The Use of Machine Learning in Database Systems

A Technical Report
presented to the faculty of the
School of Engineering and Applied Science
University of Virginia

by

Grady Roberts

with

Daniel Collins

May 9, 2021

On my honor as a University student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments.

Grady Roberts

Technical advisors: John A. Stankovic, Department of Computer Science
Sebastian Elbaum, Department of Computer Science

The Use of Machine Learning in Database Systems

Daniel Collins
University of Virginia
dpc7ns@virginia.edu

Grady Roberts
University of Virginia
gnr7aj@virginia.edu

Abstract

The total volume of enterprise data generated each year is expected to reach 175.8 zettabytes (ZB, equivalent to one trillion gigabytes) by 2025, up from 18.2 ZB in 2015 [1]. Seagate's survey also found that only 32% of this data is available in a state ready to be analyzed. With data volume growing faster than companies can keep up with, it is important to find efficient ways to store, manage, and analyze data. One approach is the application of machine learning (ML) to aspects of database design and query optimization. To this end, a literature review of the use of ML to address the problems of managing data at scale is completed. These problems include creating an efficient index of the data, the expense of joining tables, and the computational cost of copying data into another repository for analysis. We contribute a taxonomy of different methods to solving these problems as well as present open challenges in this area. This taxonomy will describe the benefits and drawbacks of each method.

1 Introduction

The technical project takes focus from two upper-level CS courses: CS 4750 Database Systems and CS 4774 Machine Learning. The goal of the project is to understand how ML can inform good database design strategies, as well as optimize database operations. Understanding the current state of the art in these areas provides insight into the best ways for companies to manage and extract useful information from the data they collect. The project addresses three problems endemic to database systems: building an efficient index, inefficient join queries, and being able to perform modeling and analysis within the database instead of having to pay the cost of pipelining the data to an external tool.

2 Related work

In a 2000 paper, Wu [15] identified future challenges in combining machine learning and database technology to create Intelligent Learning Database (ILDB) systems. Wu identified two main areas focusing on efficient representation of the data and machine learning techniques capable of handling enterprise-scale database systems. He also noted that real-world databases are typically noisy and contain missing or incorrect data, anticipated to be a challenge for learning systems. While Wu's paper is not contemporary, it provided a foundational view of likely problems for the future.

In a 2015 literature review, Najafabadi et al. [12] identified challenges in adapting current machine learning methods to modern database scenarios. The areas the authors identified are streaming data, high-dimensional data, scalability of models, and distributed computing. These areas relate to problems in the ML algorithms themselves rather than problems in using ML to create better database systems.

3 Literature review

To find relevant publications for our literature review we searched Google Scholar for works that included both machine learning and database systems. As we read, we synthesized a taxonomy of problems facing database systems that were being addressed by machine learning. The problems we identified were indexing the database, optimizing join queries, and performing modeling and analysis within the database system.

3.1 Indexing

To perform queries efficiently, database designers create a data structure called an index to order data so that it can be quickly accessed. The type of index used depends both on the structure of the data and the types of queries that will be performed most often. The most common types of indices used are the BTree-Index for searching ranges, Hash-Index for single lookup, and BitMap-index to check key existence. All these types of indices are built using fixed data structures that do not change in response to underlying data patterns.

Beutel et al. [2] argued that these indices fail to exploit patterns in the data to further optimize database operation. Using indices created using neural networks, Beutel et al. demonstrated a 44% improvement in the speed of queries and a 75% reduction in the memory required to store the index compared to a BTree-Index on a read-only database. The choice of ML-based indices also has drawbacks, such as planning to handle false negatives (i.e. the ML index reports that the key does not exist when it does) which do not occur with traditional indices, and ML indices returning approximate locations of data rather than exact record locations. To address these, Beutel et al. combine learned indices with traditional indices, allowing the faster ML index to perform the bulk of the search and the traditional index to provide guarantees that the search is correct. This work only examined read-only databases, but the authors argue that an online learning ML index could handle writable databases.

Database management systems (DBMS) workloads typically vary over time. For example, a bus tracking app may experience high workload during the day when buses operate and few queries at night when buses don't operate. Ma

et al. [10] argue that a DBMS capable of forecasting its workload can optimize its indices to perform better as the workload changes and operate autonomously. In current DBMS, human administrators are responsible for manually tuning the optimizations. Ma et al. present a workload forecasting framework called QueryBot 5000 which uses an ensemble model of linear regression, recurrent neural network, and clustering to predict future workload patterns. Clustering queries is important to reduce the complexity of the workload modeling and allow efficient prediction. Ma et al. found that 95% of the workload on the experimental systems could be modeled using the five largest clusters. This framework provides both short- and long-term predictions of workload, allowing the automated DBMS to choose the best optimizations. Unlike Beutel et al.'s index optimizing framework, the QueryBot 5000 framework uses online ML models that adapt to live changes in incoming workloads. One limitation of this framework is that some manual tuning of the threshold at which the workload has changed enough to trigger a recluster is still required.

Licks and Meneguzzi [8] also explores the design space of developing machine learning algorithms to tackle the problem of creating efficient indices for a database, a job typically carried out by database administrators (DBA). The goal was to remove the DBMS's dependence on domain experts for index selection, and to instead design indices based on the database schema and workload that it receives. The architecture, named SmartIX, abstracts away the task of analyzing and deciding whether candidate columns for the index are going to improve performance, a task that is normally done by humans. Instead, SmartIX contains a reinforcement learning agent whose purpose is to explore possible candidate columns for the index. It also contains an environment where the agent receives rewards for its decisions and has its next actions transferred to the third component, the DBMS interface which writes the commands which change the database and also reads in statistics about the current indices performance. Agent training was done using a two-layer perceptron neural network over 64,000 time steps, where training statistics were gathered every 128 steps and used to give a graphical representation of the results. When the performance of the index configuration created by SmartIX was compared to configurations made by genetic algorithms and other reinforcement models, it was found to have higher queries per hour than its competitors, while still maintaining a comparable index size in MB. However, SmartIX cannot deal with composite indices (those that use two or more columns). Licks and Meneguzzi say that in future work that they plan to "(1) investigate techniques that allow us to deal with composite indices; (2) improve the reward function to provide feedback in case of write-intensive workloads; (3) investigate pattern recognition techniques to predict incoming queries to index ahead of time; and (4) evaluate SmartIX on big data ecosystems (e.g. Hadoop)."

The first self-driving DBMS, Peloton, was developed by Pavlo et al. [13]. Peloton is distinct from previously mentioned architectures as it doesn't focus on a single aspect of the database, like how SmartIX focuses exclusively on indexing of the database. Pavlo et al. claim that these architectures are "insufficient for a completely autonomous system because they are (1) external to the DBMS, (2) reactionary, or (3) not able to take a holistic view that considers more than one problem at a time. That is, they observe the DBMS's behavior from outside of the system and advise the DBA on how to make corrections to fix only one aspect of the problem after it occurs." Existing DBMS like Postgres or MySQL were ill-suited to being made self-driving, which further pushed for the creation of a new architecture. The first component of Peloton is the Workload Classifier, which uses unsupervised learning methods to cluster queries by type, which makes them easier to analyze later on. Then the Workload Forecaster predicts the query arrival rate for each cluster and creates forecast models for them utilizing Recurrent Neural Networks (RNNs). Using these forecasts Peloton populates a catalog of actions it could take to potentially improve database performance. Actions are stored along with their cost, which is measured by the number of CPUs they will use. Peloton implements actions during times of low demand when more CPUs are available. Peloton chooses which action to deploy using a receding-horizon control model (RHCM). Finally Peloton tries to update the database while minimizing impact to users trying to query the database. Results from testing Peloton were promising, the authors concluded that "(1) RNNs accurately predict the expected arrival rate of queries, (2) hardware-accelerated training has a minor impact on the DBMS's CPU and memory resources, and (3) the system deploys actions without slowing down the application."

3.2 Optimizing join Queries

One of the most expensive database operations is that of joining multiple tables together to construct the result of a query. Join operations are expensive partly because the database must determine the most efficient ordering of the tables in the join. This decision is typically done using heuristics to reduce the search space.

Markl et al. [11] created LEO (LEarning Optimizer) in 2003, the first example of a query optimizer that uses ML to improve performance by learning from historical data. LEO incrementally adjusts cardinality and selectivity estimates for predicates to correct modeling mistakes made in old query execution plans. LEO can detect estimation errors during a query execution and automatically trigger a reoptimization. LEO works by recording actual cardinalities and learning to minimize the error between actual and predicted cardinalities. One challenge faced by Markl et al. that is still common in modern ML is the convergence problem in which the learning rate of a model is highly dependent

on the data and assumptions about the data. The authors addressed this issue by allowing LEO to operate in two modes: an exploratory mode which chooses risky but low-cost plans based on assumptions and a more concrete mode that favors plans based on experience and ground truth. LEO also addresses the issue of correlated joins because it can detect the discrepancy between the predicted and actual cardinality. Another concern raised by the authors is the problem of deciding when it is worthwhile to reoptimize a query mid-execution because of non-linearity in the cost model, a problem worked on by Krishnan et al. [6] later.

Liu et al. [9] utilized a 3-layer neural network to approach optimizing join queries. They created their own algorithm to generate the initial training queries for their neural network. The neural network can only read non-strict range operators from queries, operators like "<=" or ">=" , but they devised ways of rewriting all other relational operators like "=" and "<" or ">." They compared the neural network to the query optimizer built into DB2, a DBMS from IBM. The results showed that neural networks better estimated the cardinality of queries than statistics-based estimation methods. However, Liu et al. noted that there was further work they wished to achieve, including being able to compare columns with relational operators since, as of the moment, one can only compare column values to constants ($c_1 >= 3$ but not $c_1 >= c_2$). Secondly, they wanted to try implementing other types of neural networks such as RNNs and deep neural networks. Later developments in this area have used more complex models.

Kipf et al. [5] apply supervised learning to predict the cardinality of intermediate steps in a plan generated by a classical join optimizer. One of the biggest challenges for an optimizer selecting between plan alternatives is to have an accurate estimation of the cardinality of the intermediate steps, which can be complicated by cross-join correlations. A new ML model created by the authors called multi-set convolutional network (MSCN) allows the already known structure of the data to be encoded before training the model, avoiding the need to serialize the data which is required by other convolutional models. When MSCN was tested on a real-world IMDB dataset, it was competitive with state-of-the-art sampling-based join optimizers in the median case and greatly outperformed them in the presence of correlated joins. In addition to good performance, MSCN required less space than traditional optimizers. One limitation is that the training data contained at most two joins per query and the testing data included queries with more joins, requiring the model to extrapolate. Another limitation, similar to Beutel et al., is that the paper assumes a read-only database.

In contrast with the work by Kipf et al., Krishnan et al. [6] attempted to solve the problem of optimizing execution plans for join queries with reinforcement learning. The authors showed that heuristics tend to fail to find the most efficient join ordering when there is non-linearity in the database cost model, a problem that can be caused by correlation. To correct this, Krishnan et al. created a reinforcement learning ML model capable of outperforming existing heuristics by learning and accounting for non-linearity. The reason for the improved performance is that heuristics tend to be greedy, meaning they will take short-term benefit potentially at the cost of greater long-term benefit. In contrast, Krishnan et al.'s ML model is capable of learning to optimize for the long-term. In addition to producing better optimization plans, the RL join optimizer was also shown to generate plans more than ten times faster than traditional optimization methods such as left-deep heuristic. A limitation of this work is that the authors only consider one type of join operation that composed the bulk of the benchmarks tested.

3.3 In-database modeling

Another interesting area of research is the creation of new programming languages that create in-database ML models. These models can exploit the locality of the data and avoid lengthy pipelines to move data into another repository to be analyzed.

MADlib is an analytics library for Greenplum and SQL developed by Hellerstein et al. [4]. MADlib supports the implementation of traditional machine learning algorithms such as SVM and K-means. It also supports myriad supervised and unsupervised machine learning algorithms, and is open-source. Its codebase receives contributions from industry and academia alike. The authors' goal was that it would "accelerate innovation and technology transfer in the Data Science community via a shared library of scalable in-database analytics." Machine learning and other advanced analysis techniques with MADlib are implemented using simple SQL scripts. To run a sequence of these SQL statements or do iterative runs of analysis, one can also use a python-based driver file to run MADlib. Users also have access to templated queries which assist when the schema for input tables to a module are not necessarily fixed. MADlib has seen use in collaborations with researchers at the University of Wisconsin, and also at the University of Florida, and at Berkeley. MADlib has not yet seen any use commercially.

Another similar language is MLog, developed by Li et al. [7], a declarative programming language that interfaces directly with the database. MLog improves on MADlib by allowing users to create more complicated models. MLog allows users to create complex deep learning models within the database in the same manner that users can open a SQL session and run simple queries against the data in-place. Deep learning models typically require a large amount of training data to be accurate. MLog allows the model to be created where the data is, saving time and simplifying the model-building process. The software operates by viewing the data as tensors, and includes functionality to automatically optimize the model-building process. ML models created with MLog's declarative query language can be compiled into

native TensorFlow programs with comparable performance to hand-optimized models. The authors found that, in some cases, TensorFlow programs created with MLog can still be up to two times slower than hand-tuned code. Another limitation is that the MLog language was not fully integrated with SciDB.

3.4 Open challenges

Wang et al. [14] discusses how both deep learning systems could possibly be improved from a database perspective and how database applications can be improved by deep learning techniques. They found that deep learning could take advantage of the optimization techniques used in the database field in the training of its models. Techniques such as operation scheduling, memory management, parallelism, and concurrency/consistency. Likewise, the database field could utilize natural language processing, a hallmark of deep learning, to interpret informal queries to a database and translate them to formal requests. Additionally, query planning, traditionally done using complex cost models in database, could instead be done more efficiently by an RNN model. Finally, deep learning can be used to help in the analysis of spatial/temporal data stored in databases by mapping the problem into an image-like structure and then using a convolution neural network to exploit the spatial locality of the data. In all, an exchange of techniques and representations between the database and deep learning communities would greatly benefit both fields.

Guo and Daudjee [3] describe limitations of recent uses of deep learning for join query optimization. They found that, in the worst case, these models produce execution plans that are orders of magnitude more expensive than the cheapest plan and can even crash the DBMS by requiring too much memory. They argue that the extra time required to execute the worst plans that the model outputs may outweigh the time saved by using deep learning. This indicates the need to balance high performance with consistency. The authors believe that more research is needed to improve model robustness for this application. The authors also argue that better feature encoding is necessary to extend deep learning join query optimizers to enterprise databases with many tables that may change often, requiring retraining the model. They argue that feature encoding that is robust to table changes should be a requirement for future research.

4 Conclusions

We presented a literature review about the use of machine learning in database systems. In our research we did not come across any prior literature review that covered this topic. We also contribute a taxonomy of research in this area. It is apparent that machine learning has the capability to greatly improve database efficiency and autonomy, but there are still challenges in making these hybrid systems viable for commercial applications.

5 Future work

The use of machine learning to improve database design and operation is a relatively new and fast-moving area of study. We recommend that further literature reviews be completed to track the progress of this field.

References

[1] 2020. *Rethink Data: Put More of Your Business Data to Work—From Edge to Cloud.* Technical Report. Seagate. 56 pages. https://www.seagate.com/files/www-content/our-story/rethink-data/files/Rethink_Data_Report_2020.pdf

[2] Alex Beutel, Tim Kraska, Ed H Chi, Jeffrey Dean, and Neoklis Polyzotis. 2017. A Machine Learning Approach to Databases Indexes. *Conference on Neural Information Processing Systems* (2017), 5.

[3] Runsheng Benson Guo and Khuzaima Daudjee. 2020. Research challenges in deep reinforcement learning-based join query optimization. In *Proceedings of the Third International Workshop on Exploiting Artificial Intelligence Techniques for Data Management (aiDM '20).* Association for Computing Machinery, New York, NY, USA, 1–6. https://doi.org/10.1145/3401071.3401657

[4] Joe Hellerstein, Christopher Ré, Florian Schoppmann, Daisy Zhe Wang, Eugene Fratkin, Aleksander Gorajek, Kee Siong Ng, Caleb Welton, Xixuan Feng, Kun Li, and Arun Kumar. 2012. The MADlib Analytics Library or MAD Skills, the SQL. *arXiv:1208.4165 [cs]* (Aug. 2012). http://arxiv.org/abs/1208.4165 arXiv: 1208.4165.

[5] Andreas Kipf, Thomas Kipf, Bernhard Radke, Viktor Leis, Peter Boncz, and Alfons Kemper. 2018. Learned Cardinalities: Estimating Correlated Joins with Deep Learning. *arXiv:1809.00677 [cs]* (Dec. 2018). http://arxiv.org/abs/1809.00677 arXiv: 1809.00677.

[6] Sanjay Krishnan, Zongheng Yang, Ken Goldberg, Joseph Hellerstein, and Ion Stoica. 2019. Learning to Optimize Join Queries With Deep Reinforcement Learning. *arXiv:1808.03196 [cs]* (Jan. 2019). http://arxiv.org/abs/1808.03196 arXiv: 1808.03196.

[7] Xupeng Li, Bin Cui, Yiru Chen, Wentao Wu, and Ce Zhang. 2017. MLog: towards declarative in-database machine learning. *Proceedings of the VLDB Endowment* 10, 12 (Aug. 2017), 1933–1936. https://doi.org/10.14778/3137765.3137812

[8] Gabriel Paludo Licks and Felipe Meneguzzi. 2020. Automated Database Indexing using Model-free Reinforcement Learning. *arXiv:2007.14244 [cs]* (July 2020). http://arxiv.org/abs/2007.14244 arXiv: 2007.14244.

[9] Henry Liu, Mingbin Xu, Ziting Yu, Vincent Corvinelli, and Calisto Zuzarte. 2015. Cardinality estimation using neural networks. In *Proceedings of the 25th Annual International Conference on Computer Science and Software Engineering (CASCON '15).* IBM Corp., USA, 53–59.

[10] Lin Ma, Dana Van Aken, Ahmed Hefny, Gustavo Mezerhane, Andrew Pavlo, and Geoffrey J. Gordon. 2018. Query-based Workload Forecasting for Self-Driving Database Management Systems. In *Proceedings of the 2018 International Conference on Management of Data.* ACM, Houston TX USA, 631–645. https://doi.org/10.1145/3183713.3196908

[11] V. Markl, G. M. Lohman, and V. Raman. 2003. LEO: An autonomic query optimizer for DB2. *IBM Systems Journal* 42, 1 (2003), 98–106. https://doi.org/10.1147/sj.421.0098 Conference Name: IBM Systems Journal.

[12] Maryam M. Najafabadi, Flavio Villanustre, Taghi M. Khoshgoftaar, Naeem Seliya, Randall Wald, and Edin Muharemagic. 2015. Deep learning applications and challenges in big data analytics. *Journal of Big Data* 2, 1 (Feb. 2015), 1. https://doi.org/10.1186/s40537-014-0007-7

[13] Andrew Pavlo, Gustavo Angulo, Joy Arulraj, Haibin Lin, Jiexi Lin, Lin Ma, Prashanth Menon, Todd C Mowry, Matthew Perron, Ian Quah,

Siddharth Santurkar, Anthony Tomasic, Skye Toor, Dana Van Aken, Ziqi Wang, Yingjun Wu, Ran Xian, and Tieying Zhang. 2017. Self-Driving Database Management Systems. *Conference on Innovative Data Systems Research* (2017), 6.

[14] Wei Wang, Meihui Zhang, Gang Chen, H. V. Jagadish, Beng Chin Ooi, and Kian-Lee Tan. 2016. Database Meets Deep Learning: Challenges and Opportunities. *ACM SIGMOD Record* 45, 2 (Sept. 2016), 17–22. https://doi.org/10.1145/3003665.3003669

[15] Xindong Wu. 2000. Building Intelligent Learning Database Systems. *AI Magazine* 21, 3 (Sept. 2000), 61–68. https://doi.org/10.1609/aimag.v21i3.1524 Number: 3.