Learned Cardinalities: Estimating Correlated Joins with Deep Learning

Cardinality estimation problem
what it is + why is it hard

Key ideas

Discussion

CS294 AI-Sys
Presented by: Zongheng Yang
zongheng@berkeley.edu
Cardinality Estimation

Single-table

```
SELECT * FROM sal
WHERE sal.position = 'Manager I'
AND sal.salary > 100,000
```
Cardinality Estimation

Single-table

```sql
SELECT * FROM sal
WHERE sal.position = 'Manager I'
AND   sal.salary >100,000
```

Likely! (correlation)
Cardinality Estimation

Single-table

SELECT * FROM sal
WHERE sal.position = 'Manager I'
AND sal.salary > 100,000

SELECT * FROM twitter_graph
WHERE following = 'Michael Jordan'

SELECT * FROM cars
WHERE make = 'Honda'
AND model = 'Jetta'

Likely! (correlation)

Most! (uniformity)

Anti-correlation!

---

<table>
<thead>
<tr>
<th>emp_id</th>
<th>position</th>
<th>country</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Manager II</td>
<td>USA</td>
</tr>
<tr>
<td>2</td>
<td>Engineer I</td>
<td>CAN</td>
</tr>
<tr>
<td>3</td>
<td>Engineer II</td>
<td>USA</td>
</tr>
<tr>
<td>4</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sal_id</th>
<th>position</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Manager I</td>
<td>120000.00</td>
</tr>
<tr>
<td>2</td>
<td>Manager II</td>
<td>150000.00</td>
</tr>
<tr>
<td>3</td>
<td>Engineer I</td>
<td>78000.00</td>
</tr>
<tr>
<td>4</td>
<td>Engineer II</td>
<td>91000.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>tax_id</th>
<th>country</th>
<th>rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>USA</td>
<td>0.32</td>
</tr>
<tr>
<td>2</td>
<td>CAN</td>
<td>0.45</td>
</tr>
<tr>
<td>3</td>
<td>CHN</td>
<td>0.17</td>
</tr>
<tr>
<td>4</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Cardinality Estimation

Single-table

**SELECT** * FROM sal
WHERE  sal.position = ‘Manager I’
AND  sal.salary >100,000

**SELECT** * FROM twitter_graph
WHERE  following = ‘Michael Jordan’

**SELECT** * FROM cars
WHERE  make = ‘Honda’
AND  model = ‘Jetta’

**Likely!** (correlation)

**Most!** (uniformity)

**Anti-correlation!**

Reduction(query) = R(pred 1) * R(pred 2)
Reduction(col=val) = 1 / num_distinct(col)
# Cardinality Estimation

## Joins

```sql
SELECT * FROM emp, sal
WHERE emp.position = 'Manager I'
AND sal.salary > 100,000
```
Cardinality Estimation

Joins

```sql
SELECT * FROM emp, sal
WHERE emp.position = 'Manager I'
AND sal.salary > 100,000
```

```
correlated joins
```

<table>
<thead>
<tr>
<th>emp_id</th>
<th>position</th>
<th>country</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Manager II</td>
<td>USA</td>
</tr>
<tr>
<td>2</td>
<td>Engineer I</td>
<td>CAN</td>
</tr>
<tr>
<td>3</td>
<td>Engineer II</td>
<td>USA</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sal_id</th>
<th>position</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Manager I</td>
<td>120000.00</td>
</tr>
<tr>
<td>2</td>
<td>Manager II</td>
<td>150000.00</td>
</tr>
<tr>
<td>3</td>
<td>Engineer I</td>
<td>78000.00</td>
</tr>
<tr>
<td>4</td>
<td>Engineer II</td>
<td>91000.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>tax_id</th>
<th>country</th>
<th>rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>USA</td>
<td>0.32</td>
</tr>
<tr>
<td>2</td>
<td>CAN</td>
<td>0.45</td>
</tr>
<tr>
<td>3</td>
<td>CHN</td>
<td>0.17</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Cardinality Estimation

Joins

```
SELECT * FROM emp, sal
WHERE emp.position = 'Manager I'
AND sal.salary > 100,000
```

Reduction(join) = 1 / max {
   Cardinality("emp where emp.pos = Mgr1"),
   Cardinality("sal where sal.sal > 100K")
}
Figure 3: Quality of cardinality estimates for multi-join queries in comparison with the true cardinalities. Each boxplot summarizes the error distribution of all subexpressions with a particular size (over all queries in the workload).
How bad is it?

For 6-way joins: median 100x off, outliers up to $10^8$x off

**Figure 3: Quality of the error distributions**

VLDB’15, Leis et al., How Good Are Query Optimizers, Really?
Key Ideas

• Recall: uniformity & independence assumptions are bad
Key Ideas

• Recall: uniformity & independence assumptions are bad

• What if we give a model:

<table>
<thead>
<tr>
<th>Features</th>
<th>Labels (cardinality)</th>
</tr>
</thead>
<tbody>
<tr>
<td>following = Jordan</td>
<td>1 million (likely)</td>
</tr>
<tr>
<td>following = Nadorj</td>
<td>10 (unlikely)</td>
</tr>
<tr>
<td>age &lt; 20 &amp;&amp; salary &gt; 100K</td>
<td>1K (unlikely)</td>
</tr>
<tr>
<td>age &gt; 30 &amp;&amp; salary &gt; 100K</td>
<td>100K (likely)</td>
</tr>
</tbody>
</table>
Key Ideas

• Recall: uniformity & independence assumptions are bad

• What if we give a model:

<table>
<thead>
<tr>
<th>Features</th>
<th>Labels (cardinality)</th>
</tr>
</thead>
<tbody>
<tr>
<td>following = Jordan</td>
<td>1 million (likely)</td>
</tr>
<tr>
<td>following = Nadorj</td>
<td>10 (unlikely)</td>
</tr>
<tr>
<td>age &lt; 20 &amp;&amp; salary &gt; 100K</td>
<td>1K (unlikely)</td>
</tr>
<tr>
<td>age &gt; 30 &amp;&amp; salary &gt; 100K</td>
<td>100K (likely)</td>
</tr>
</tbody>
</table>

• It should then learn to fix unif./indep. assumptions!
Key Ideas

• This is exactly what they did!
Key Ideas

• This is exactly what they did!

Figure 2: Query featurization as sets of feature vectors.
Key Ideas

• This is exactly what they did!

“Our query generator first uniformly draws the number of joins $|J_q| (0 \leq |J_q| \leq 2)$ and then uniformly selects a table that is referenced by at least one table. For $|J_q| > 0$, it then uniformly selects a new table that can join with the current set of tables (initially only one), adds the corresponding join edge to the query and (overall) repeats this process $|J_q|$ times. For each base table $t$ in the query, it then uniformly draws the number of predicates $|P_{t q}| (0 \leq |P_{t q}| \leq \text{num non-key columns})$. For each predicate, it uniformly draws the predicate type ($=, <, \text{or} >$) and selects a literal (an actual value) from the corresponding column.”

Figure 2: Query featurization as sets of feature vectors.
Assumptions
Assumptions

SELECT COUNT(*) FROM title t, movie_companies mc WHERE t.id = mc.movie_id AND t.production_year > 2010 AND mc.company_id = 5

Table set \{[0101 \ldots 0], [0010 \ldots 1]\} \quad Join set \{[0010]\} \quad Predicate set \{[100001000.72], [000100100.14]\}

table id \quad samples \quad join id \quad column id \quad value \quad operator id

Figure 2: Query featurization as sets of feature vectors.
Assumptions

- Assume
  - Static column range (2010 -> 0.72); no appends
  - Static DB schema (same set of tables, cols)
  - Training data MUST cover well desired queries
  - Quality depends on ACTUAL execution on a small sample from each table, at query time
## Results

<table>
<thead>
<tr>
<th>System</th>
<th>median</th>
<th>90th</th>
<th>95th</th>
<th>99th</th>
<th>max</th>
<th>mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>PostgreSQL</td>
<td>7.93</td>
<td>164</td>
<td>1104</td>
<td>2912</td>
<td>3477</td>
<td>174</td>
</tr>
<tr>
<td>Random Samp.</td>
<td>11.5</td>
<td>198</td>
<td>4073</td>
<td>2274</td>
<td>23992</td>
<td>1046</td>
</tr>
<tr>
<td>IB Join Samp.</td>
<td><strong>1.59</strong></td>
<td>150</td>
<td>3198</td>
<td>14309</td>
<td>15775</td>
<td>590</td>
</tr>
<tr>
<td>MSCN</td>
<td>3.82</td>
<td><strong>78.4</strong></td>
<td>362</td>
<td>927</td>
<td>110</td>
<td><strong>57.9</strong></td>
</tr>
</tbody>
</table>

Table 4: Estimation errors on the JOB-light workload.
Results

Up to 4 joins (5 tables):

3x better than Postgres @max and @mean

<table>
<thead>
<tr>
<th></th>
<th>median</th>
<th>90th</th>
<th>95th</th>
<th>99th</th>
<th>max</th>
<th>mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>PostgreSQL</td>
<td>7.93</td>
<td>164</td>
<td>1104</td>
<td>2912</td>
<td>3477</td>
<td>174</td>
</tr>
<tr>
<td>Random Samp.</td>
<td>11.5</td>
<td>198</td>
<td>4073</td>
<td>22748</td>
<td>23992</td>
<td>1046</td>
</tr>
<tr>
<td>IB Join Samp.</td>
<td>1.59</td>
<td>150</td>
<td>3198</td>
<td>14309</td>
<td>15775</td>
<td>590</td>
</tr>
<tr>
<td>MSCN</td>
<td>3.82</td>
<td>78.4</td>
<td>362</td>
<td>927</td>
<td>1110</td>
<td>57.9</td>
</tr>
</tbody>
</table>

Table 4: Estimation errors on the JOB-light workload.
What is actually learned?
What is actually learned?

Reduction(join) = 1 / max {
    Cardinality("emp where emp.pos = Mgr1"),
    Cardinality("sal where sal.sal > 100K")
}
What is actually learned?

- My interpretation
  - It learns a *dampened* version of this formula per column/predicate combination
  - This “solves” correlation
  - Deep nets are great at capturing patterns

\[
\text{Reduction(join)} = \frac{1}{\max \{ \text{Cardinality(“emp where emp.pos = Mgr1”),} \\
\text{Cardinality(“sal where sal.sal > 100K”)}} \}
\]
Discussion

• Vision and Control - is it useful to have “vision” in understanding databases’ data?

• Tree/graph neural nets needed (or even helpful) here?

• Do learning solutions have a place for “easy” cases? (How to afford data/training/operational costs?)

Levine et al., Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection