Machine Learning for Databases

Guoliang Li
Tsinghua

Xuanhe Zhou
Tsinghua

Lei Cao
MIT
Motivation of ML4DB

- Machine Learning gets more practical. And empirical databases meet bottlenecks.

- Various ML models are available. It is challenging to select proper ML models.

- Rigorous requirements for ML in databases, e.g., performance, robustness, interpretable.
Database Problems

- Database Core
  - Query Rewrite
  - Cost/Cardinality Estimation
  - Join Order Selection
- Database Configuration
  - Index/View Advisor
  - Knob Tuning
  - Workload Prediction

Machine Learning for Databases

- Database Configuration
  - Knob Tuner
  - View Advisor
  - Index Advisor
  - Database Partition
- Query Optimization
  - Cardinality Estimator
  - Cost Estimator
  - Plan Enumerator
  - End-to-End Learned Optimizer
- Database Design
  - Learned Indexes
  - Learned KV Storage
  - Transaction Prediction
  - Transaction Scheduling
- Database Diagnosis
  - Performance Prediction
  - System Diagnosis
- Database Security
  - Data Discovery
  - Access Control
  - SQL Injection
- Autonomous Databases Systems
  - Paloton
  - SegaDB
  - openGauss
  - ...
## Overview of ML4DB

<table>
<thead>
<tr>
<th>Problem</th>
<th>Description</th>
<th>DB Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Offline NP Optimization</strong></td>
<td>Optimize an NP-hard problem with large search space</td>
<td>Knob Tuning</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Index/View Selection</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Partition-key Selection</td>
</tr>
<tr>
<td><strong>Online NP Optimization</strong></td>
<td>Optimize an NP-hard problem with large search space (instant feedback)</td>
<td>Query rewrite</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Plan Enumeration</td>
</tr>
<tr>
<td><strong>Regression</strong></td>
<td>Determine the relationship between one dependent variable and a series of other independent variables</td>
<td>Cost/Cardinality Estimation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Index/View Benefit Estimation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Latency Estimation</td>
</tr>
<tr>
<td><strong>Prediction</strong></td>
<td>Forecast the likelihood of a particular outcome</td>
<td>Trend Forecast</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Workload Prediction &amp; Scheduling</td>
</tr>
</tbody>
</table>
## Overview of NP-hard Problems

<table>
<thead>
<tr>
<th>Offline Optimization (knob tuning, view selection, index selection, partition-key selection)</th>
<th>Method</th>
<th>Strategy</th>
<th>Search Space</th>
<th>Training Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gradient based</td>
<td>Local search</td>
<td>Small</td>
<td></td>
<td>Huge</td>
</tr>
<tr>
<td>Deep Learning (DL)</td>
<td>Continuous space</td>
<td>Large</td>
<td></td>
<td>Huge</td>
</tr>
<tr>
<td>Meta Learning</td>
<td>Share common model weights</td>
<td>Various spaces</td>
<td></td>
<td>Huge</td>
</tr>
<tr>
<td>Reinforcement Learning (RL)</td>
<td>Multi-step search</td>
<td>Large</td>
<td></td>
<td>--</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Online Optimization (query rewrite, plan enumeration)</th>
<th>Method</th>
<th>Strategy</th>
<th>Search Space</th>
<th>Training Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCTS (Monte Carlo Tree Search) + DL</td>
<td>Multi-step search</td>
<td>Large</td>
<td></td>
<td>Huge</td>
</tr>
<tr>
<td>Multi-armed</td>
<td>Multi-step search</td>
<td>Small</td>
<td></td>
<td>Small</td>
</tr>
</tbody>
</table>
# Overview of Regression Problems

<table>
<thead>
<tr>
<th>Method</th>
<th>Task</th>
<th>Feature Space</th>
<th>Feature Type</th>
<th>Training Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classic ML (e.g., tree-ensemble, gaussian, autoregressive)</td>
<td>cost estimation, view/index benefit estimation</td>
<td>Small</td>
<td>Continuous</td>
<td>Huge</td>
</tr>
<tr>
<td>Sum-Product Network</td>
<td>cost estimation</td>
<td>Small</td>
<td>Discrete</td>
<td>Small</td>
</tr>
<tr>
<td>Deep Learning</td>
<td>cost estimation, benefit estimation, latency estimation</td>
<td>Large</td>
<td>Continuous</td>
<td>Huge</td>
</tr>
<tr>
<td>Graph Embedding</td>
<td>benefit estimation, latency estimation</td>
<td>Large</td>
<td>Continuous</td>
<td>Huge</td>
</tr>
</tbody>
</table>
## Overview of Prediction Problems

<table>
<thead>
<tr>
<th>Method</th>
<th>Task</th>
<th>Target</th>
<th>Training Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clustering Algorithm</td>
<td>Trend Forecast</td>
<td>High accuracy</td>
<td>Huge</td>
</tr>
<tr>
<td></td>
<td>Workload Scheduling</td>
<td>High performance</td>
<td>--</td>
</tr>
</tbody>
</table>

---

7

VLDB’21 Tutorial
Optimizing NP-hard Problems

- Offline Optimization vs Online Optimization
  - Model Selection
    - E.g., Offline is model-free and online is model-based
  - Overhead
    - Online requires instant feedback and offline is insensitive
  - Performance
    - Generally offline has better performance
Offline Optimizing NP-hard Problems
Offline Optimization for Knob Tuning

- **Motivation:**
  - DBMSs have different optimal knob settings, which significantly affect the query performance and resource utilization.
  - DBMSs have numerous runtime metrics. Classic ML models cannot efficiently select knobs based on all the metrics.
  - DBMSs have numerous system knobs to choose from, which makes it harder to find optimal knobs.
Offline Optimization for Knob Tuning

Problem Definition

Consider a database with different workloads, the target is to find the optimal knob settings, i.e., satisfying SLA or resource requirements under several constraints (e.g., over 5% throughput improvement).
Offline Optimization for Knob Tuning

☐ Existing Works

(1) Gradient-based Methods

[Dana et al. SIGMOD 2017], [Kunjir et al. SIGMOD 2020]
[Cereda et al. VLDB 2021]

(2) Deep Learning Method [Tan et al. VLDB 2019]

(3) Meta Learning Method [Zhang et al. SIGMOD 2021]

(4) Deep Reinforcement Learning Methods

[Zhang et al. SIGMOD 2019], [Li et al. VLDB 2019]
Feature Extraction

- Characterize Workload Behaviors
- Extract and Prune Runtime Metrics (e.g., #-page-read, #-page-write)
- Identify Important Knobs
- Estimate the knob correlations by minimizing the square errors

Model Construction

- Search Optimal Knobs based on the Runtime Metrics
- Gaussian Process: (1) Approximate the knob-performance relations with numerous historical data; (2) Recommend knobs based on the most similar historical workload

(1) Gradient-based Method for Spark Tuning

Feature Extraction

- Spark tuning considers knobs at different levels →
- Empirically estimate execution profiles at resource/APP/VM levels

E.g., Memory Efficiency:

\[ q_2^x = \frac{M_i + m_c}{\min(m_o^x, m_c^x)} \]

- Tested knob setting \( x \)
- \( M_i \): Code overhead value
- \( m_c \): Required cach storage
- \( m_o \): GC settings

Model Construction

- Gaussian Process is black-box and requires much training data →
- Guided Gaussian Process: (1) Enhance tuning with the estimated execution profiles as inputs; (2) Use GP to fit existing tuning data
(2) Deep Learning for Buffer Tuning

- **Feature Extraction**
  - Buffer Pool is critical resource in cloud databases
  - Only tune the buffer_pool_size knob for higher resource utilization
  - Many Metrics affect the response time besides buffer_pool_size
  - Database metrics: logical-read, io-read, QPS, CPU usage, historical RT

- **Model Construction**
  - Tune buffer size that maximizes resource utilization under SLAs
  - (1) Select buffer sizes for databases with similar database metrics; (2) Design a neural network to estimate SLA as tuning feedback

J. Tan, T. Zhang, F. Li, et al. iBTune: Individualized Buffer Tuning for Large-Scale Cloud Databases. VLDB’21 Tutorial
(3) Meta Learning for Knob Tuning

Feature Extraction

- Characterize the common features of workloads
- Meta-Features: Reserved words in the SQLs
- Cluster historical workloads (random forest) and learn a base learner (meta-features as inputs) for each workload cluster

Model Construction

- Boost tuning for new instances
  - (1) Learn meta-learner based on the weighted sum of the base learners;
  - (2) Fine-tune the meta-learner on the new instance;
  - (3) Recommend promising knobs

(4) Reinforcement-learning for Knob Tuning

- Challenge:
  - Basic ML models tune a small part of knobs. It is challenging to support more knobs with complex correlations.
  - High-quality training samples are hard to obtain, especially in real-world scenarios.
(4) Reinforcement-learning for Knob Tuning

Feature Extraction

- Map knob tuning into an RL problem

<table>
<thead>
<tr>
<th>RL</th>
<th>CDBTune</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agent</td>
<td>The tuning system</td>
</tr>
<tr>
<td>Environment</td>
<td>DB instance</td>
</tr>
<tr>
<td>State</td>
<td>Internal metrics</td>
</tr>
<tr>
<td>Reward</td>
<td>Performance change</td>
</tr>
<tr>
<td>Action</td>
<td>Knob configuration</td>
</tr>
<tr>
<td>Policy</td>
<td>Deep neural network</td>
</tr>
</tbody>
</table>

Throughput Latency SLAs

<Agent>
CDBTune

Network

Knobs

<Policy>

Action

<Environment>

CDB

effective_cache_size
checkpoint_timeout
io_concurrency

<State>
Metrics

xact_commit
blk_reads/hit
tuple_fetched
conflicts

(4) Reinforcement-learning for Knob Tuning

☐ Model Construction

• Many Continuous system metrics and knobs

  • Value-based method (DQN)  
    - Replace the Q-table with a neural network  
    - Input: state metrics; Output: Q-values for all the actions

  • Policy-based method (DDPG)  
    - (actor) Parameterized policy function: \( a_t = \mu(s_t | \theta^\mu) \)  
    - (critic) Score specific action and state: \( Q(s_t, a_t | \theta^Q) \)
# Summarization of Knob Tuning

<table>
<thead>
<tr>
<th>Optimization Target</th>
<th>Loss/Reward Function</th>
<th>Training Data</th>
<th>Adaptive (workload)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gradient-based</td>
<td>The weighting coefficients are equal to the mean estimates of the target values</td>
<td>High</td>
<td>--</td>
</tr>
<tr>
<td>[SIGMOD 2017]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[SIGMOD 2020]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deep Learning</td>
<td>$L : (e, \lambda) \rightarrow \begin{cases} l(e)I(e \geq 0) \ \lambda l(e)I(e &lt; 0) \end{cases}$</td>
<td>High</td>
<td>--</td>
</tr>
<tr>
<td>[VLDB 2019]</td>
<td>$l(e)$: mean square error; $\lambda$: Control the impact of overestimating</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Meta Learning</td>
<td>The loss is the number of misranked pairs the model predicted</td>
<td>High</td>
<td>✓</td>
</tr>
<tr>
<td>[SIGMOD 2021]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deep Reinforcement</td>
<td>$r = \begin{cases} ((1 + \Delta_{t\rightarrow 0})^2 - 1)[1 + \Delta_{t\rightarrow t-1}, \Delta_{t\rightarrow 0} &gt; 0 \ -(1 - \Delta_{t\rightarrow 0})^2 - 1)[1 - \Delta_{t\rightarrow t-1}, \Delta_{t\rightarrow 0} \leq 0 \end{cases}$</td>
<td>Low</td>
<td>✓ (Pre-train a query model)</td>
</tr>
<tr>
<td>Learning</td>
<td>$r$: the reward; $\Delta T_{t\rightarrow t-1}/\Delta T_{t\rightarrow 0}$: the performance change</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[SIGMOD 2019]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[VLDB 2019]</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Take-aways of Knob Tuning

- **Gradient-based method** reduces the tuning complexity by filtering out unimportant features. However, different scenarios may have different key features, which makes it hard to train a generalizable tuning model.

- **Deep learning method** considers both query performance and resource utilization. And they work better for resource-sensitive scenarios.

- **Reinforcement learning methods** take longest training time, e.g., hours, from scratch. However, it only takes minutes to tune the database after well trained and gains relatively good performance.

- **Learning based methods** may recommend bad settings when migrated to a new workload. Hence, it is vital to validate the tuning performance.

- **Open problems:**
  - Predict workload execution performance for knob tuning
  - One tuning model fits multiple databases
  - Utilize empirical knowledge
Materialized Views (MVs) optimize queries

- Share common subqueries

Space-for-time trade-off principle

- Materialize hot data (MVs) within limited space
- How to estimate the MV utilities

The number of potential MVs grows exponentially

- Greedy/Genetic/other-heuristics work bad
Offline Optimization for View Selection

Problem Definition

Given a workload $Q$ and a space budget, select optimal subqueries to materialize (MV$\text{Vs}$), including (i) candidate MV generation; (ii) MV Selection.
Offline Optimization for View Selection

- Two sub-problems
  - Benefit estimation
    - Estimate the benefit of materializing a view
      - $\text{Cost}(q) - \text{Cost}(q,v)$, $q$ is a query and $v$ is a view
  - View selection
    - Select views from a large number of possible combinations to maximize the benefit within a budget
Offline Optimization for View Selection

- Existing Works

  (1) Hybrid View Selection
      [Ahmed et al. VLDB 2020]

  (2) DRL for View Selection
      [Yuan et al. ICDE 2020]

  (3) Encoder-Decoder for View Benefit Estimation
      [Han et al. ICDE 2021]
Feature Extraction

- Numerous common subqueries among workload queries
- Cluster equivalent queries and select the least overhead ones as the candidate;

Model Construction

- Numerous combinations of candidate subqueries
- (1) Solve MV Selection with Q-learning: (2) Estimate the MV utility with a deep neural network

\[
Z = \{z_j\} : z_j \text{ is a 0/1 variable indicating whether to materialize the subquery } s_j \\
Y = \{y_{ij}\} : y_{ij} \text{ is a 0/1 variable indicating whether to use the view } v_{s_j} \text{ for the query } q_i
\]

Indexes are essential for efficient execution

- SELECT c_discount from bmsql_customer where c_w_id = 10;
- CREATE INDEX on bmsql_customer(c_w_id);

Select from numerous indexable columns

- Columns have different access frequencies, data distribution

Redundant indexes may cause negative effects

- Increase maintenance costs for update/delete operations
Offline Optimization for Index Selection

Problem Definition

Given a workload and constraints (e.g., disk limit), find an index set, such that the performance is optimal with the constraint.
Offline Optimization for Index Selection

- Two sub-problems
  - Benefit estimation
    - Estimate the benefit of creating an index
      - Cost(q) - Cost(q, index), q is a query
  - Index selection
    - Select indexes from a large number of possible combinations to maximize the benefit within a budget
DRL for Index Selection

- Feature Extraction
  - Map candidate indexes with empirical rules

- Model Construction
  - Map the index selection problem into a reinforcement learning model
    - State: Information of current built indexes
    - Action: Choose an index to build
    - Reward: Cost reduction ratio after building the index

Rule 1: Construct all single-attribute indexes by using the attributes in J, EQ, RANGE.
Rule 2: When the attributes in 0 come from the same table, generate the index by using all attributes in 0.
Rule 3: If table a joins table b with multiple attributes, construct indexes by using all join attributes.

Take-aways of View/Index Selection

- Learned selection is more robust than heuristics
- Learned selection works well in online service, but takes much time for model training (cold start)
- Query encoding models need to be trained periodically when data update
- Open problems:
  - Benefit prediction for future workload
  - Cost for future updates
  - Support updates/eviction
Motivation:

- A vital component in distributed database
  - Place partitions on different nodes to speedup queries
  - Trade-off between data balance & access frequency

- Database partition problem is NP-hard
  - Combinatorial problem: 61 TPC-H columns, 145 query relations, $2.3 \times 10^{18}$ candidate combinations
Offline Optimization for Database Partition

Problem Definition

<table>
<thead>
<tr>
<th>customer</th>
<th>c_custkey</th>
<th>c_nationkey</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>lineitem</th>
<th>l_orderkey</th>
<th>l_suppkey</th>
</tr>
</thead>
<tbody>
<tr>
<td>197</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>69</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>161</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>15</td>
<td></td>
</tr>
</tbody>
</table>

Distribute by HASH (c_custkey)

Node 0

<table>
<thead>
<tr>
<th>customer_p_00</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 13</td>
</tr>
<tr>
<td>4 4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>lineitem_p_00</th>
</tr>
</thead>
<tbody>
<tr>
<td>64 15</td>
</tr>
</tbody>
</table>

Node 1

<table>
<thead>
<tr>
<th>customer_p_01</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 15</td>
</tr>
<tr>
<td>3 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>lineitem_p_01</th>
</tr>
</thead>
<tbody>
<tr>
<td>197 15</td>
</tr>
<tr>
<td>69 13</td>
</tr>
<tr>
<td>161 1</td>
</tr>
</tbody>
</table>

SELECT * FROM customer, lineitem WHERE c_nationkey = l_suppkey and c_nationkey < 4;
Heuristic Method for Offline Co-Partition

- Select from foreign-key relations between tables
  - **↑** Data-locality: Maximum spanning tree for each query
    - **Schema Graph** $G_s$ (with weights):
      - $O$ to $L$: 1.5m
      - $C$ to $S$: 150k
      - $N$: 25
    - **Maximum Spanning Tree (MAST):**
      - $O$ to $L$: 1.5m
      - $C$ to $S$: 150k
      - $N$: 25
    - **Partitioning Configuration:**
      - $O$ to $L$: SP
      - $C$ to $S$: PREF on $O$
      - $N$: PREF on $C$

- **↓** Data-redundancy: Enumerate selected partitions with DP
  - **Maximum Spanning Trees (MAST):** $Q_1, Q_2, Q_3, Q_4$
    - $Q_1$ to $Q_{4+2}$: 150k
    - $Q_2$ to $Q_{4+2}$: 1.5m
    - $Q_3$ to $Q_{4+2}$: 10k
    - $Q_4$ to $Q_{4+2}$: 25
  - **Merged MASTs (First Phase):**
    - $Q_3$ to $Q_{4+2}$: SP
    - $Q_4$ to $Q_{4+2}$: PREF on $N$
  - **Merged MASTs (Second Phase):**
    - $Q_3$ to $Q_{4+4}$: SP
    - $Q_4$ to $Q_{4+4}$: PREF on $N$

---

Hybrid Methods for Partition-Key Selection

- Combine exact and heuristic algorithms to find good partition strategies
  - The partitioning performance is affected by the join queries
  - Build a weighted undirected graph, where the nodes are tables and edges are join relations.
  - Key Selection on the graph is a maximum weight matching problem
  - Provide both exact (i.e., each table uses a column, and turn into a integer programming problem) and heuristic (i.e., select the table columns whose edge weights are maximal) algorithms; and apply the appropriate algorithm under the time budget.
DRL for Offline Partition-Key Selection

- **Feature Extraction**
  - Typical OLAP Workloads contain complex and recursive queries →
  - State Features: [ tables, query frequencies, foreign keys ]

- **Model Construction**
  - To select from numerous partition-key combinations and support new queries →
  - (1) Use DQN to partition or replicate tables; (2) Pre-train a cluster of RL models.

Takeaways of Database Partition

- Learned key-selection partition outperforms heuristic partition
- Learned key-selection partition has much higher partition latency for model training

Open Problems:
- Adaptive partition for relational databases
- Partition quality prediction
- Improve partition availability with replicates
Online Optimizing NP-hard Problems
Motivation:

- Many queries are poorly-written
  - Terrible operations (e.g., subqueries/joins, union/union all);
  - Looks pretty to humans, but physically inefficient (e.g., take subqueries as temporary tables);

- Existing methods are based on heuristic rules
  - Top-down order may not be optimal (e.g., remove aggregates before pulling up subqueries)
  - No evaluation of different rewrite orders

Trade-off in SQL Rewrite

- Best Performance: Enumerate for the best rewrite order
- Minimal Latency: SQL Rewrite requires low overhead (milliseconds)
Online Optimization for Query Rewrite

Problem Definition

Given a slow query and a set of rewrite rules, apply the rewrite rules to the query so as to gain the equivalent one with the minimal cost.
Online Optimization for Query Rewrite

Challenge:

- The rewrite space is large
  - Exponential to the number of rewrite rules

- Search rewrite space within time constraints
  - Rewrite within milliseconds;

- Estimate rewrite benefits by multiple factors
  - Reduced costs after rewriting
  - Future cost reduction if further rewriting the query
MCTS for Query Rewrite

Feature Extraction

- A slow query may have various rewrite of different benefits
- Policy Tree Model
  - Node $v_i$: any rewritten query
  - $C^\uparrow(v_i)$: previous cost reduction
  - $C^\downarrow(v_i)$: subsequent cost reduction

Model Construction

- To select from numerous rewrite orders
- (1) Policy Tree Search Algorithm
  $$U(v_i) = (C^\uparrow(v_i) + C^\downarrow(v_i)) + \gamma \sqrt{\frac{\ln(F(v_0))}{F(v_i)}}$$
- (2) Multiple Node Selection

Policy Tree

Node Selection

MCTS for Query Rewrite

Node $v_i$ represents a rewritten query.

0.5

0.2

0.4

0.2

0.1

maximize utility

$C^\uparrow = 0.3$

$C^\downarrow = 0.1$

To optimal direction
Take-aways of Query Rewrite

- Traditional query rewrite method is unaware of cost, causing redundant or even negative rewrites
- Search-based rewrite works better than traditional rewrite for complex queries
- Rewrite benefit estimation improves the performance of simple search based rewrite

Open Problems
- Balance Rewrite Latency & Performance
- Adapt to different rule sets/datasets
- Design new rewrite rules
Plan Enumerator

- **Motivation:**
  - Planning cost is hard to estimate
    - The plan space is huge
  - Traditional optimizers have some limitations
    - DP gains high optimization performance, but causes great latency;
    - Random picking has poor optimization ability
  - Steer existing optimizers can gain higher performance
    - Hint join orders; Hint operator types
Join Order Enumerator

Problem Definition

Given a SQL query, select the “cheapest” join ordering (according to the cost model).
Join Order Enumerator

Method Classification

- Offline Optimization Methods.
  - Characteristic: given Workload, RL based.
  
  - Key idea: Use existing workload to train a learned optimizer, which predicts the plan for future queries.

- Online Optimization Methods.
  - Characteristic: No workload, but rely on customized Database.
  
  - Key idea: The plan of a query can be changed during execution. The query can switch to another better plan. It learns when the database executes the query.
Offline Optimization for Join Order Enumerator

- Map into RL Models (DQ, ReJOIN) [1,2]
  - Agent: optimizer
  - Action: join
  - Environment: Cost model, database
  - Reward: Cost, Latency
  - State: join order

1. Marcus R, Papaemmanouil O. Deep reinforcement learning for join order enumeration
Feature Encoding for Join Order Enumerator

Feature Extraction

- The structural information of the execution plan is vital to join order selection.
- Encode the operator relations and metadata features of the query.
- Embed the query features with Tree-LSTM; Decide join orders with RL model.

Query q:
Select *
From T1, T2, T3, T4
Where T1.h > 30
and T1.t < 50
and T1.a = T2.a
and T3.b = T3.b
and T1.c = T1.c

(A) Query Representation for input query

(B) Table and column representation

(C) Join tree and join state representation

VLDB’21 Tutorial

Online Optimization for Join Order Enumerator

- Update execution orders of tuples on the fly
  - Update the plan on the fly and preserve the execution state
  - Tuples flows into the Eddy from input relations (e.g., R, S, T);
  - Eddy routes tuples to corresponding operators (the order is adaptively selected by the operator costs);
  - Eddy sends tuples to the output only when the tuples have been handled by all the operators.


VLDB’21 Tutorial
MCTS for Join Order Enumerator

- Support online reorder with MCTS
  - Do not require pre-training
  - Time Slides: 0.001s
    - Learn during runtime
  - Customize Database
    - Switch Plan in Low Latency

Monte Carlo tree search (MCTS).

# Join Order Enumerator

<table>
<thead>
<tr>
<th></th>
<th>Quality</th>
<th>Training Cost</th>
<th>Adaptive (workload)</th>
<th>Adaptive (DB Instance)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Traditional Methods</strong></td>
<td>Low</td>
<td>Low</td>
<td>✓</td>
<td>High</td>
</tr>
<tr>
<td>[Genetic algorithms]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[Dynamic Programming]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Offline Optimization Methods</strong></td>
<td>High</td>
<td>High</td>
<td>×</td>
<td>Medium</td>
</tr>
<tr>
<td>[NEO VLDB2019]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[RTOS ICDE2020]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Online Optimization Methods</strong></td>
<td>Medium</td>
<td>Low</td>
<td>✓</td>
<td>Low</td>
</tr>
<tr>
<td>[Eddies SIGMOD2000]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[SkinnerDB SIGMOD2019]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Online Optimization for Plan Hinter

- Enhance query optimization with minor changes
  - E.g., Activate/Deactivate loop join for different queries

- Model Plan Hinter as a Multi-armed Bandit Problem
  - Model each hint set $HSet_i$ as a query optimizer
    \[ HSet_i : Q \rightarrow T \]
  - For a query $q$, it aims to generate optimal plan by selecting proper hint sets, which is dealt as a regret minimization problem:
    \[ R_q = \left( P(B(q)(q)) - \min_{i} P(HSet_i(q)) \right)^2 \]
Take-aways of Plan Enumerator

- Learning based algorithm usually gives the plan with low time complexity, especially for large queries.
- Offline learning methods use the sampled workload to pretrained the model. It will give good plans for the incoming queries.
- A new database (updates) will lead to model retraining.
- Online-learning methods do not need previous workload and can give good plans. But it needs the *customized engine* and is hard to be applied in existing databases.

Open Problems

- Raise the generalization performance of offline learning methods for unseen queries.
- Ensure the plan given by learned model is robust (explicable).
- Speed up the model training time, e.g. transferring previous knowledge.
- Make the model aware of the data update.
Regression Problems
Regression Problems

**Database estimation problems** can be modeled as regression problems, which fit the high-dimension input variables into target features (e.g., cost, utility) and estimate the value of another variable.

- **Cardinality/Cost Estimation** aims to estimate the cardinality of a query and a regression model (e.g., deep learning model) can be used.

- **Index/View Benefit Estimation** aims to estimate the benefit of creating an index (or a view), and a regression model can be used to estimate the benefit.

- **Query Latency Estimation** aims to estimate the execution time of a query and a regression model can be used to estimate the performance based on query and concurrency features.
Automatic Cardinality/Cost Estimation

- **Motivation:**
  - One of the most challenging problems in databases
    - Achilles Heel of modern query optimizers
  - Traditional methods for cardinality estimation
    - Sampling (on base tables or joins)
    - Kernel-based Methods (Gaussian Model on Samples)
    - Histogram (on single column or multiple columns)
  - Traditional cost models
    - Data sketching/data histogram based methods
    - Sampling based methods
Categories of Cardinality Estimation

(1) Supervised Query Methods
- Multi-set Convolutional network
- Tree-based ensemble

(2) Supervised Data Methods
- Gaussian kernel
- Uniform mixture model

(3) Unsupervised Data Methods
- Autoregressive
- Sum product network
1 Supervised Query Methods for Cardinality Estimation

[Diagram: Supervised Model Training and Online Cardinality Estimation]

Problem Definition

A regression problem: learn the mapping function between query \( Q \) and its actual cardinality.

VLDB’21 Tutorial
1.1 Deep Learning for Cardinality Estimation

- Model Construction
  - Multi-set Convolutional Neural Network
    - Linear Models for different part of SQL (table, joins, predicates)
    - Pooling Varying-sized representations (avg pooling)
    - Concatenate different parts
1.2 Tree-Ensembling for Cardinality Estimation

- **Model Construction**
  - **Challenge:** Traditional cost estimation methods assume column independency
  - **Any conjunctive query on columns C can be represented as:**
    \[(c_1 \leq lb_1 < c_2) \land (c_3 < ub_1 \leq c_4) \land (c_5 \leq ub_2 \leq c_6)\]
  - **Tree-based ensembles:** pass query encoding vectors through the traversal of multiple binary trees

2 Supervised Data Methods for Cardinality Estimation

Problem Definition

A density estimation problem: learn a joint data distribution of each data point
2.1 Kernel-Density for Cardinality Estimation

- **Model Construction**
  - Support point queries on single tables

  - Sample rows from the table and initialize the **bandwidth** (distance from the true distribution) of the kernel density model.

  - Pick optimal bandwidth via stochastic gradient descent.

  - Estimate the cardinality based on the kernel density model.

---

2.2 Mixture Model for Cardinality Estimation

- Model Construction
  - Support Range Queries
    - Sample points within each history queries.
    - Generating subgroups for the points.
    - Learn the weights of all the Uniformity Mixture Models for range queries.

(a) Case 1: Highly-overlapping query workloads

Yongjoo Park, Shucheng Zhong, and Barzan Mozafari. Quicksel: Quick selectivity learning with mixture models. SIGMOD 2020
3 Unsupervised Data Methods for Cardinality Estimation

Problem Definition

A regression problem: learn a probability function for each data point

Diagram:
- **Data Sampler** -> **Data Model Training** -> **Well-Trained Data Model** -> **Optimizer**
  - **Query Dataset** -> **Sampled Tuples**
3.1 Autoregressive for Cardinality Estimation (single table)

- **Model Construction**
  - Learn the joint probability distribution over columns for range queries
    - Use Autoregressive Model to fit the joint probability of different columns
    - Support range query with Progressive Sampling

---

3.2 Autoregressive for Cardinality Estimation (multi-tables)

- **Model Construction**
  - Deep AR models can only handle single tables, and we need to learn from join correlations.
    - Learn a single autoregressive model for all the tables (joined).
    - Join Sampler provides correct training data (sampled tuples from join) by using unbiased join counts.
    - Down sampling some tuples when estimating query with only a subset of tables according to the fanout scaling.
3.3 Sum-Product Network for Cardinality Estimation

Model Construction

- Different data distributions over the tables, which are independent from each other

  - Split data table into multiple segments and columns in each segment are near independent.

  - SPN: Sum for different filters and Product for different joins.

  - RSPN is for AVG aggregation, NULL values support, non-key attributes modeling and updatability.

---

<table>
<thead>
<tr>
<th>c.id</th>
<th>c.age</th>
<th>c.region</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>80</td>
<td>EU</td>
</tr>
<tr>
<td>2</td>
<td>70</td>
<td>EU</td>
</tr>
<tr>
<td>3</td>
<td>60</td>
<td>ASIA</td>
</tr>
<tr>
<td>4</td>
<td>20</td>
<td>EU</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>998</td>
<td>20</td>
<td>ASIA</td>
</tr>
<tr>
<td>998</td>
<td>25</td>
<td>EU</td>
</tr>
<tr>
<td>999</td>
<td>30</td>
<td>ASIA</td>
</tr>
<tr>
<td>1000</td>
<td>70</td>
<td>ASIA</td>
</tr>
</tbody>
</table>

(a) Example Table

(b) Learning with Row/Column Clustering

(c) Resulting SPN

(d) Probability of European Customers younger than 30
The Relations of Card/Cost Estimation

- **Task Target**
  - Cost estimation is to approximate the execution-time/resource consumption;

- **Correlations**
  - Cost estimation is based on cardinality

- **Estimation Difficulty**
  - Cost is harder to estimate than cardinality, which considers multiple factors (e.g., seq scan cost, cpu usage)
Model Construction

- Traditional cost estimation uses estimated card, which is inaccurate without predicate encoding.
Tree-LSTM for Cost Estimation

- Model Construction
  - The representation layer learns an embedding of each subquery (global vector denotes the subquery, local vector denotes the root operator)
  - The estimation layer outputs cardinality & cost simultaneously

SQL Query

Model Construction

Parameter Sharing

Representation Model

Predicates Embedding Layer

Concatenate

Estimation Layer

Sigmoid

Linear

ReLU

Linear
Take-aways

- Data-driven methods are more effective for single tables.
- Query-driven methods are more effective for multiple tables.
- Query-driven methods are more efficient than Data-drive methods.
- Data-driven methods are more robust than Query-driven methods.
- Training queries are vital to Query-driven methods.
- Samples are crucial to Data-driven methods.
- Estimators based on neural network are more accurate than statistic-based estimators.
- Statistic-based query model is the most efficient.
Challenge

- The index/view benefit is hard to evaluate
  - Multiple evaluation metrics (e.g., index benefit, space cost)
  - Cost estimation by the optimizer is inaccurate

- Interactions between existing data structures
  - Multiple column access, Data refresh
  - Conflicts between MVs
1 Deep Learning for Index Benefit Estimation

- **Model Construction**

  - **Motivation:** It is critical to compare execution costs of plans and decide index benefits →
  
  - **Training Data:** Workloads and execution feedback from customers
  - **Well-trained Evaluation Model:** Predict the index performance
  - Use the evaluation model to create indexes with performance gains

Feature Extraction

- Previous work take candidate views as fixed length →
- Encode various number and length of queries and views with an encoder-reducer model, which captures correlations with attention

Model Construction

- It is hard to jointly consider MVs that may have conflicts →
- (1) Split the problem into sub-steps that select one MV; (2) Use attention-based model to estimate the MV benefit

Take-aways of Benefit Estimation

- Learned utility estimation is more accurate than traditional empirical methods.
- Learned utility estimation is also accurate for multiple-MV optimization.
- Query encoding models need to be trained periodically when data update.
- Open problems:
  - Benefit prediction for future workload
  - Cost of initialization and future updates
Deep Learning for Query Latency Estimation

Model Construction

- Performance prediction of single queries
  - Represent each operator with a neural unit
  - Each neural unit predicts the execution time of its operator
  - Construct a network that matches the query structure to predict the query latency
  - Take effects of concurrent queries as parallel operators (e.g., gather, parallel join)

Graph Embedding for Query Latency Estimation

Model Construction

- **Performance prediction of concurrent queries**
  - Represent concurrent queries with a graph model
  - Embed the graph with graph convolution network and predict the latency of all the operators with a simple dense network
Prediction Problems
Prediction Problems

- Motivation
  - Effective Scheduling can Improve the Performance
    - Minimize conflicts between transactions
  - Concurrency Control is Challenging
    - #-CPU Cores Increase
  - Transaction Management Tasks
    - Transaction Prediction
    - Transaction Scheduling
Learned Transaction Prediction

- Predict the future trend of different workloads
  - **Pre-Processor** identifies query templates and the arrival-rate from the workload;
  - **Clusterer** combines templates with similar arrival rate patterns
  - **Forecaster** utilizes ML models to predict arrival rate in each cluster

Learned Transaction Scheduling

- Learn to schedule queries to minimize disk access requests
  - Collect requested data blocks (buffer hit) from the buffer pool:
  - State Features: buffer pool size, data block requests, ;
  - Schedule Queries to optimize global performance with Q-learning
ML Methods in ML4DB
# Summarization of ML4DB Techniques

<table>
<thead>
<tr>
<th>Database Problem</th>
<th>Method</th>
<th>Performance</th>
<th>Overhead</th>
<th>Training Data</th>
<th>Adaptivity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Offline NP Problem</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>knob space exploration</td>
<td>gradient-based [1, 18, 47]</td>
<td>High</td>
<td>High</td>
<td>High</td>
<td>–</td>
</tr>
<tr>
<td>dense network [37]</td>
<td>Medium</td>
<td>High/Medium</td>
<td>High</td>
<td>– / instance</td>
<td></td>
</tr>
<tr>
<td>DDPG [23, 46]</td>
<td>High</td>
<td>High</td>
<td>Low/Medium</td>
<td>query</td>
<td></td>
</tr>
<tr>
<td>index selection</td>
<td>q-learning [19]</td>
<td>–</td>
<td>High</td>
<td>Low</td>
<td>–</td>
</tr>
<tr>
<td>view selection</td>
<td>q-learning [43]</td>
<td>Medium</td>
<td>High</td>
<td>Low</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>DDQN [9]</td>
<td>High</td>
<td>High</td>
<td>Low</td>
<td>query</td>
</tr>
<tr>
<td>Online NP Problem</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>join order selection</td>
<td>q-learning [27]</td>
<td>High</td>
<td>High</td>
<td>Low</td>
<td>–</td>
</tr>
<tr>
<td>DQN [26, 42]</td>
<td>High</td>
<td>High</td>
<td>Low</td>
<td>query</td>
<td></td>
</tr>
<tr>
<td>MCTS [38]</td>
<td>Medium</td>
<td>Low</td>
<td>Low</td>
<td>instance</td>
<td></td>
</tr>
<tr>
<td>query rewrite</td>
<td>MCTS [21, 49]</td>
<td>–</td>
<td>Low</td>
<td>Low</td>
<td>query</td>
</tr>
<tr>
<td>Regression Problem</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cost estimation</td>
<td>tree-LSTM [35]</td>
<td>High</td>
<td>High</td>
<td>High</td>
<td>query</td>
</tr>
<tr>
<td></td>
<td>autoregressive [41]</td>
<td>High</td>
<td>High/Medium</td>
<td>Low</td>
<td>data</td>
</tr>
<tr>
<td></td>
<td>dense network [16]</td>
<td>High</td>
<td>High</td>
<td>High</td>
<td>query</td>
</tr>
<tr>
<td></td>
<td>sum-product [12]</td>
<td>Medium</td>
<td>High</td>
<td>Low</td>
<td>data</td>
</tr>
<tr>
<td>index benefit estimation</td>
<td>dense network [3]</td>
<td>–</td>
<td>High</td>
<td>High</td>
<td>query</td>
</tr>
<tr>
<td>view benefit estimation</td>
<td>dense network [9]</td>
<td>–</td>
<td>High</td>
<td>High</td>
<td>query</td>
</tr>
<tr>
<td>latency prediction</td>
<td>dense network [28]</td>
<td>Medium</td>
<td>High</td>
<td>High</td>
<td>query</td>
</tr>
<tr>
<td></td>
<td>graph embedding [50]</td>
<td>High</td>
<td>High</td>
<td>High</td>
<td>instance</td>
</tr>
<tr>
<td>learned index</td>
<td>dense network [3]</td>
<td>–</td>
<td>High</td>
<td>High</td>
<td>query</td>
</tr>
<tr>
<td>Prediction Problem</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>trend prediction</td>
<td>clustering-based [24]</td>
<td>–</td>
<td>Medium</td>
<td>Medium</td>
<td>instance</td>
</tr>
<tr>
<td>transaction scheduling</td>
<td>q-learning [44]</td>
<td>–</td>
<td>High</td>
<td>Low</td>
<td>query</td>
</tr>
</tbody>
</table>
Classical ML Methods

- Techniques
  - Gradient methods (e.g., GP); Regression methods (e.g., tree-ensembling, kernel-density estimation)

- Advantages
  - Lightweight; Easier to interpret than DL

- Disadvantages
  - Hard to extend to large data; Complex feature engineering

- ML4DB Applications
  - Knob Tuning; Cardinality Estimation
## Classical ML Methods

### Application Difference

<table>
<thead>
<tr>
<th>Knob Tuning</th>
<th>Feature Engineering</th>
<th>Model Selection</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Reduce the knob space with linear regression like Lasso;</td>
<td>• Gaussian Process: Search local-optimal settings within the selected knob space</td>
</tr>
<tr>
<td></td>
<td>• Reduce redundant metrics with factor analysis and clustering like k-means;</td>
<td>• Reuse the historical data by matching workloads by their metric values</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cardinality Estimation</th>
<th>Feature Engineering</th>
<th>Model Selection</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Assumptions like column independency or linear relations between columns</td>
<td>• Query-based: Define input space as conjunction of the query ranges on data columns (Tree-Ensemble)</td>
</tr>
<tr>
<td></td>
<td>• Determine supported queries like range queries</td>
<td>• Data-based: Partition data into independent regions (Sum-Product) or learn column correlations (AR)</td>
</tr>
</tbody>
</table>
Classical ML Methods

☐ How to apply to a new problem?

☐ Problem Modelling: As a regression or gradient-based optimization problems

☐ Feature Engineering: Determine the input with feature engineering techniques

☐ Model Construction: Select proper classic ML models, collect sample data, and learn the mapping relations

☐ Additional Requirements: Reuse classic ML models in limited scenarios (e.g., similar workloads)
Reinforcement Learning Methods

☐ Techniques
  • Model-based (e.g., MCTS+DL);
  • Model-free (e.g., value-based like Q-learning, policy-based like DDPG)

☐ Advantages
  • High performance on large search space; No prepared data

☐ Disadvantages
  • Long exploration time; Hard to migration to new scenarios

☐ ML4DB Applications
  • Knob Tuning, View/Index/Partition-key Selection, Optimizer, Workload Scheduling
# Reinforcement Learning Methods

<table>
<thead>
<tr>
<th>Knob Tuning</th>
<th>Input Features</th>
<th>RL Method</th>
<th>Reward Design</th>
<th>Estimation Model</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Knobs Values</td>
<td>• DDPG for both continuous state and continuous actions</td>
<td>• Performance improvements over last tuning action • Performance improvements over first tuning action</td>
<td>• Design a dense network as the estimation (critic) model</td>
</tr>
<tr>
<td></td>
<td>• Innter Metrics</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Workloads</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Reinforcement Learning Methods

<table>
<thead>
<tr>
<th>View Selection</th>
<th>Input Features</th>
<th>RL Method</th>
<th>Reward Design</th>
<th>Estimation Model</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Candidate Views</td>
<td>• DQN for continuous state and discrete actions</td>
<td>• Utility increase on creating the views</td>
<td>• Encoder-decoder for inputs; Nonlinear layers for utility estimation</td>
</tr>
<tr>
<td></td>
<td>• Built Views</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Workload</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Index Selection</td>
<td>• Candidate Indexes</td>
<td></td>
<td></td>
<td>• Design a dense network as the estimation model</td>
</tr>
<tr>
<td></td>
<td>• Built indexes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Workload</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Partiton-key Selection</td>
<td>• Columns</td>
<td></td>
<td>• Estimated costs before/after partitioning</td>
<td>• Design a dense network as the estimation model</td>
</tr>
<tr>
<td></td>
<td>• Tables</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Query templates</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Reinforcement Learning Methods

<table>
<thead>
<tr>
<th>Input Features</th>
<th>RL Method</th>
<th>Reward Design</th>
<th>Estimation Model</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Query Rewrite</strong></td>
<td>MCTS for tree search</td>
<td>Utility increase for future optimal queries</td>
<td>Multi-head attention for rules, query, data</td>
</tr>
<tr>
<td><strong>Join Order Selection</strong></td>
<td>DQN for continuous state and discrete actions</td>
<td>Saved costs</td>
<td>Design a dense network as the estimation model</td>
</tr>
<tr>
<td><strong>Plan Hinter</strong></td>
<td>Contextual Multi-armed for limited actions</td>
<td>Saved costs</td>
<td>Traditional Optimizer</td>
</tr>
</tbody>
</table>

**Query Rewrite**
- Logical Query
- Rewrite Rules
- Table Schema

**Join Order Selection**
- Physical Plan
- Candidate Joins
- Table Schema

**Plan Hinter**
- Physical Plan
- Hint Sets
How to apply to a new problem?

- Problem Modelling: Map to the 6 factors in a RL model (state, action, reward, policy, agent, environment)
- Feature Characterization: Select target-related features as the state of the RL problem
- Model Construction: Select proper RL models (e.g., MCTS, DQN, DDPG), design the networks and the reward function
- Additional Requirements: E.g., encode the query costs with Deep Learning; encode the join relations with GNN
Deep Learning Methods

- Techniques
  - Dense Layer ((non)-linear); Convolutional Layer; Graph Embedding Layer; Recurrent Layer

- Advantages
  - Approximate the high-dimension relations

- Disadvantages
  - Data-consuming

- ML4DB Applications
  - Cost Estimation; Benefit Estimation; Latency Estimation
## Deep Learning Methods

<table>
<thead>
<tr>
<th></th>
<th>Input Features</th>
<th>Feature Encoding</th>
<th>Model Design</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cost Estimation</strong></td>
<td>• Physical Plan</td>
<td>• Encode operators with LSTM</td>
<td>• Plan-structured Neural Network</td>
</tr>
<tr>
<td></td>
<td>• Optimization Actions (e.g., views, indexes)</td>
<td>• Encode actions like Encoder-Decoder for Views and linear layer for Indexes</td>
<td></td>
</tr>
<tr>
<td><strong>Benefit Estimation</strong></td>
<td>• Physical Plan</td>
<td>• Encode actions like Encoder-Decoder for Views and linear layer for Indexes</td>
<td>• Design a dense network as the estimation model</td>
</tr>
<tr>
<td></td>
<td>• Optimization Actions (e.g., views, indexes)</td>
<td>• Encode actions like Encoder-Decoder for Views and linear layer for Indexes</td>
<td></td>
</tr>
<tr>
<td><strong>Latency Estimation</strong></td>
<td>• Physical Plan</td>
<td>• Encoder query correlations with graph convolutions</td>
<td>• Design a K-layer graph embedding network for K-hop neighbors</td>
</tr>
<tr>
<td></td>
<td>• Query Relations</td>
<td>• Encoder query correlations with graph convolutions</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• DB State</td>
<td>• Encoder query correlations with graph convolutions</td>
<td></td>
</tr>
</tbody>
</table>
Deep Learning Methods

☐ How to apply to a new problem?
  ☐ Input Features: Select features that affect the estimation targets (e.g., latency, utility)
  ☐ Encoding Strategy: Encode based on the feature structures (e.g., Graph embedding for query relations)
  ☐ Model Design: Design the network structures (e.g., layers, activation functions, loss functions) based on the input embedding (e.g., fixed-length or varied-length)
Open Problems of ML4DB
Open Problem #1: Reduce Model Training Overhead

- Lightweight Model Training
  - **Featurization:** Some features are *not available* in real-word scenarios, e.g., by privacy constraints;
  - **Data Collection:** Costly to *collect data* on different datasets/databases, e.g., high collection latency, overhead;
  - **Model Migration & Application:** ML models trained on small datasets are hard to generalize to large datasets

- Possible solutions: few-show learning; from data-driven to knowledge-driven; super-large pre-trained model
Open Problem #2: Validate Learning-based Models

- Model Validation
  - Whether a model is effective?
  - Whether a model outperforms existing ones?
  - Whether a model can adapt to new scenarios?
Open Problem #3: One Model Fits Various Scenarios

- **High Adaptability**
  - **Workloads:** query operators; plan structures; underlying data access
  - **Datasets:** tables; columns; data distribution; indexes / views; data updates
  - **DB Instances:** state metrics (DB, resource utilization): hardware configurations
  - **DBMSs:** MySQL; PostgreSQL; MongoDB; Spark

- **Possible Solutions:** common knowledge extraction; meta learning
Open Problem #4: Automatic Learned Model Selection

Automatic Database Assembling

- Automatically select ML models/algorithms for different tasks
- Evaluate the overall performance

<table>
<thead>
<tr>
<th>Category</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supervised Learning</td>
<td>Linear Regression, Logistic Regression, Decision Tree, Deep Learning</td>
</tr>
<tr>
<td>Unsupervised Learning</td>
<td>K-Means Clustering, Association Rules, Reinforcement Learning</td>
</tr>
<tr>
<td>Descriptive Statistics</td>
<td>Count-Min Sketch, Data Profiling</td>
</tr>
</tbody>
</table>

Database Assembling

The Stack of ML Algorithms
Open Problem #5: Unified Database Optimization

- Arrange Multiple Database Optimization Tasks
  - Multiple Requirements: (1) Optimizer can produce good plans with not very accurate estimator; (2) Creating indexes may incur the change of optimal knobs
  - Hybrid Scheduling: Arrange different optimization tasks based on the database configuration and workload characters
  - Optimization Overhead: Achieve maximum optimization without competing resources with user processes

- Challenges: various task features; correlations between tasks; trend changes
Thanks
Traditional Methods and Problems

- **Manual-based Methods (e.g., knob tuning)**
  - It is costly and time-consuming for DBAs to optimize components

- **Heuristic Search/Equations/Rules (e.g., cost/view/index estimation)**
  - Produce sub-optimal solutions; cannot learn from historical data; fail to handle complex scenarios

- **Optimal algorithms (e.g., join order selection, view selection)**
  - Assumptions may not be satisfied in most scenarios
# ML Models for Optimization Problems

<table>
<thead>
<tr>
<th>ML Method</th>
<th>Description</th>
<th>Example</th>
<th>DB Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gradient-based Methods</td>
<td>Approximate the data distribution with gaussian functions, and select the optimal point by the guidance of gradients</td>
<td><img src="image1" alt="Gradient-based Methods Example" /></td>
<td>Knob Tuning; Cardinality Estimation</td>
</tr>
<tr>
<td>Contextual Multi-armed Bandit</td>
<td>Maximize the reward by repeatedly selecting from a fixed number of arms</td>
<td><img src="image2" alt="Contextual Multi-armed Bandit Example" /></td>
<td>Plan Hint; Knob Tuning; MV Selection; Index Selection; Database Partition; Join Order Selection; Workload Schedule</td>
</tr>
<tr>
<td>Deep Reinforcement Learning</td>
<td>Learn the selection (actor) or estimation (critic) policy with neural networks</td>
<td><img src="image3" alt="Deep Reinforcement Learning Example" /></td>
<td>Query Rewrite; Online Join Order Selection</td>
</tr>
<tr>
<td>Monte Carlo Tree Search</td>
<td>Repeated iterations of four steps (selection, expansion, simulation, back-propagation) until termination</td>
<td><img src="image4" alt="Monte Carlo Tree Search Example" /></td>
<td></td>
</tr>
<tr>
<td>ML Method</td>
<td>Description</td>
<td>Example</td>
<td>DB Tasks</td>
</tr>
<tr>
<td>-----------------------</td>
<td>--------------------------------------------------------------------------------------------------------</td>
<td>------------------------------------------------------------------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>Statistical ML</td>
<td>Build a regression model to approximate real distribution based on sampled data</td>
<td>![Graph 1]</td>
<td>Cardinality Estimation; Trend Prediction</td>
</tr>
<tr>
<td>Sum-Product Network</td>
<td>Learn distributions with <strong>Sum</strong> for different filters and <strong>Product</strong> for different joins</td>
<td>![Graph 2]</td>
<td>Cardinality Estimation</td>
</tr>
<tr>
<td>Deep Learning (e.g., DNN, CNN, RNN)</td>
<td>Learn the mapping relations from the input features to the targets by gradient descent</td>
<td>![Graph 3]</td>
<td>Knob Tuning; Cardinality Estimation; Cost Estimation</td>
</tr>
</tbody>
</table>
## ML Models for Others

<table>
<thead>
<tr>
<th>ML Method</th>
<th>Description</th>
<th>Example</th>
<th>DB Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Generative Model</strong> (e.g., Encoder-Decoder)</td>
<td>Encode varied-length input features into fixed-length vector with mechanisms like multi-head attention</td>
<td><img src="image1.png" alt="Diagram" /></td>
<td>MV Selection</td>
</tr>
<tr>
<td><strong>Graph Convolutional Network</strong></td>
<td>Encode graph-structured input features with convolutions on the vertex features and their K-hop neighbor vertices</td>
<td><img src="image2.png" alt="Diagram" /></td>
<td>Query Latency Prediction</td>
</tr>
<tr>
<td><strong>Meta Learning</strong></td>
<td>Use the base models to form the target model based on the task similarity and the prediction accuracy during usage</td>
<td><img src="image3.png" alt="Diagram" /></td>
<td>Knob Tuning</td>
</tr>
</tbody>
</table>