DATA ANALYTICS USING DEEP LEARNING

GT 8803 // FALL 2018 // JOY ARULRAJ

LECTURE #19:
LEARNING STATE REPRESENTATIONS FOR QUERY OPTIMIZATION WITH DEEP REINFORCEMENT LEARNING

Georgia Tech

CREATING THE NEXT®

PAPER

- **Learning State Representations for Query Optimization with Deep Reinforcement Learning**
  – *Jennifer Ortiz, Magdalena Balazinska, Johannes Gehrke, S. Sathiya Keerthi*
  – University of Washington , Microsoft , Criteo Research
- **Key Topics**
  – Deep reinforcement learning
  – Query optimization

RELATED LINKS

- Paper - https://arxiv.org/abs/1803.08604

- *J Ortiz - https://homes.cs.washington.edu/~jortiz16/*
- *M Balazinska - https://www.cs.washington.edu/people/faculty/magda*
- *J Gehrke - http://www.cs.cornell.edu/johannes/*
- *SS Keerthi - http://www.keerthis.com/*

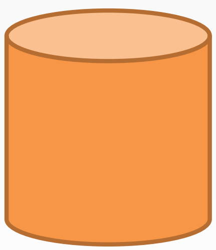
AGENDA

- Problem Overview
- Background
- Key Ideas
- Technical Details
- Experiments
- Discussion

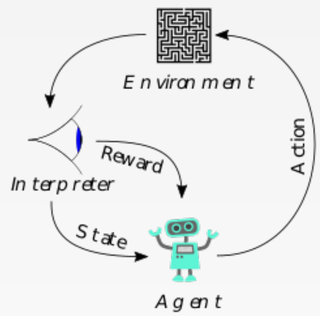
TODAY'S PAPER

```
SELECT   *
FROM     FRUIT F INNER join FRUIT_COLOR FC
ON       F.color = FC.id
WHERE    F.name='orange'
```

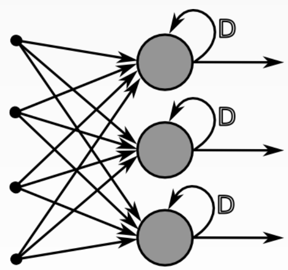
Query



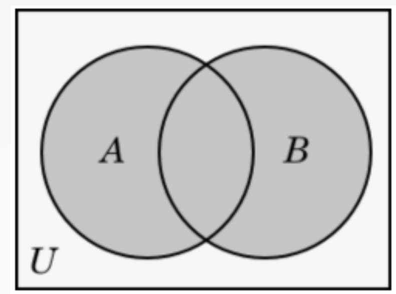
Database



Reinforcement Learning



Deep Learning



Query Cardinality

PROBLEM OVERVIEW

- **Query Optimization** is still a difficult problem
- **Deep Reinforcement Learning (DRL)** is an evolving approach to solve complex problems.
- *Can DRL be used to improve query plan optimization?*

PROBLEM OVERVIEW

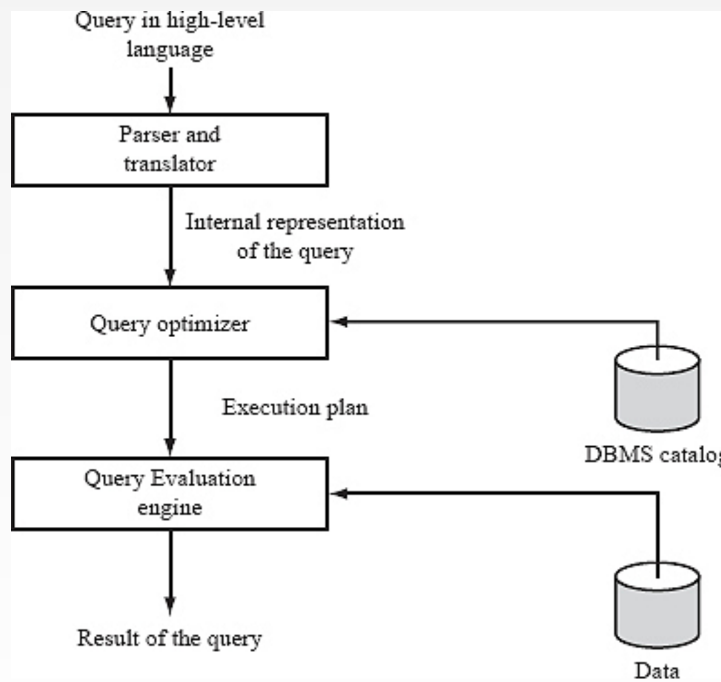
- **Contribution #1**: Generate a model that determine a subquery's cardinality
- **Contribution #2**: Use reinforcement learning as a Markov process to propose a query plan
- **Some Challenges:**
  - State isn't obvious like in some contexts (e.g. games)
  - Choosing the reward can be tricky

BACKGROUND: QUERY OPTIMIZATION

- Ongoing problem in database systems research
- Current systems still aren’t great - Why???
  - Plans must be efficient in time and resources - tradeoffs
  - Current DBMSs make simplified assumptions
    - Avoid multidimensional/complex methods
  - Result -> Estimation errors and poor query plans

BACKGROUND: QUERY OPTIMIZATION

- Join order
  - When join includes more than 2 relations, join time can vary depending on size of relation
- Subquery optimization
  - group by, exists operators can often be simplified, but…
  - can be computationally complex to determine
- Cardinality estimation
  - Hard to map predicates as new data comes in
  - Requires stats to be updated



https://en.wikipedia.org/wiki/Query_optimization

EXPRESS LEARNING - DATABASE MANAGEMENT SYSTEMS

BACKGROUND: QUERY OPTIMIZATION

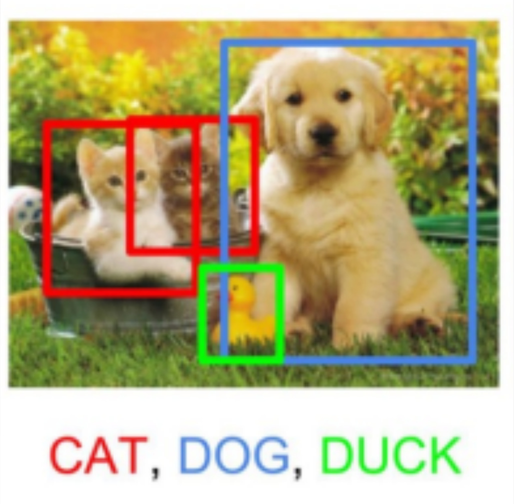
- Commonly used approaches
  - Data sketches
  - Sampling
  - Histograms
  - Heuristics

BACKGROUND: DEEP LEARNING

- What is it?
  - Maps input *x* to output *y* though a series of hidden layers.
  - Transforms data into *representations*
    - e.g. images of cats become pixels
  - Hidden layers apply of series of functions
  - Errors decrease over time via backpropagation

BACKGROUND: DEEP LEARNING

- What is it good for?
  - Machine translation
  - Object detection
  - Winning games
  - Much more…

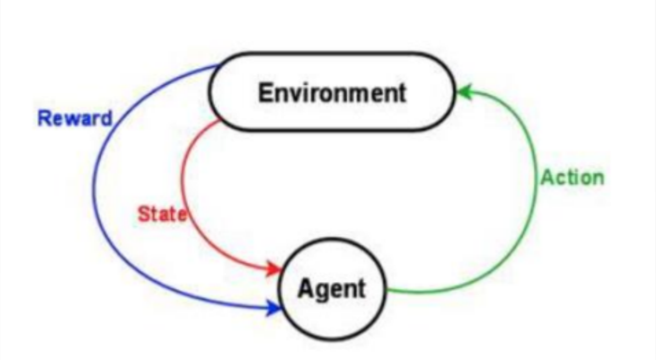


BACKGROUND: DEEP LEARNING

- Why?
  - Performs well across multiple domains
  - We have improved, cheaper hardware and large datasets for training
  - It's good at finding patterns that aren't obvious to humans (even domain experts)
  - Libraries
    - PyTorch, TensorFlow, Keras

BACKGROUND: REINFORCEMENT LEARNING

- What is it?
  - **Agents** – the learner in the model
  - **States** – condition of the environment
  - **Actions** – Inputs from the agent (based on previous learning or trial/error)
  - **Rewards** – Feedback to agent to reward (or not)



http://introtodeeplearning.com/materials/2018_6S191_Lecture5.pdf

BACKGROUND: REINFORCEMENT LEARNING

- What is it good for?
  - Beating Atari games
  - Training autonomous vehicles, robots
  - Optimizing stocks, gambling, auction bids, etc.

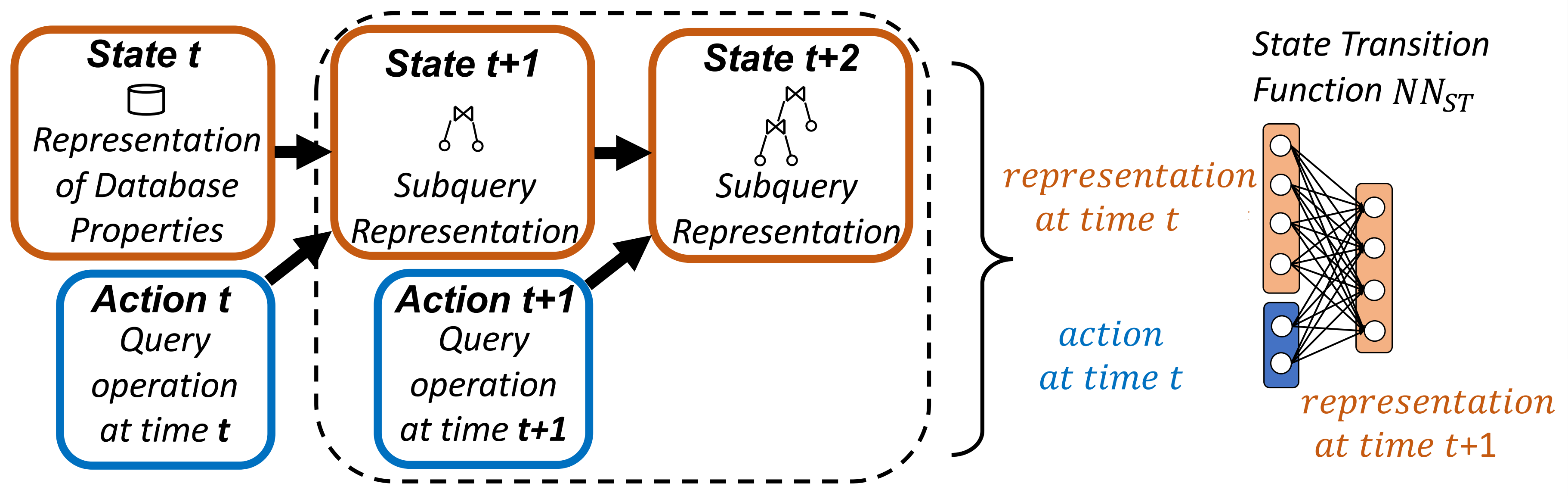
BACKGROUND: REINFORCEMENT LEARNING

- Why?
  - May perform better than brute-force deep learning models
  - Agents can use trial/error or greedy approaches to optimize reward
  - Can be good in complex state spaces because you don't have have to provide fully labeled outputs for the model to train on; can just provide more simpler rewards

KEY IDEA

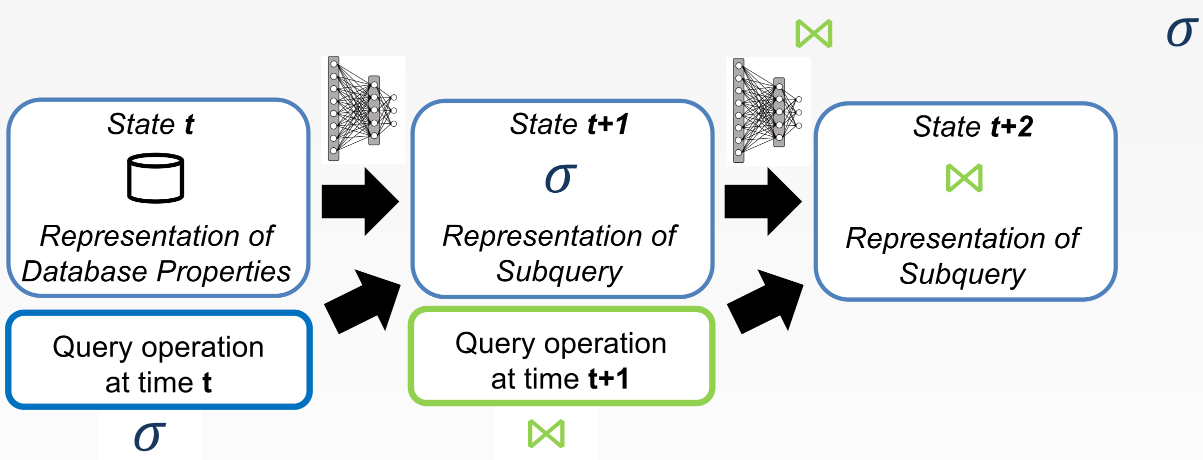
Can deep reinforcement learning be used to learn query representations?

SUBQUERY LEARNING VIA DRL



EXAMPLE

select * from customers C, orders O where C.col1 = O.col1 and **O.col1 <= 10**



APPROACH

- Map **query** and **database** to a **feature vector**

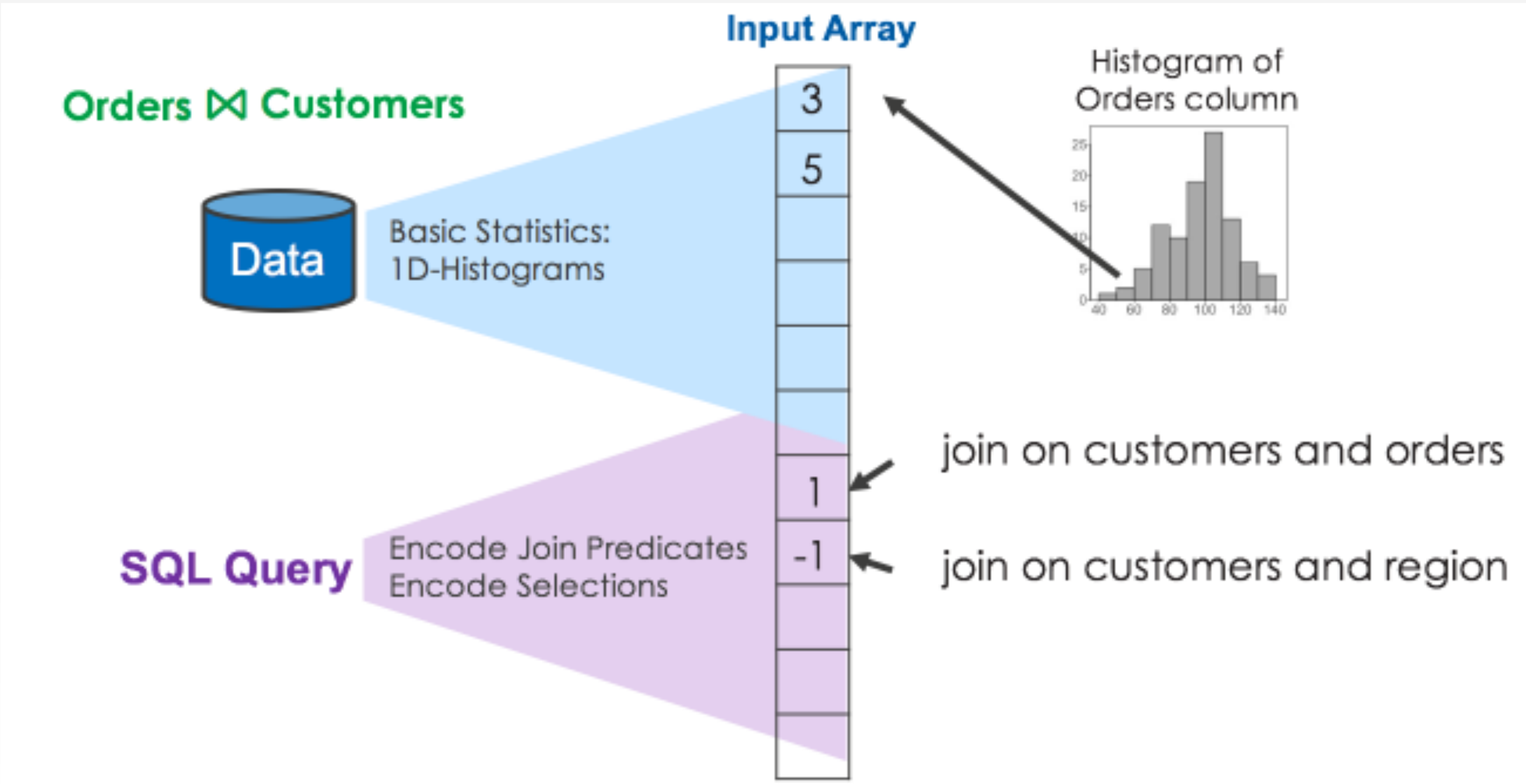
$$(Q,D) \longrightarrow x = \begin{bmatrix} x_1 \\ x_2 \\ \\ x_d \end{bmatrix}$$

- Two options:
  - Transform values using deep networks and output cardinality
  - Recursive approach taking subquery (h_t) and operation (a_t) as input

MORE ON APPROACH

- Two options:
  - Transform values using deep networks and output cardinality
    - Needs lots of data – very sparse
    - Recursive approach is selected
  - **Recursive approach** taking subquery (h_t) and operation (a_t) as input
    - h_t is learned by the model
    - Thus we have NN_{ST} model that learns based on $NN_{Observed}$ and NN_{Init}

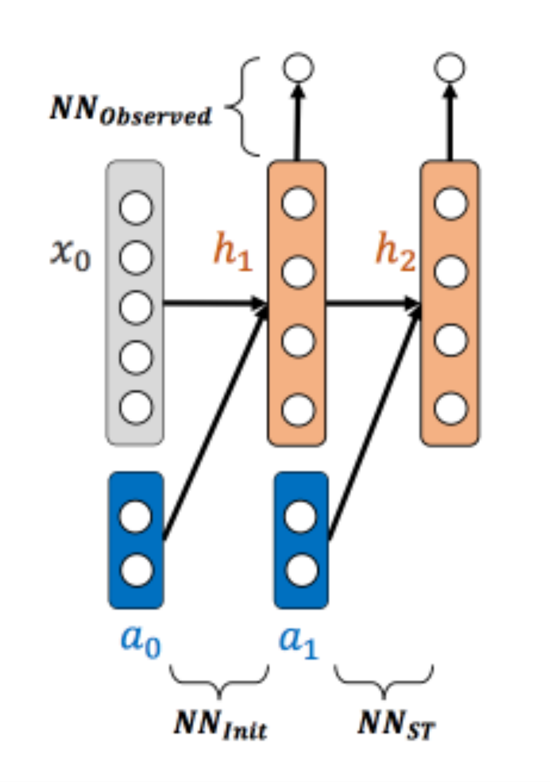
HOW TO ENCODE DATA



TECHNICAL DETAIL

Prior subquery representation

Query Operation



Use **cardinality** as an observed variable

STEPS

- NN$_{Init}$ = $f(x_0, a_0)$

  x = database properties (min/max values, # distinct values, 1D histogram)

  a $=$ single relational operator ($= \neq < > \leq \geq$)

- NN$_{ST}$ $= f(h_t, a_t)$

  h = latent representation of model itself (a subquery)

  a = single relational operation ($\bowtie$)

- NN$_{Observed}$

Mapping from hidden state to observed variables at time *t*

EXPERIMENTS

- Uses IMDB dataset
  - 3 GB
  - Real data (has skew and correlations between columns)
- TensorFlow (Python)
- Baseline estimates against SQL Server

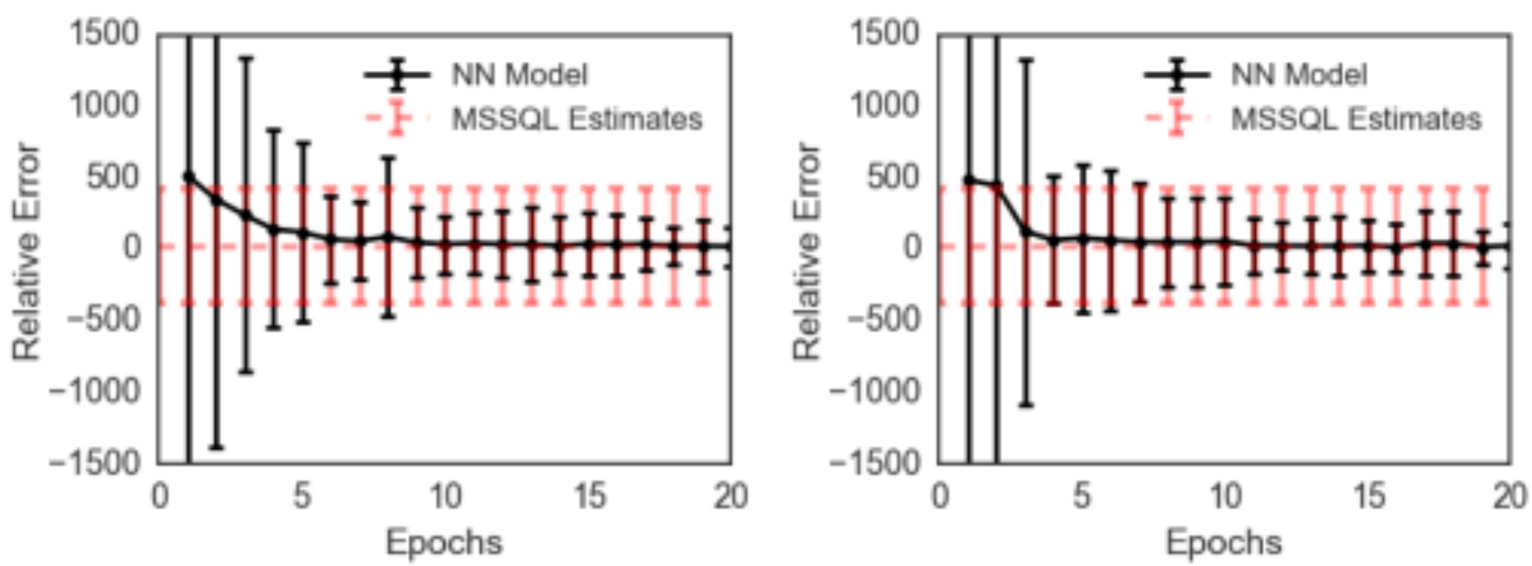
EXPERIMENT #1

- Train init function with properties from IMDB
- 20K queries (15K train/5K test)
- Model uses stochastic gradient descent (SGD)
- Learning rate of .01
- 50 hidden nodes in hidden layer

EXPERIMENT #1

- Fewer epochs == greater errors
- m=3, 6th epoch similar to SQL Server
- > 6th epoch, outperforms SQL Server
- Greater cardinality == longer to converge (outperforms SQL Server by 9th epoch)

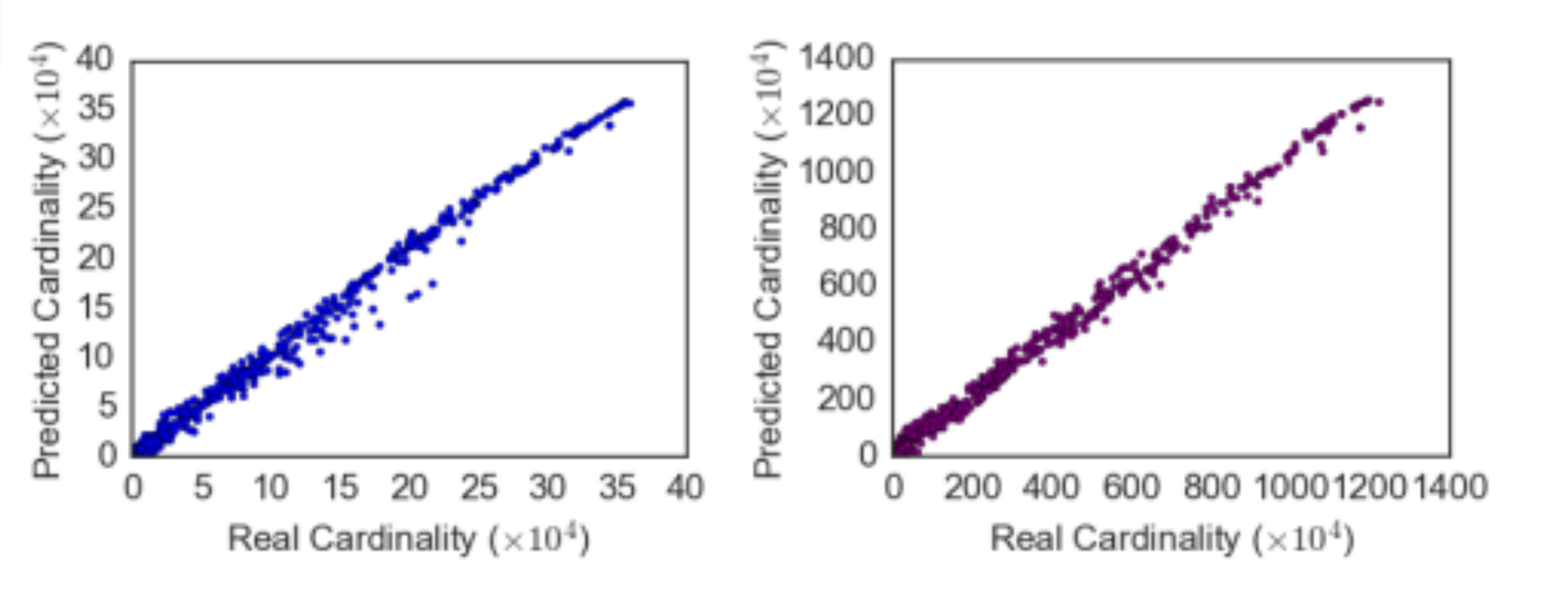
EXPERIMENT #1



EXPERIMENT #2

- Combined models
- Select and join operation
  - Where *a* is the join ($\bowtie$)
- Hidden state is able to store enough info to predict cardinality

EXPERIMENT #2

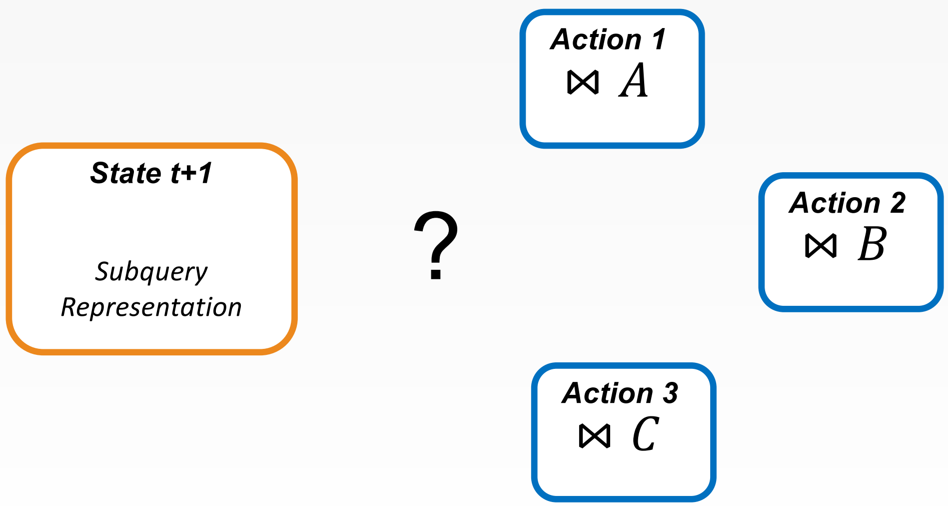


NEXT STEPS

Can subquery representations be used to build query plans?

GOAL

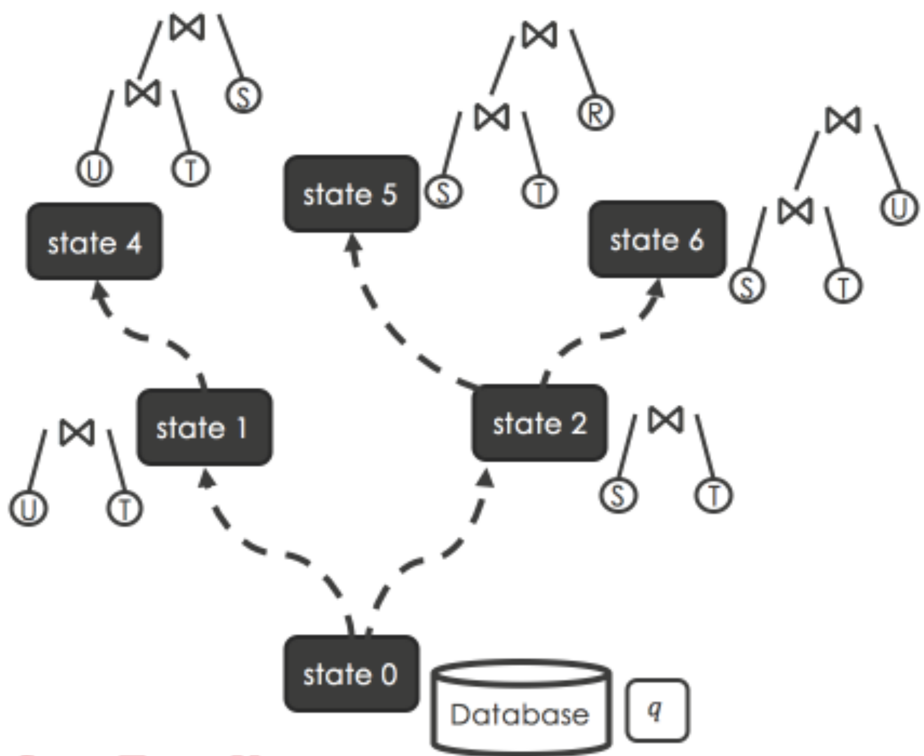
- Given a database *D* and a query *Q*, train a model that can learn to predict subquery cardinalities (and the best join)…



ASSUMPTIONS

- Model-free environment where probabilities between states are unknown
- Each state encodes operations that have already been done
- The model needs a good reward to be successful
- Need to determine optimal policy

EXAMPLE



Example: $S \bowtie T \bowtie U$

APPROACH

- For all relations in a database, assume a set of relations with attributes
- Vector at time *t*, represents equi-join predicates and 1D selection predicates
  - e.g. if a predicate exists, set value to 1, otherwise 0

HOW TO REWARD?

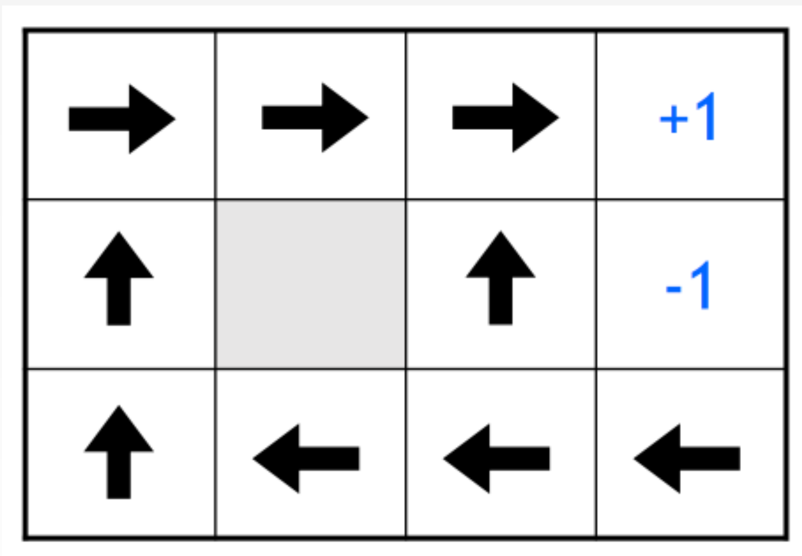
- Can be given at each state or at the end.
- Option 1:
  - Minimize cost based on existing query estimators
- Option 2:
  - Use cardinality from learned model
  - Experimental

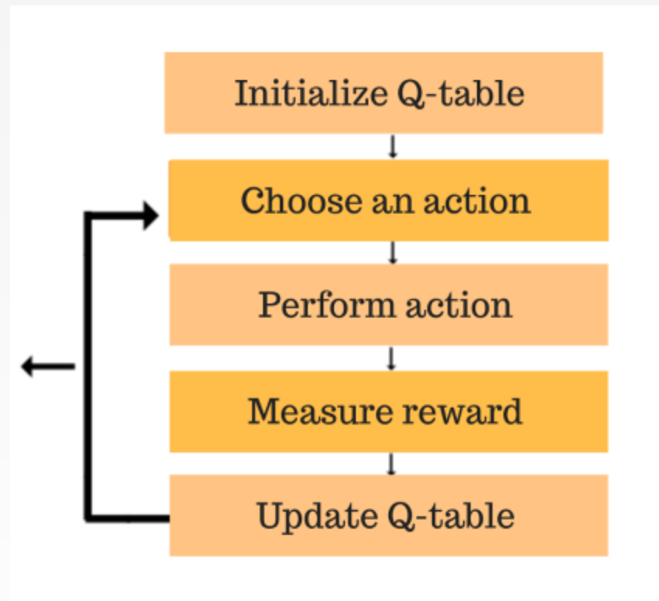
Q-LEARNING

- Init with random values
- For each state, the next value of Q comes from:
  - Current value of Q
  - Learning rate
  - Reward
  - Max value for a reward given a greedy policy
  - Discount factor

$$QL(s_t, a_t) \leftarrow QL(s_t, a_t) + \alpha[r_{t+1} + \gamma max_{a'} QL(s_{t+1}, a') - QL(s_t, a_t)] \quad (1)$$

Q-LEARNING





https://medium.freecodecamp.org/an-introduction-to-q-learning-reinforcement-learning-14ac0b4493cc

OPEN PROBLEMS

- How to choose rewards?
- State space is large and Q-learning can be impractical
  - Need to approximate solutions

RELATED WORK

- Eliminate optimizer
- Use RL for query processing
- Feedback loop on optimizer
- Neural networks to estimate cardinality
- Neural networks to build fast indexes
- DRL to determine join order

STRENGTHS

- Deep learning is a more feasible approach than manually written queries
- Unique approach with using recursive model
- Deep learning models can approximate and exceed performance of industry-standard optimizers

WEAKNESSES

- Q-Learning is impractical and difficult
  - Large state space
  - Reward selection problem
- Evaluating query plans takes time, but so does training iterative models, would be valuable to compare.

MODEL DETAILS (NOT IN PAPER)

- Space: 1MB – 2MB
- Prediction Time: ~1ms
- Training Time: 20min – 1hr

FURTHER DISCUSSION

- Strategies to pick a reward function?
- For actions, discuss a value-based recursive approach vs. a policy gradient approach.
- Is there a way to pick the most representative queries to reduce state space?

REFERENCES

1. Slides from Jennifer Ortiz, "Deep Learning for Query Plan Resource and Cost Estimation", Teradata Analytics Universe, 2018.
2. LIMITED, I. E. (2012). *EXPRESS LEARNING - DATABASE MANAGEMENT SYSTEMS*. S.l.: PEARSON EDUCATION INDIA.
3. MIT 6.S191: Introduction to Deep Learning, http://introtodeeplearning.com/