PAUL G. ALLEN SCHOOL
OF COMPUTER SCIENCE & ENGINEERING

DeepQuery

Deep Reinforcement Learning for Query Optimization

Jennifer Ortiz *(UW),* Magdalena Balazinska *(UW),* Johannes Gehrke *(Microsoft)*
and *S.* Sathiya Keerthi *(Criteo Reserach)*

Query Optimization is an Old Problem

1979

Access Path Selection
in a Relational Database Management System

P. Griffiths Selinger
M. M. Astrahan
D. D. Chamberlin
R. A. Lorie
T. G. Price

IBM Research Division, San Jose, California 95193

ABSTRACT: In a high level query and data manipulation language such as SQL, requests are stated non-procedurally, without reference to access paths. This paper describes how System R chooses access paths for both simple (single relation) and complex queries (such as joins), given a user specification of desired data as a boolean expression of predicates. System R is an experimental database management system developed to carry out research on the relational model of data. System R was designed and built by members of the IBM San Jose Research Laboratory.

1. Introduction

System R is an experimental database management system based on the relational model of data which has been under development at the IBM San Jose Research Laboratory

access path for each table in the SQL statement. Of the many possible choices, the optimizer chooses the one which minimizes "total access cost" for performing the entire statement.

This paper will address the issues of access path selection for queries. Retrieval for data manipulation (UPDATE, DELETE) is treated similarly. Section 2 will describe the place of the optimizer in the processing of a SQL statement, and section 3 will describe the storage component access paths that are available on a single physically stored table. In section 4 the optimizer cost formulas are introduced for single table queries, and section 5 discusses the joining of two or more tables, and their corresponding costs. Nested queries (queries in predicates) are covered in section 6.

Query Optimization Is Still Hard Today

2015

How Good Are Query Optimizers, Really?

Viktor Leis
TUM
leis@in.tum.de

Andrey Gubichev
TUM
gubichev@in.tum.de

Atanas Mirchev
TUM
mirchev@in.tum.de

Peter Boncz
CWI
p.boncz@cwi.nl

Alfons Kemper
TUM
kemper@in.tum.de

Thomas Neumann
TUM
neumann@in.tum.de

ABSTRACT

Finding a good join order is crucial for query performance. In this paper, we introduce the Join Order Benchmark (JOB) and experimentally revisit the main components in the classic query optimizer architecture using a complex, real-world data set and realistic multi-join queries. We investigate the quality of industrial-strength cardinality estimators and find that all estimators routinely produce large errors. We further show that while estimates are essential for finding a good join order, query performance is unsatisfactory if the query engine relies too heavily on these estimates. Using another set of experiments that measure the impact of the cost model, we find that it has much less influence on query performance than the cardinality estimates. Finally, we investigate plan enumeration techniques comparing exhaustive dynamic programming with heuristic algorithms and find that exhaustive enumeration improves performance despite the sub-optimal cardinality estimates.

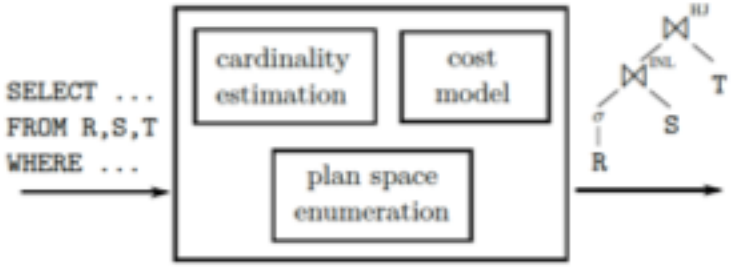


Figure 1: Traditional query optimizer architecture

- How important is an accurate cost model for the overall query optimization process?
- How large does the enumerated plan space need to be?

Two Key Query Optimization Observations

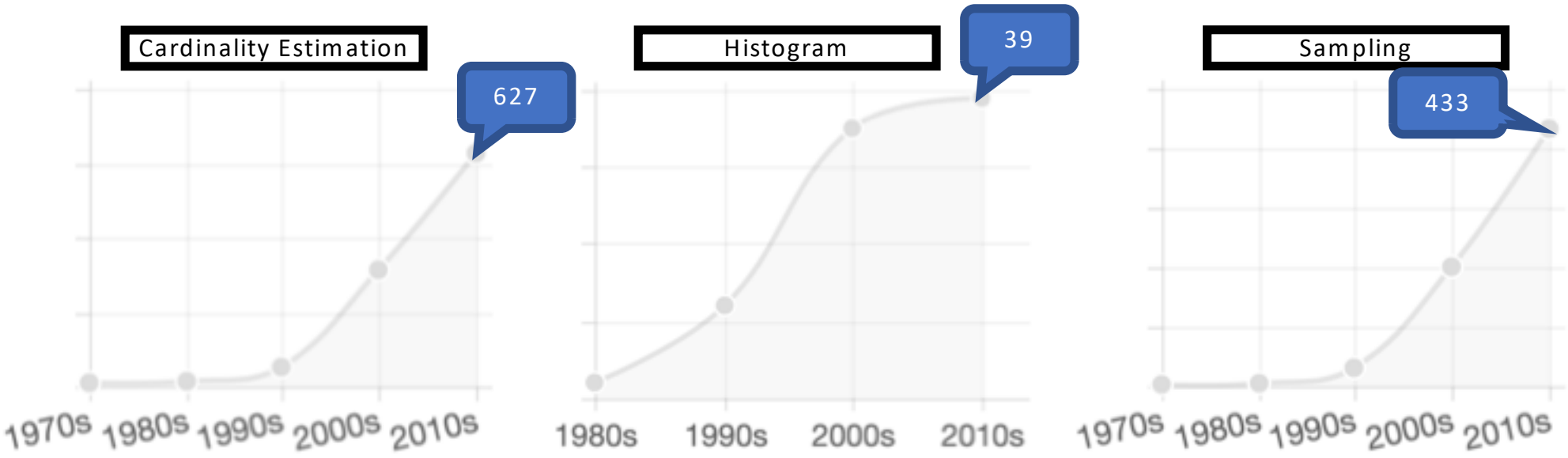
Leis et al observe (among other) that:

- Accuracy of cardinality estimation is critical to good plans
  - Especially hard in the presence of correlations between tables
- Risk associated with certain plan choices should be considered
  - E.g., nested-loops are seldom beneficial without an index

Active Research Areas

ACM DIGITAL LIBRARY

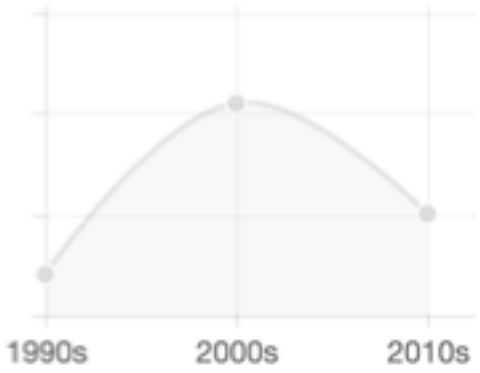
Sponsors: SIGMOD



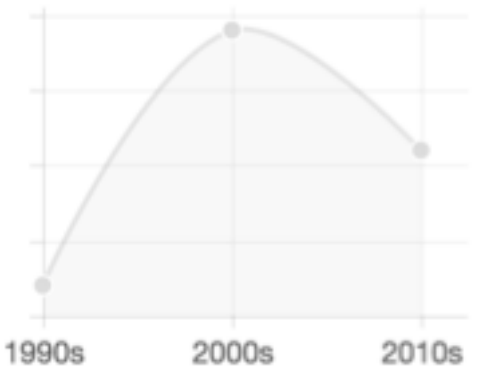
ACM DIGITAL LIBRARY

Sponsors: SIGMOD

Wavelets



Synopses



Recent Success of Deep Learning

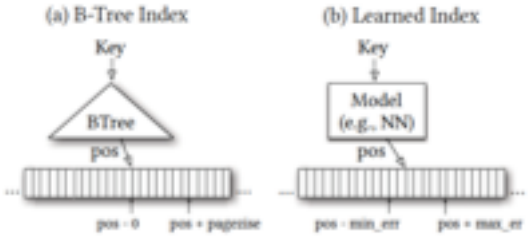
Black-and-white image colorization

Object detection and classification

Text and image translation

Automatic game playing

Indexing! [SIGMOD'18]



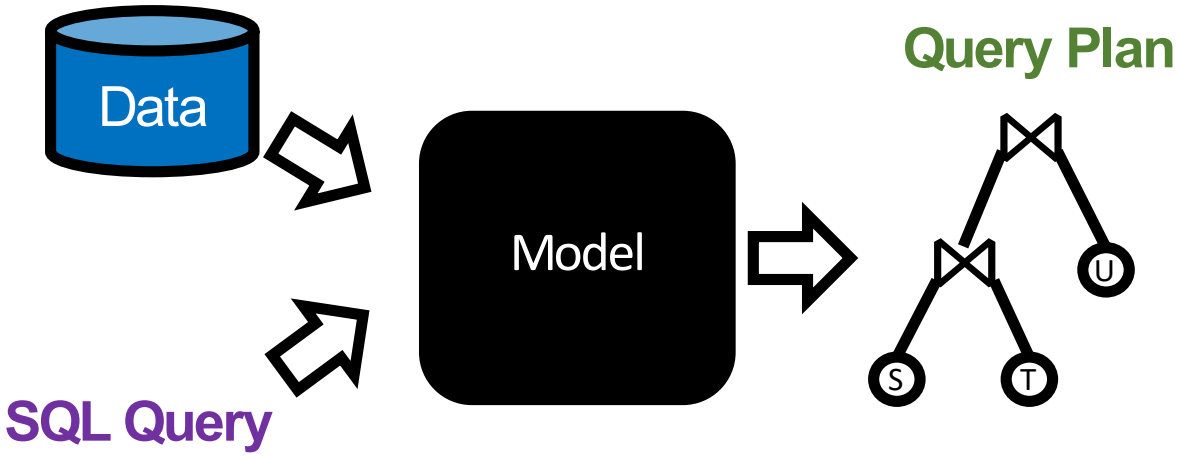
The Case for Learned Index Structures

Tim Kraska* MIT kraska@mit.edu
Alex Beutel Google, Inc. abeutel@google.com
Ed H. Chi Google, Inc. edchi@google.com
Jeffrey Dean Google, Inc. jeff@google.com
Neoklis Polyzotis Google, Inc. npoly@google.com

Examples & Images (except indexing) from: https://machinelearningmastery.com/inspirational-applications-deep-learning/

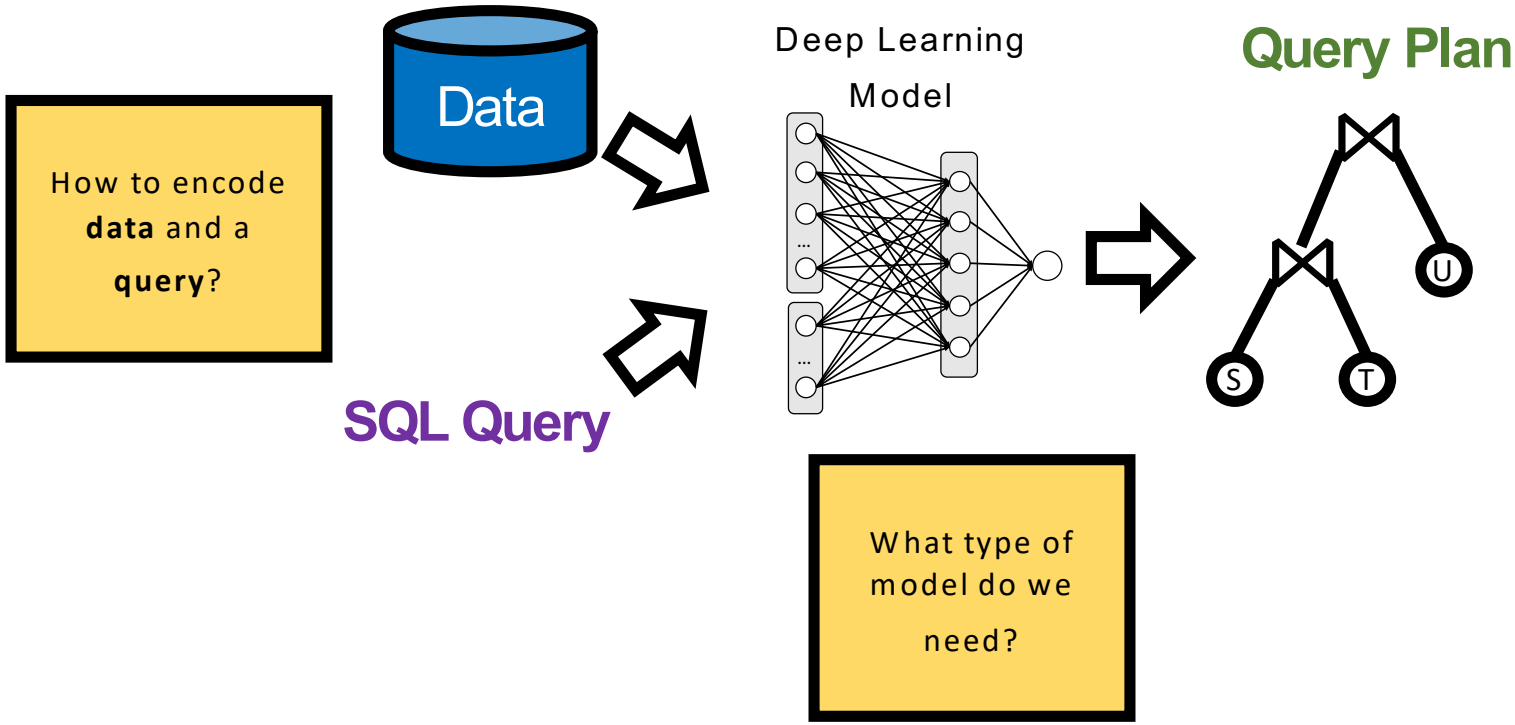
Our Vision

Can we rethink **query optimization** in the context of deep learning?



Our Vision

Can we rethink **query optimization** in the context of deep learning?

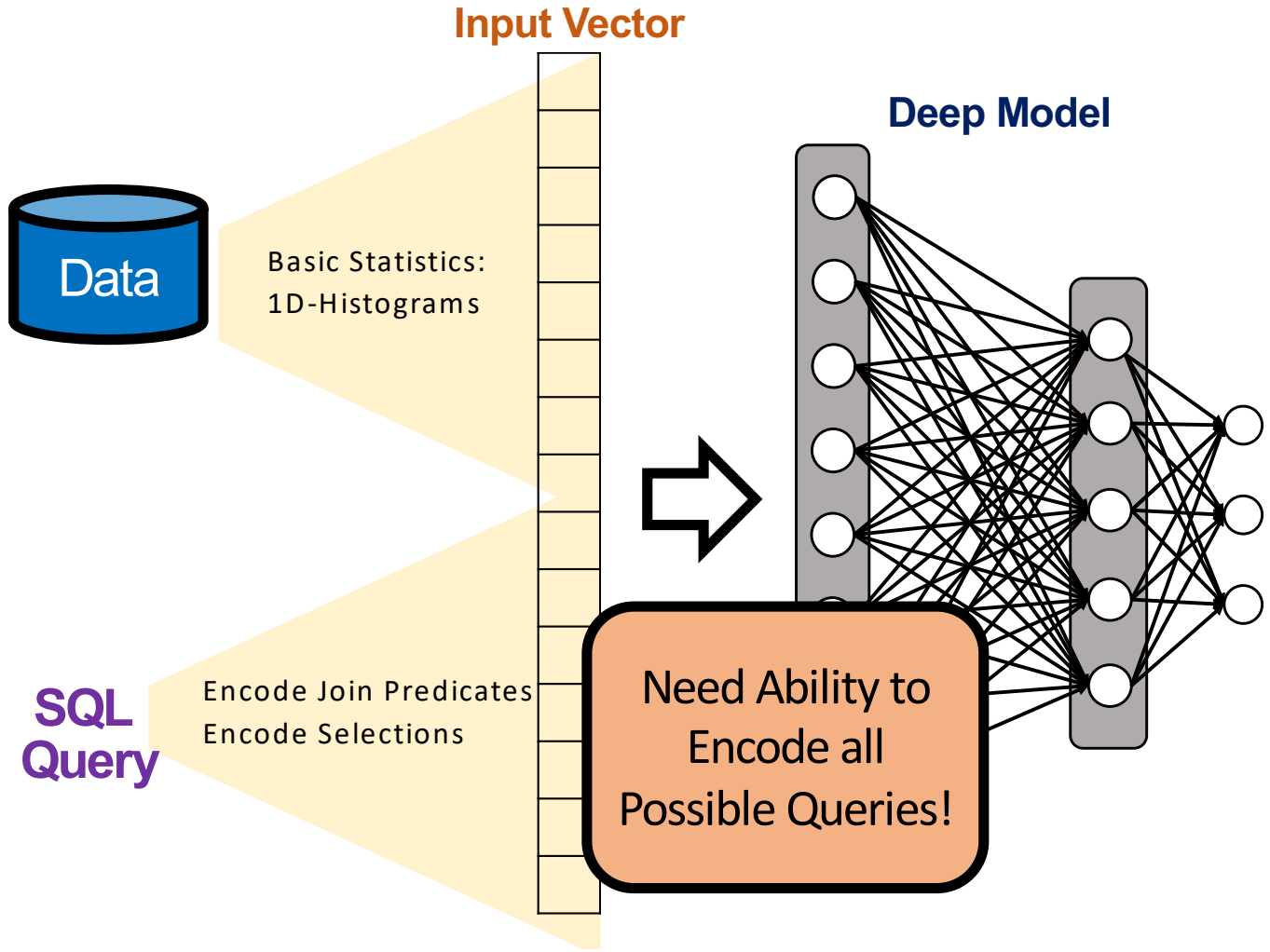


Can Deep Learning Help Query Opt?

- Can we automatically learn a query (subquery) representation?
  - That suffices to predict the query cardinality?
  - That can serve to derive the representation for more complex queries?

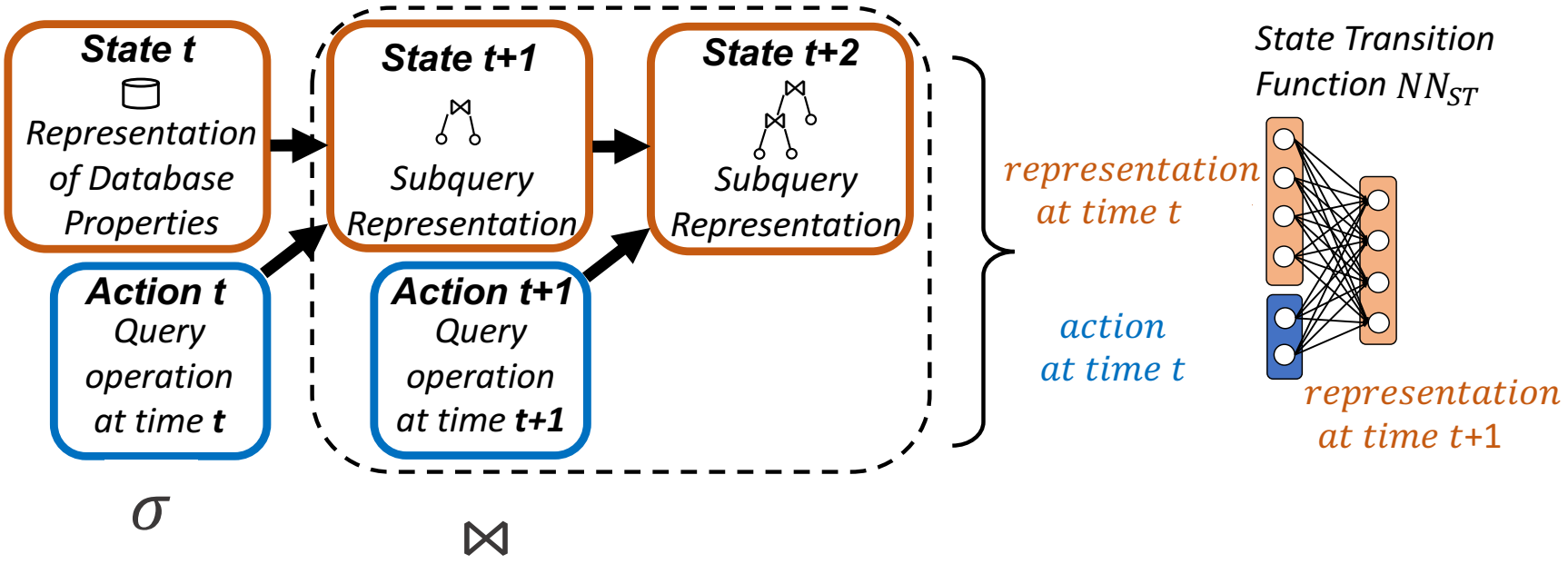
- Can we use the learned representation to find good plans?
  - Combine it with reinforcement learning to incrementally build plans

A Deep Learning Model for Query Optimization

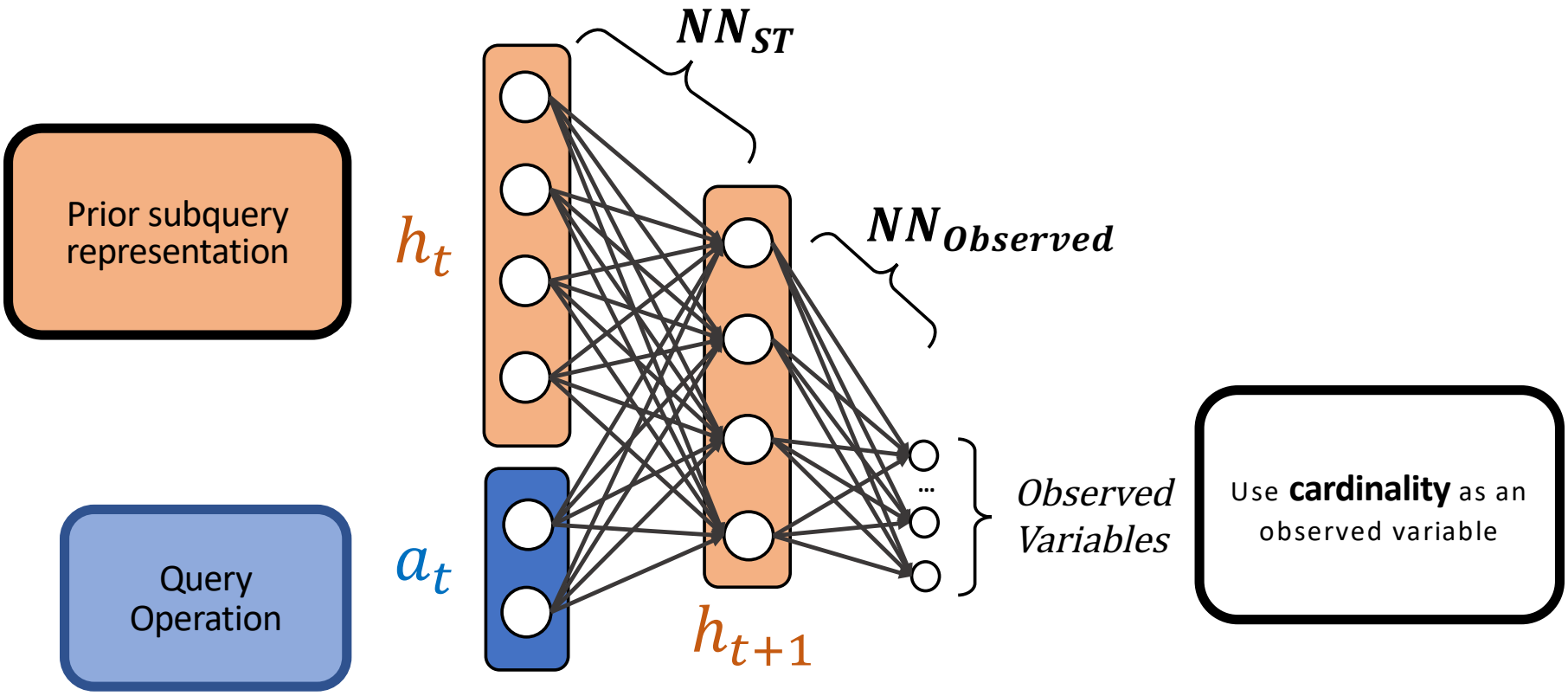


Learning Subquery Representations

- Use deep learning to capture **properties** of intermediate relations

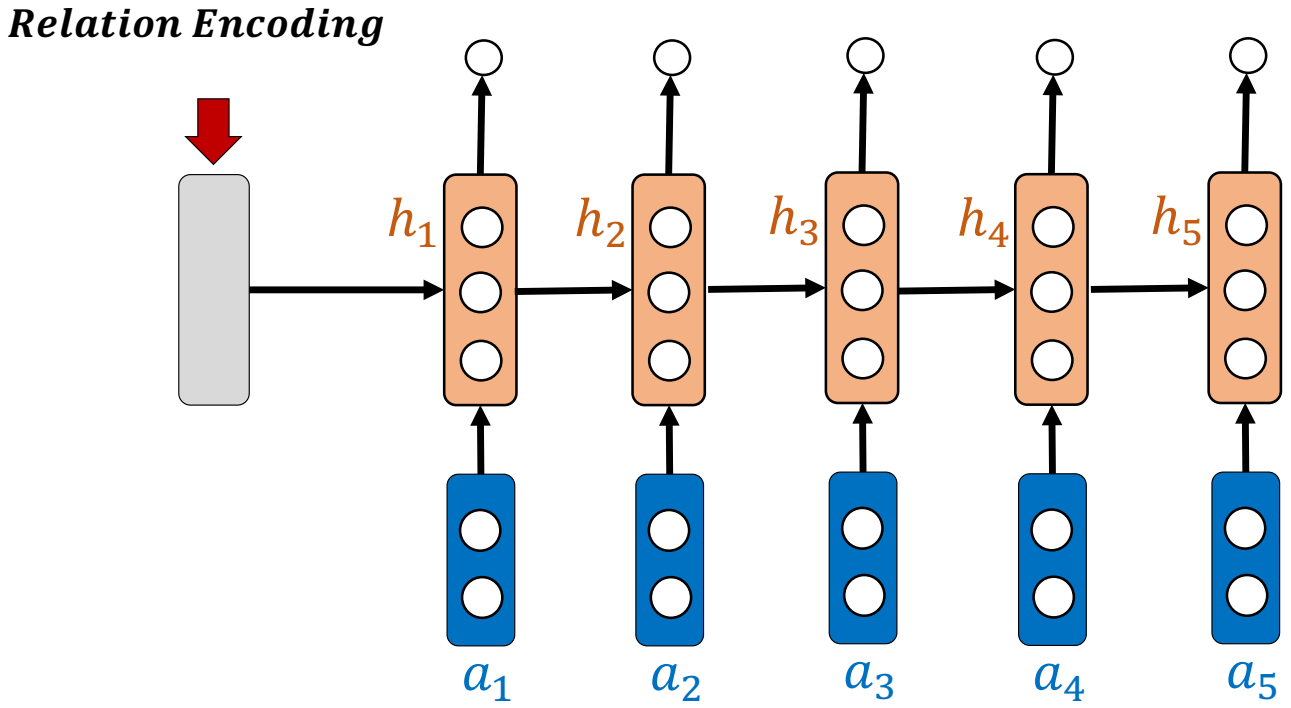


A Recursive Function



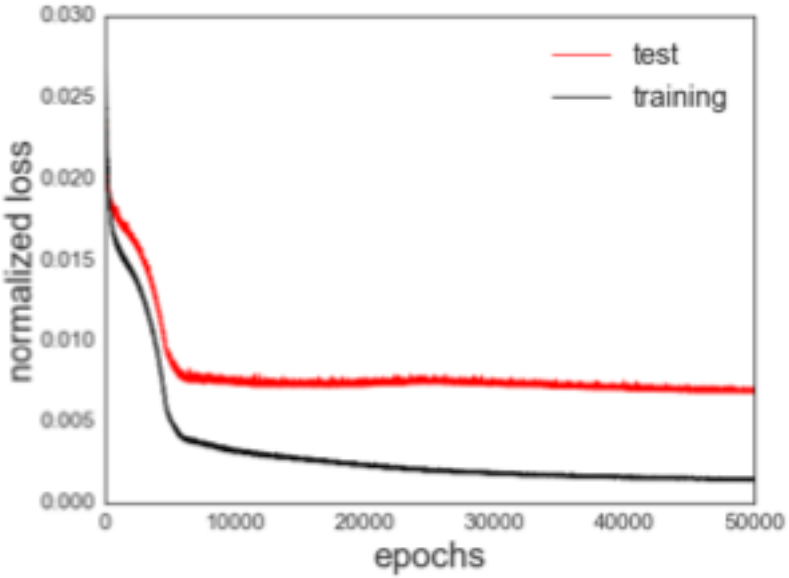
Learning the Recursive Function

- One approach is to train a recurrent neural network (RNN)



Preliminary Experiments

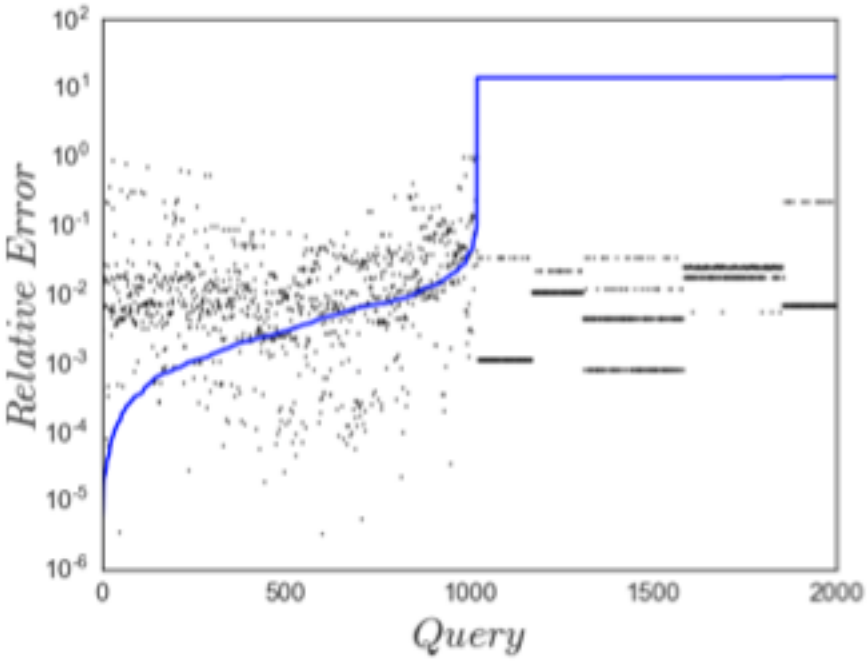
- Synthetic database generated using the PDGF Framework
  - 8 relations (1 fact table and 7 dimension tables)
  - Data size ~1.6 GB
- 8k random queries
  - 6k for training/2k for testing
- Select-join queries
  - All queries have 5 selection and/or joins
- Implementation in Tensorflow



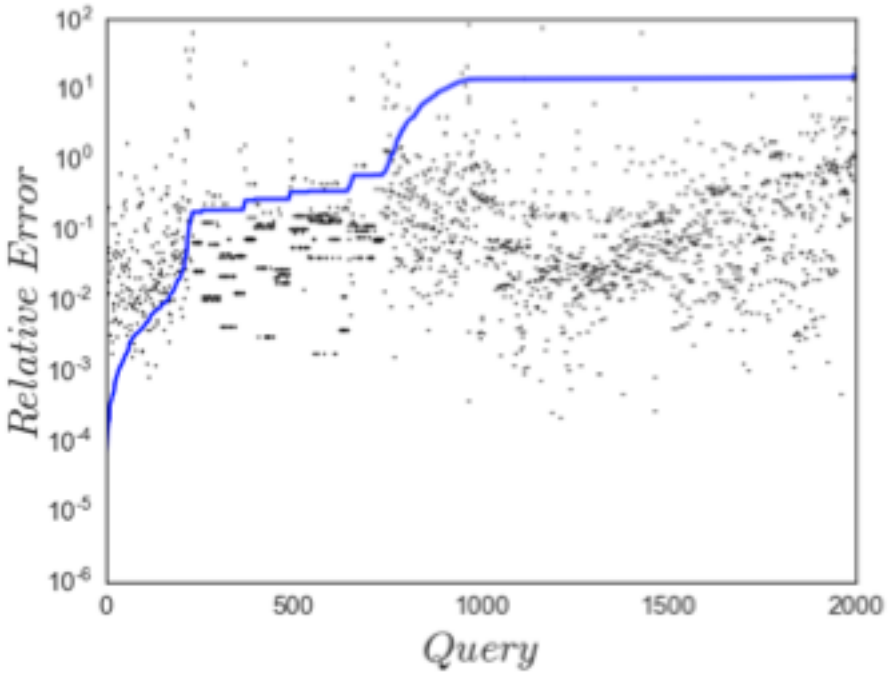
Results

- Predictions compared to Postgres estimates
- First action is either a selection or a join

First Action

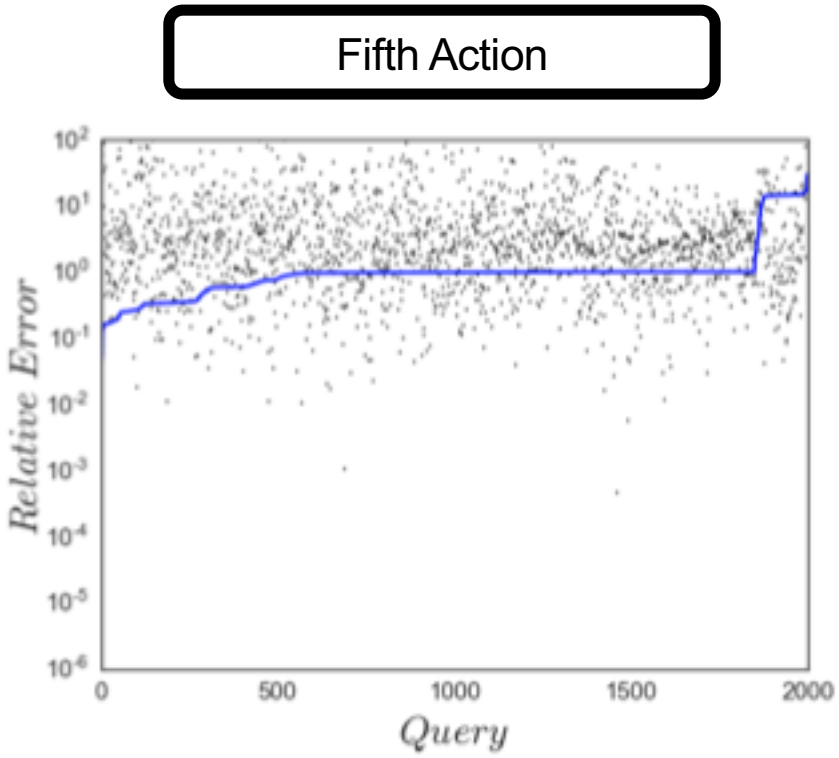


Second Action



Results

- Need to improve predictions for later actions



Can Deep Learning Help Query Opt?

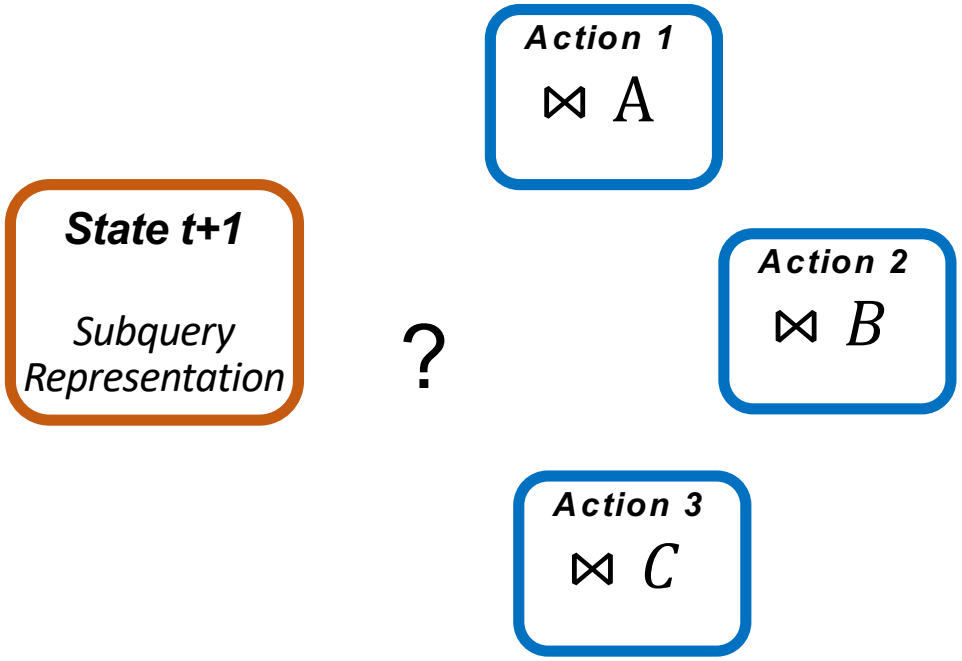
- Can we automatically learn a query (subquery) representation?
  - That suffices to predict the query cardinality?
  - That can serve to derive the representation for more complex queries?

- **Can we use the learned representation to find good plans?**
  - Combine it with reinforcement learning to incrementally build plans

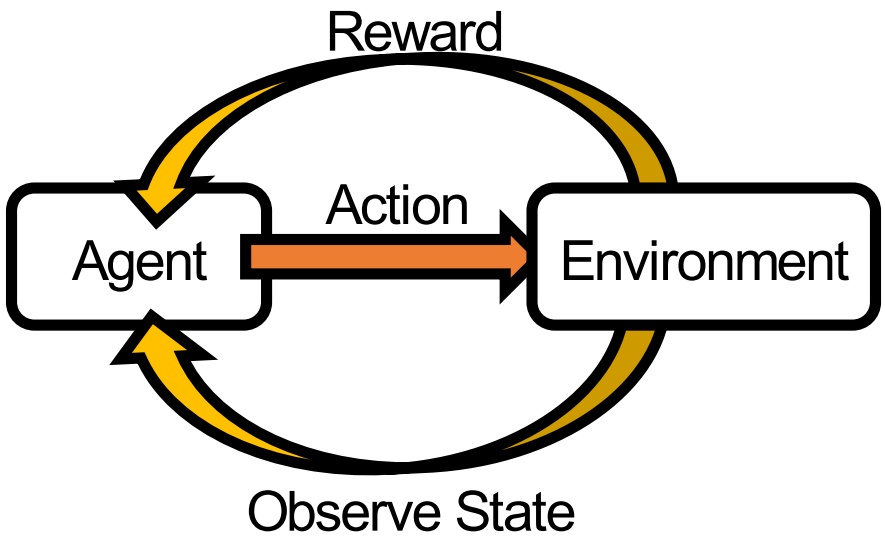
How to Find Good Query Plans?

- Use Subquery Representations with **Reinforcement Learning**

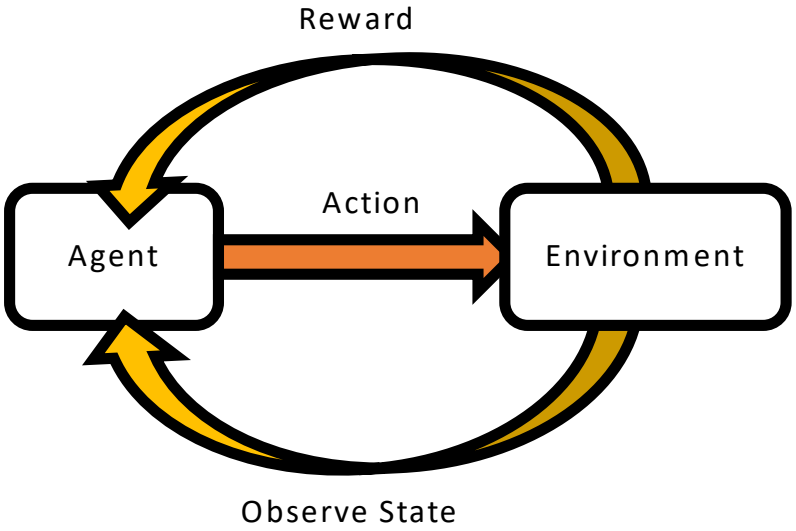
- **Given:** A ***subquery representation*** and a ***set of possible actions***, which is best?

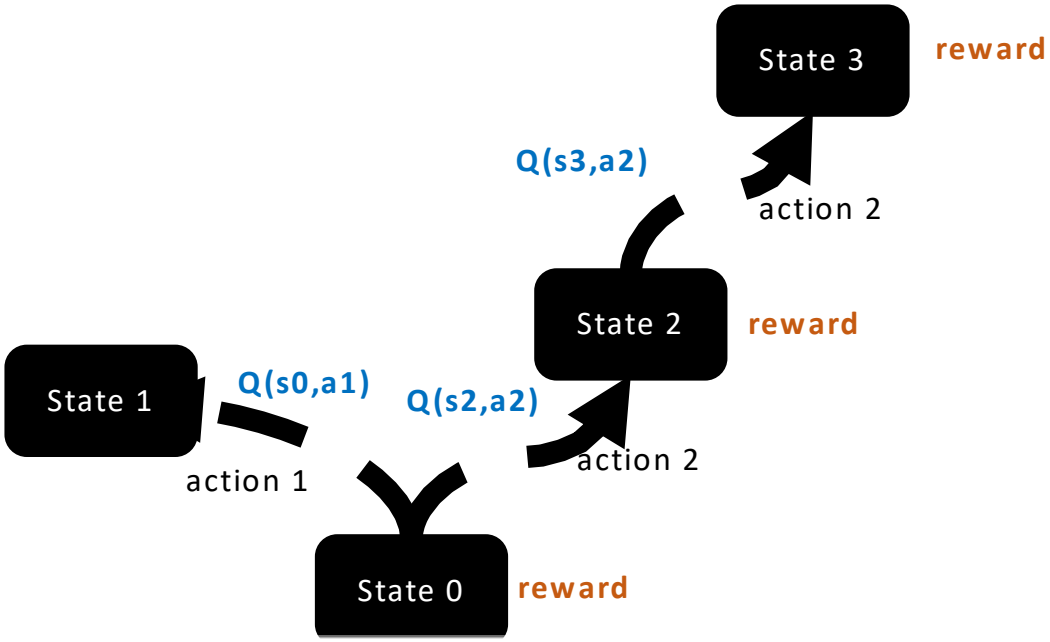


Reinforcement Learning



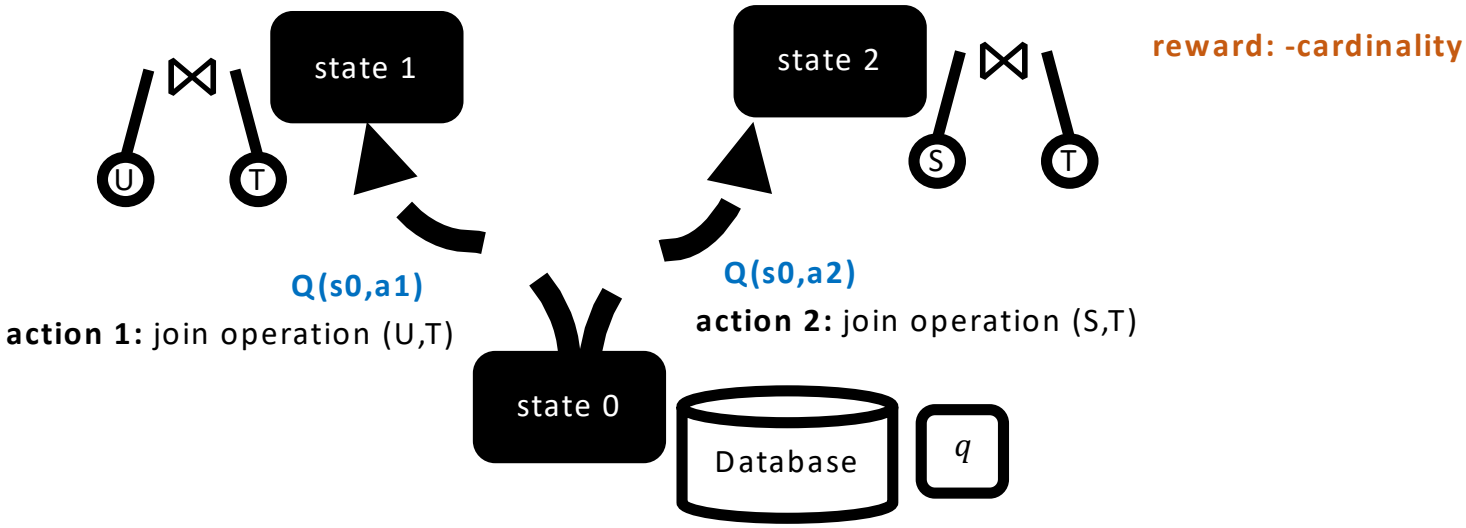
Reinforcement Learning: Q-learning





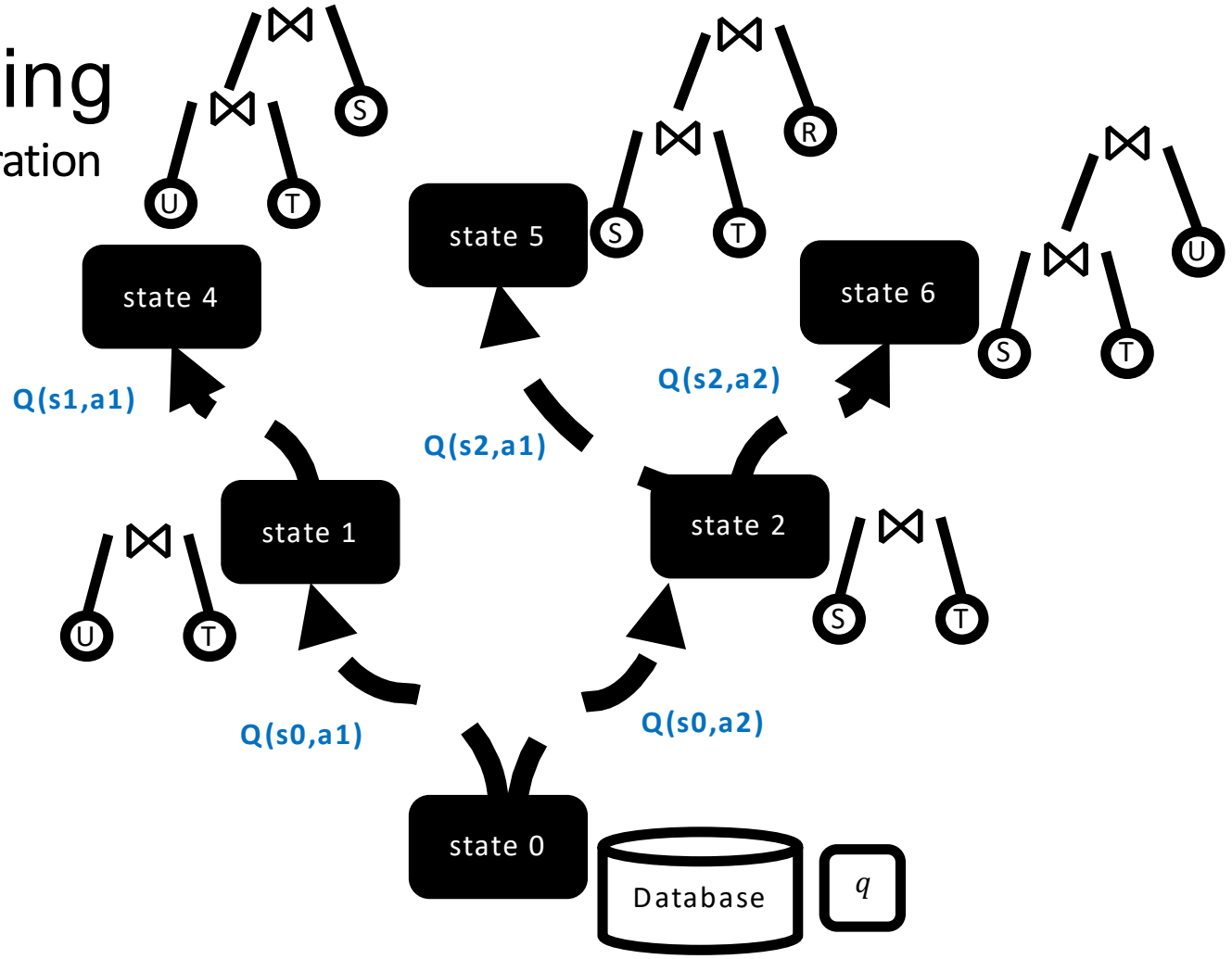
$$Q^{new}(s_t, a_t) \leftarrow (1-\alpha) \cdot \underbrace{Q(s_t, a_t)}_{old\ value} + \underbrace{\alpha}_{learning\ rate} \cdot \overbrace{\left(\underbrace{r_t}_{reward} + \underbrace{\gamma}_{discount\ factor} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{estimate\ of\ optimal\ future\ value} \right)}^{learned\ value}$$

Reinforcement Learning with Subquery Representations



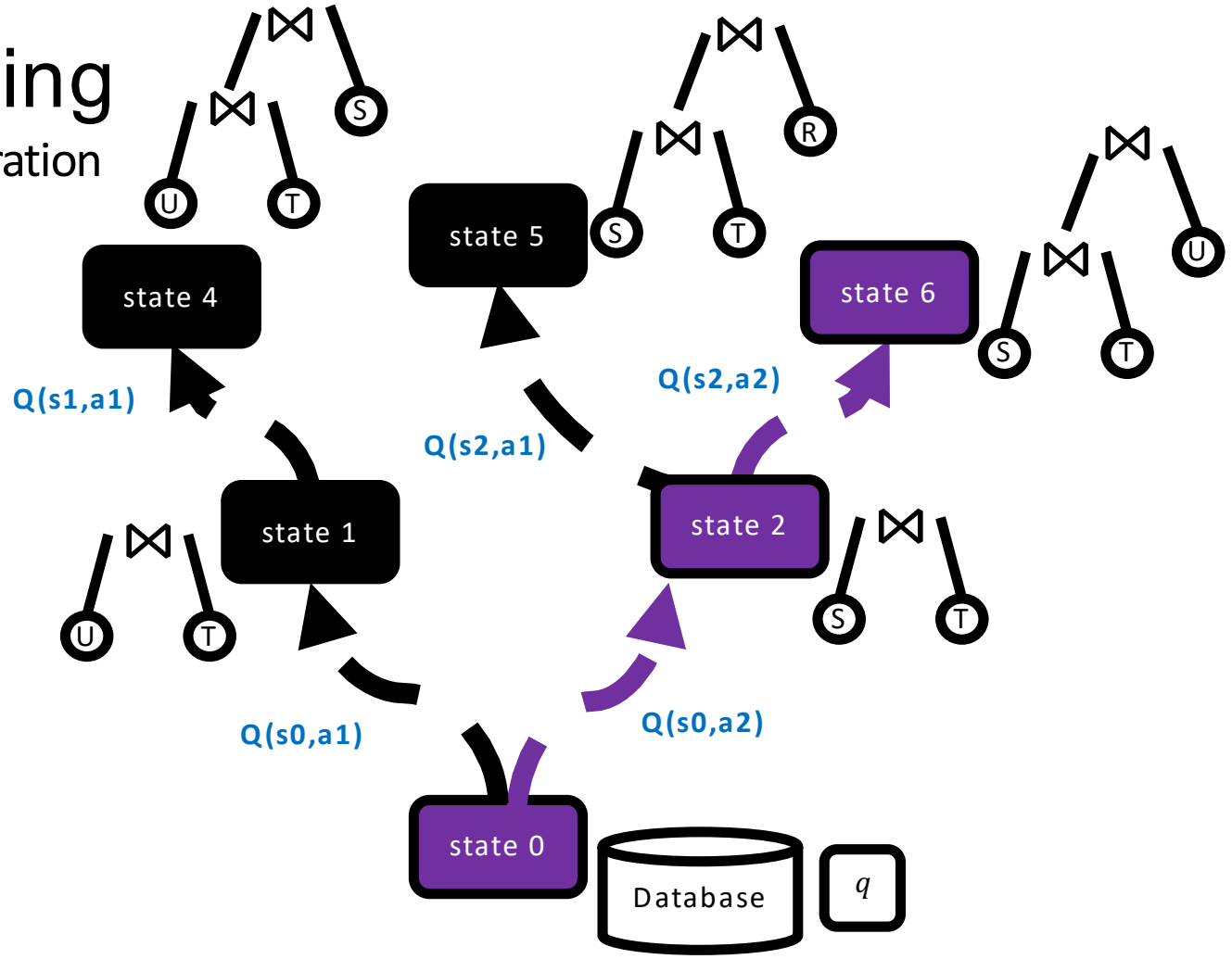
Q-Learning

Value-based iteration



Q-Learning

Value-based iteration



Project Status

- Short paper outlines the vision: **Learning State Representations for Query Optimization with Deep Reinforcement Learning**. Jennifer Ortiz, Magdalena Balazinska, Johannes Gehrke, and S. Sathiya Keerthi
  DEEM Workshop @ SIGMOD 2018

- Experimenting with different approaches for both components of the problem

- Expect more results soon

Related Work

- The Case for Learned Index Structures. *Kraska, et. al.*
  - Use models to replace/enhance indexes on single attributes

- Deep Reinforcement Learning for Join Order Enumeration *Marcus, et. al.*
  - Using RL with Postgres estimates as the reward.

Conclusion

- Query optimization remains a hard problem
- Can we use deep learning and reinforcement learning to improve
  - Cardinality estimation?
  - Query plan selection?
- DeepQuery: Deep reinforcement learning for query optimization
  - Learn **subquery representations** through a recursive function
  - Use **reinforcement learning** for join enumeration
  - Preliminary results are promising