Umar *et al.LGU (IJECI)* 2020

LGU (IJECI)
ISSN: 2522-3429 (Print)
ISSN: 2616-6003 (Online)

LGU International Journal for
Electronic Crime Investigation

Research Article

Vol. 5 issue 3 Year 2021

Optimal Query Execution Plan with Deep Reinforcement Learning

Umar Jamshid1, Muhammad Umar Afzal[2]
umer.khokher@gmail.com1 umar.afzl@ymail.com[2]
University of Lahore

Abstract:

We examine the use of profound support learning for inquiry development. The technique is to gradually construct queries by encoding features of sub-inquiries using a learnt representation. We specifically focus on the organization of the state progress effort and the state portrayal issue. We provide preliminary results and investigate how we might use the state representation to further refine question streamlining using assist learning.

1. Introduction

Inquiry advancement is still a big concern in the field of data sets. Existing DBMS select helpless execution strategies for certain queries. To make inquiries more competent, we wished to design them optimally, utilizing fewer assets. Existing DBMSs carry out a vital stage of cardinality evaluation by working on assumptions about the information (e.g., incorporation standard, consistency or freedom suppositions). When these concerns are not confirmed, cardinality evaluation errors occur, resulting in poor arrangement choices. [1]. By using cardinality gauges as input, the expenditure model selects the least expensive alternative from semantically equivalent arrangement options. To achieve an efficient inquiry strategy, a subset of the valid join orders is counted by the question enhancer, for example, using dynamic programming.

Theoretically this architecture can obtain the effective optimal plan if the cardinality estimation and cost model is precise. In reality cardinality estimates depends on computer-based assumptions. But in real world databases the assumptions like uniformity and independence are wrong.

In this research work, rather than banking on previously used formulas and data driven with the help of statistics, to predict the queries cardinalities we will train a deep reinforcement learning model for better execution plan. We will construct a model that can learn data and

properties of data to be exact about the estimates. The major part of this model is that it will learn the subquery representations of complex queries which will be used to build query execution plan using deep reinforcement learning. Since the 1970s, information base analysts have been dealing with framework enhancement and large-scale information-driven applications, which are strongly associated with the first two components. Although deep learning methodologies are not often used in dealing with DBMS challenges, it is natural to wonder about the linkages between information bases and profound learning.

First, we have to examine that is the database community ready to adapt deep learning for DBMS. However, there are fewer examples of using machine learning for traditional database problems that are less uncertain, like indexing etc. whereas deep learning is good at predicting the events that are mostly contains uncertainty. There are problems faced in databases that are probabilistic like crowdsourcing etc. In particular, we divide the method in two parts, first representation of state of table using deep learning and then we will present a way to computes plan for the given query using the above states together with deep reinforcement learning.

Challenge of this approach, to represent data and query. First, we develop an approach that will incrementally generate result of subqueries. Subqueries and a new operation will be provided as input that will further predict the representation of output. This representation of output subquery will be used to derive the subquery's cardinality.

The major part is that we will present a method that will use this representation to enumerates the query execution plan through deep reinforcement learning. Support learning is a generally beneficial structure used for dynamic in circumstances where a framework is absorbed by experimentation from remunerations and discipline. [2]. We propose to use this deep learning approach to build an optimal query execution plan by modelling it as Markov process; in which each decision has its dependency on each stage. The figure below will illustrate our method. The figure below has DB and a query, the model will generate an optimal query execution plan by determining the series of state transitions

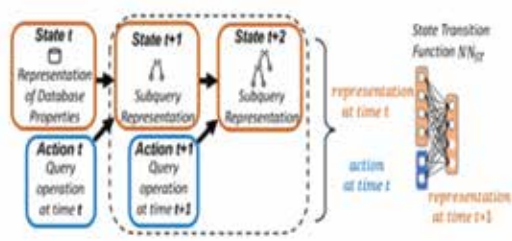


Figure 1: Database System

The system in the initial state t in the illustration represents a whole database. We will select as action using deep reinforcement learning, the model moves to a new state at **t +1**, We've now built a bigger subquery. Each action is a query operation, and each state reflects the intermediate results of the subquery. To build this representation, we used a neural network, i.e. a state transition function, NNST. NNST is a recursive function that generates the subquery representation at time

t+1 by taking the previous subquery representation as input and an action at time t.

Let us now define the setup that is engineered above. The query plan's dynamics are bottom-up, with one operation at a time. Assume a subquery has been constructed at any step t of the query plan, and the state at t is represented by an n-dimensional vector h_t. When the next action applied to the setup, at to this current database state leads to the next state, ht+1. The mapping, $\mathbf{NN_{ST}: (h_t, a_t) \rightarrow h_t+1}$ is called the state transition function. This state and state transition function are well known in applications of deep reinforcement learning. For Example, in the game of Chess, each possible position is called state and the transition of these states from one board position to another is well-defined. However, in the case of a database, if the query execution plan is not sufficiently described, we cannot forecast the status of the query. The core of our proposed strategy is to identify each state using a finite dimensional vector and then learn the state transition function using a deep reinforcement learning model. We employ input signals and context from observed variables linked with database status to drive the training procedure for this network. Throughout this work, we will utilise the cardinality of each subquery as an observable variable at any point of the plan. If we can learn a function, $NN_{observed}$, that maps this state to projected cardinalities at stage t, we should be able to learn a function, $NN_{observed}$, that maps this state to predicted cardinalities at stage t. In the figure below, we display both NNST and $NN_{observed}$.

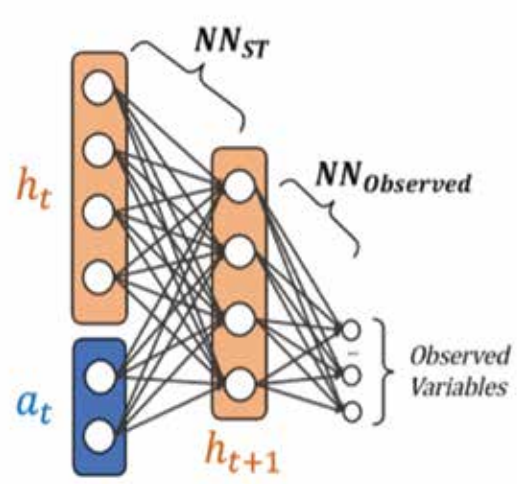


Figure 2: Representation of NN_{ST} and $NN_{Obsereved}$

When the suggested model has been sufficiently trained, we will update the network parameters depending on query operation sequences. Each state will learn to precisely depict a representation using this approach. Once the technique has been fully trained, we may correct it and use deep reinforcement learning to create a suitable action policy, resulting in an optimal query execution plan.

2. Literature Review:

Extensive Learning Deep learning methods, commonly known as feedforward neural networks, may imprecisely approximate a nonlinear function, f [3]. Through a collection of learnt parameters spread across multiple layers, these models establish a mapping from an input x to an output y. The behavior of the interior layers is not dictated by the input data during training; instead, these models must learn how to employ the layers to create the proper output. Because the layers have no

direct interactions with the input training data, they are referred to as hidden layers [3]. These feedforward networks are crucial in representation learning. While training to perform a goal function, the hidden layers of a neural network might indirectly learn a representation that can later be employed for other tasks [3]. There is a trade-off between retaining as much information as possible and understanding relevant data qualities. The context of these representations might alter depending on the network output [3]. Learning through Reinforcement learning models can map states to appropriate actions with the objective of maximizing a larger reward. In contrast to supervised learning, the learner is not clearly informed which action is ideal; instead, the agent must determine the best action through hit-and-run by either exploiting current information or finding novel states [11]. At each time step, the learner will examine the condition of the environment, S_T, and choose an action, a_t. The policy determines the action to be taken. This strategy has the potential to reconstruct a variety of behaviors. As a result, either behave greedily or strike a balance between discovering and utilizing through a greedy (or better) method. The policy is determined by the predicted rewards of each state, which the model must learn on the fly. The model will arrive at a new state, S_{T+1}, based on the action chosen. The environment then provides the agent a reward, r_{T+1}, signifying the "worthiness" of the chosen action. The goal of the agent is to maximize the overall reward [11]. One method is to employ a value-based iteration methodology in which the model records state-action values, such as QL (s, a). These values indicate the state's long-term worth by weighting rewards for states that are anticipated to follow.

3. Methodology:

They are two approaches that we proposed to achieve the target solution for getting the optimal query execution plan. We will be supposed input database ***D*** and Query ***Q.*** we will apply deep reinforcement learning to derive compact but informative representation of queries, then we will try to train these representations to predict the next action. In first approach we would suppose a feature vector containing (Q, D) as input and apply deep reinforcement learning to predict an output as cardinality values. There is problem with this approach that whenever the database and query complexity increase the input vector grow heavily. Thus, the long-extended vectors required large training datasets.

Instead of wasting our resources and never getting our required result, we will move forward and apply our better different approach, a recursive approach: We train a model to anticipate a query's cardinality. This model is fed a pair of (h_t, a_t) as input, where h_t is a vector representation of a subquery and a_t is a single relational action on h_t. Importantly, because h_t is the representation that the model will learn on its own, it should not be interpreted as a manually supplied feature vector. The NN_{ST} function, seen in the image above, builds these representations by modifying the weights in response to feedback from the $NN_{Observed}$ function. This $NN_{Observed}$ function learns to predict observed variables by mapping a

subquery representation. Back propagation is used to alter the weights for both functions while we train this model.

Before moving forward and start using the recursive NN_{ST} model, we have to understand an additional function, $\mathbf{NN_{init}}$. NN_{init} will take $\mathbf{(x_0, a_0)}$ as input, where x_0 represents a vector that holds the properties of the database D whereas a_0 shows a relational operator. The model outputs the cardinality of the query that executes the operation encoded in a_0 on D. By this we can achieve the optimal time on the execution of the query.

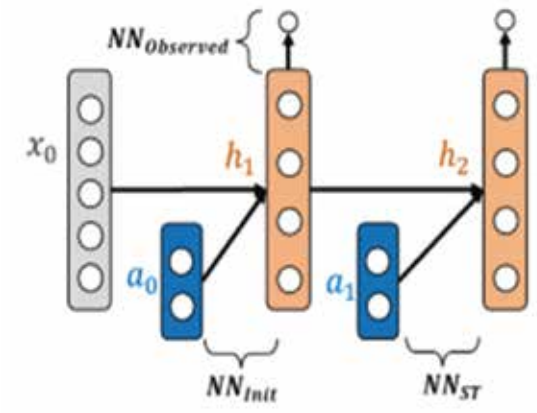
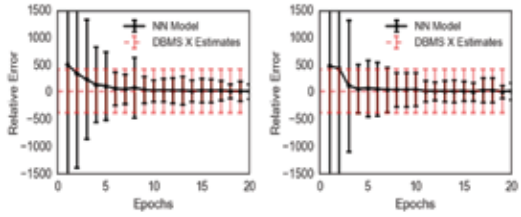


Figure 3: Input and Output Variables

x_0 the vector represents simple properties of the database, D. For each attribute in the dataset D, we use the following features to define x_0: ***the min value***, ***the max value*** and ***the number of distinct values***.

4. Conclusion

In this paper, we provide a model for query optimization that uses deep reinforcement learning. We employ deep neural networks to gradually learn state representations of subqueries by storing fundamental information about the input. In future work, we propose combining these state representations with a reinforcement learning model to develop optimum plans.



(a) Predicting Cardinality (m = 3) **(b) Predicting Cardinality (m = 5)**

5. References:

[1] Kostas Tzoumas et al. A reinforcement learning approach for adaptive query processing. In A DB Technical Report, 2008.

[2] Ron Avnur et al. Eddies: Continuously adaptive query processing. SIGMOD Record, 2000.

[3] Volodymyr Mnih et al. Human-level control through deep reinforcement learning. Nature, 2015.

[4] Viktor Leis, Andrey Gubichev. How Good Are Query Optimizers, Really? CoRR, 2017.

[5] Richard S. Sutton et al. Reinforcement learning I: Introduction, 2016.

[6] Wei Wang†, Meihui Zhang et al. Database Meets Deep Learning: Challenges and Opportunities. January 2020.

[7] Ryan Marcusetal. Deep reinforcement learning for join order enumeration. CoRR, 2018.

[8] Ian Goodfellow et al. Deep Learning. MIT Press, 2016. http://www.deeplearningbook.org.

[9] David Silver. UCL Course on Reinforcement Learning, 2015.

[10] Michael Stillger et al. Leo - db2's learning optimizer. In VLDB 2001.

[11] Viktor Leis et al. How good are query optimizers, really? Proc. VLDB Endow., 2015.

[12] Csaba Szepesvari. Algorithms for reinforcement learning. Morgan and Claypool Publishers, 2009.

[13] Martín Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[14] Wei Wang et al. Database Meets Deep Learning: Challenges and Opportunities. SIGMOD Record, 2016

[15] Henry Liu et al. Cardinality estimation using neural networks. In CASCON 2015