5899.htm [doi: 10.13328/j.cnki.jos.005899]

[18] Yu X, Li GL, Chai CL, Tang N. Reinforcement learning with Tree-LSTM for join order selection. In: Proc. of the 36th IEEE Int'l Conf. on Data Engineering. Washington: IEEE, 2020. 1297−1308. [doi: 10.1109/ICDE48307.2020.00116]

[19] Zhang J, Liu Y, Zhou K, Li GL. An end-to-end automatic cloud database tuning system using deep reinforcement learning. In: Proc. of the 2019 Int'l Conf. on Management of Data. New York: ACM, 2019. 415−432. [doi: 10.1145/3299869.3300085]

[20] Li GL, Zhou XH, Li SF, Gao B. QTune: A query-aware database tuning system with deep reinforcement learning. Proc. of the VLDB Endowment, 2019, 12(12): 2118−2130. [doi: 10.14778/3352063.3352129]

[21] Joulin A, Grave E, Bojanowski P, Mikolov T. Bag of tricks for efficient text classification. In: Proc. of the 15th Conf. of the European Chapter of the Association for Computational Linguistics. Valencia: Association for Computational Linguistics, 2017. 427−431. [doi: 10.18653/v1/e17-2068]

[22] Wolf A, Barbosa CRH, Costa E. Multiple MLP neural networks applied on the determination of segment limits in ECG signals. In: Proc. of the 7th Int'l Work-Conf. on Artificial and Natural Neural Networks, Vol.2687. Berlin: Springer-Verlag, 2003. 607−614. [doi: 10.1007/3-540-44869-1_77]

[23] Liu B, Liang Y. Optimal function approximation with ReLU neural networks. Neurocomputing, 2021, 435: 216−227. [doi: 10.1016/j.neucom.2021.01.007]

[24] Graves A, Mohamed A, Hinton GE. Speech recognition with deep recurrent neural networks. In: Proc. of 2013 IEEE Int'l Conf. on Acoustics, Speech and Signal Processing. Washington: IEEE, 2013. 6645−6649. [doi: 10.1109/ICASSP.2013.6638947]

[25] Wang YZ, Li SJ, Yang JF, Sun X, Wang HF. Tag-enhanced tree-structured neural networks for implicit discourse relation classification. In: Proc. of the the 8th Int'l Joint Conf. on Natural Language Processing. Asian Federation of Natural Language Processing, 2017. 496−505.

[26] Pennington J, Socher R, Manning CD. Glove: Global vectors for word representation. In: Proc. of the 2014 Conf. on Empirical Methods in Natural Language Processing. 2014. 1532−1543. [doi: 10.3115/v1/d14-1162]

[27] Capuano EA. A TCO analysis of cluster servers based on TPC-H benchmarks. In: Proc. of the 2005 IEEE Int'l Conf. on Cluster Computing. Washington: IEEE, 2005. 1−10. [doi: 10.1109/CLUSTR.2005.347080]

[28] Mudigere D, Naumov M, Spisak J, Chauhan G, Kokhlikyan N, Singh A, Goswami V. Building recommender systems with PyTorch. In: Proc. of the 26th ACM SIGKDD Conf. on Knowledge Discovery and Data Mining. New York: ACM, 2020. 3525−3526. [doi: 10.1145/3394486.3406714]

[29] AlMouhamed MA, Khan AH, Mohammad N. A review of CUDA optimization techniques and tools for structured grid computing. Computing, 2020, 102(4): 977−1003. [doi: 10.1007/s00607-019-00744-1]

[30] Kingma DP, Ba J. Adam: A method for stochastic optimization. In: Proc. of the 3rd Int'l Conf. on Learning Representations. New York: ACM, 2015. 1−15.

[31] Shi NC, Li DW, Hong MY, Sun RY. RMSprop converges with proper hyper-parameter. In: Proc. of the 9th Int'l Conf. on Learning Representations. New York: ACM, 2021. 1−10.

[32] Duchi JC, Hazan E, Singer Y. Adaptive subgradient methods for online learning and stochastic optimization. Journal of Machine Learning Research, 2011, 12: 2121−2159. [doi: 10.1109/TNN.2011.2146788]

附中文参考文献:

[17] 李国良, 周煊赫. 轩辕: AI 原生数据库系统. 软件学报, 2020, 31(3): 831−844. http://www.jos.org.cn/1000-9825/5899.htm [doi: 10.13328/j.cnki.jos.005899]

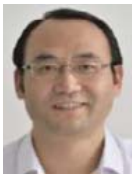
乔少杰(1981－), 男, 博士, 教授, CCF 杰出会员, 主要研究领域为数据库, 人工智能, 数据挖掘.

陈浩(1989－), 男, 学士, 主要研究领域为数据库的优化器与执行器.

杨国平(1997－), 男, 硕士生, CCF 学生会员, 主要研究领域为数据库查询优化.

毛睿(1975－), 男, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为大数据.

韩楠(1984－), 女, 博士, 副教授, 主要研究领域为数据库, 数据挖掘.



元昌安(1964－), 男, 博士, 教授, 博士生导师, CCF 专业会员, 主要研究领域为数据库.

屈露露(1998－), 女, 硕士生, CCF 学生会员, 主要研究领域为人工智能, 数据挖掘.

Louis Alberto GUTIERREZ (1980－), 男, 博士, Researcher, 主要研究领域为人工智能.