ReJOIN: A Prototype Query Optimizer using Deep Reinforcement Learning

Ryan Marcus*, Brandeis University
Olga Papaemmanouil, Brandeis University

10/3/2018

These slides:
http://rm.cab/ibm18
Join Order Enumeration

- Classic problem in query optimization
- For 12 joins, ~100 trillion possibilities
- Cost-based optimization (using cost model)

SELECT * FROM A, B, C, D WHERE ...
Join Order Enumeration

- System R: algorithm for best left-deep
- PostgreSQL: bottom-up pairwise, pruned
  - Use a genetic optimizer for $n > 12$ relations
- SQL Server: complex heuristics
- Other heuristics…

- Requires per-DB tuning to get right
- Fire and forget
ReJOIN

- A join order enumerator
- Uses reinforcement learning
  - Learns from its mistakes
- Automatically tunes itself over time
- Finds better join orderings
- Finds join orderings faster
- Even a straightforward application of deep learning produces interesting results
Reinforcement Learning

- Agent observes a state
  - Info about the world
  - Set of possible actions
- Agent selects an action, gets:
  - A reward
  - New state
- Goal: maximize reward over time
  - explore and exploit
Deep Reinforcement Learning

• Previous RL has been severely limited…
  – Q-learning, state table
  – REINFORCE, action space

• Applications of RL to QO have also been limited
  – LEO, self-tuning histograms (cardinality est)

• Deep RL → sudden ability to handle large problems
  – Complex video games, walking, real-world navigation
ReJOIN

- Each state is a partial join order
- Each action fuses two partial orderings
- Reward is the optimizer cost
ReJOIN

- Each state is a partial join order
- Each action fuses two partial orderings
- Reward is the optimizer cost
ReJOIN

- Each state is a partial join order
- Each action fuses two partial orderings
- Reward is the optimizer cost

Possible actions:
(A, B), (B, A), (A, C), (C, A), (A, D), (D, A), (B, C), (C, B), (B, D), (D, B), (C, D), (D, C)
ReJOIN

- Each state is a partial join order
- Each action fuses two partial orderings
- Reward is the optimizer cost

Possible actions:
(A, B), (B, A), (A, C), (C, A), (A, D), (D, A), (B, C), (C, B), (B, D), (D, B), (C, D), (D, C)
ReJOIN

- Each state is a partial join order
- Each action fuses two partial orderings
- Reward is the optimizer cost

Possible actions:
- ([BA], C), (C, [BA]), ([BA], D),
- (D, [BA]), (C, D), (D, C)
ReJOIN

- Each state is a partial join order
- Each action fuses two partial orderings
- Reward is the optimizer cost

Possible actions:
([BA], C), (C, [BA]), ([BA], D), (D, [BA]), (C, D), (D, C)
ReJOIN

- Each state is a partial join order
- Each action fuses two partial orderings
- Reward is the optimizer cost

Possible actions: 
([[BA]D], C), (C, [[BA]D])
ReJOIN

- Each state is a partial join order
- Each action fuses two partial orderings
- Reward is the optimizer cost

Possible actions: $([[[BA]D], C), (C, [[[BA]D])$
ReJOIN

- Each state is a partial join order
- Each action fuses two partial orderings
- Reward is the optimizer cost

Possible actions:
Results

- PPO algorithm
- JOB (join order benchmark) queries
- 114 queries on real-world (IMDB) data
- 10 held-out

- We made you a VM: https://git.io/imdb

- Harder to train, **better** plans, **faster** queries, **stronger** component.
Convergence

Harder to train: convergence is long and noisy.
Better results according to cost model
Latency

Faster latency when actually executed
Planning Time

![Graph showing planning time for different numbers of relations]

**PostgreSQL**
- ReJOIN (with updates)
- ReJOIN (no updates)

**Stronger join order selection algorithm**
Challenges & Next Steps

• Challenge #1: Dependence on cost model
  – Train directly based on latency? What about the bad plans?

• Challenge #2: Convergence / bootstrapping
  – Learn from demonstration? Promising, but we can’t have our cake and eat it too.

• Challenge #3: Debugging
  – Activation maps? Much more difficult to explain a decision than in a traditional optimizer.
Challenges & Next Steps

● Challenge #1: Dependence on cost model
  − Train directly based on latency? What about the bad plans?

● Challenge #2: Convergence / bootstrapping
  − Learn from demonstration? Promising, but we can’t have our cake and eat it too.

● Challenge #3: Debugging
  − Activation maps? Much more difficult to explain a decision than in a traditional optimizer.

Common challenges among many DL / QO papers
Addressed in submitted CIDR vision paper (under review)
Conclusions

• If a proof-of-concept application of deep learning produced these results, imagine what a nuanced approach might achieve!

Workshop paper (aiDM@SIGMOD18):

http://rm.cab/rejoin
Thanks!

- Me: Ryan Marcus
- Email: ryan@cs.brandeis.edu
- Twitter: @RyanMarcus
- Web: http://ryanmarc.us
- These slides: http://rm.cab/ibm18