Cost-Guided Cardinality Estimation: Focus Where it Matters

Parimarjan Negi, Ryan Marcus, Hongzi Mao, Nesime Tatbul, Tim Kraska, Mohammad Alizadeh
MIT CSAIL
{pnegi, rcmarcus, hongzi, tatbul, kraska, alizadeh}@mit.edu

***Abstract*—The increasing prevalence of machine learning techniques has resulted in many works attempting to replace cardinality estimation, a core component of relational query optimizers, with learned models. The majority of those works have trained models to minimize the prediction error between the model's output for a particular query and the true cardinality of that query. However, when cardinality estimators are used for query optimization, not all cardinality estimates are equally important. We present *cost-guided cardinality estimation*, a technique to train learned cardinality estimators that penalizes models for errors that lead to sub-optimal query plans, and rewards models for estimates that lead to high-quality query plans, regardless of the accuracy of those estimates. In a preliminary experimental study, we show that our technique can reduce average query runtime by 1.7–2×. Surprisingly, models trained with our approach achieve this increase in query performance while having higher prediction error than models trained without our approach, suggesting that prediction error for cardinalities is not necessarily the correct metric to optimize.**

I. Introduction

Cardinality estimation, the problem of predicting the selectivity of a query, is a core component of query optimizers. A cardinality estimator, combined with a cost model, is used to predict the cost of different query plans and search for the best one. Traditionally, query optimizers have relied on simple histogram-based cardinality estimators. These estimators make strong assumptions about the underlying data, like attribute value independence and uniformity. As a result, they perform poorly on queries that do not satisfy these assumptions. In fact, past work suggests that most failures in query optimizers are due to poor cardinality estimates [1].

A flurry of recent work applies machine learning techniques to improve cardinality estimation [2–6]. A majority of these works train models to predict cardinality by minimizing the model's prediction errors - such as the QError [7]. However, not all predictions are equally important: it is possible that a small misprediction could cause an optimizer to select a slow join order, and it is also possible that a large misprediction has no impact on a query plan. Motivated by this observation, in this paper we ask: *Is it possible to optimize a cardinality estimator for query performance, rather than prediction error?*

We propose a new technique, called cost-guided cardinality estimation, which, at training time, emphasizes predictions that matter most to query plans. Our method relies on a new metric, *Plan-Error*, that captures the impact of cardinality estimates on the optimized plans generated by a DBMS. To train the cardinality estimator, we sample queries based on their Plan-Error, giving larger weight to queries for which better cardinality predictions would improve the query plan the most. Our hypothesis is that this approach can use limited model capacity more efficiently, resulting in cardinalty estimators that increase the final quality of the resulting query plans. Our approach is general and can be used to train any learning-based cardinality estimator.

We apply our approach to two cardinality estimation models [4, 5]. When compared against an identical model trained without cost-guided learning, our approach improves resulting query runtimes by nearly a factor of two. Interestingly, *the models trained with cost-guided learning actually have much higher prediction error, despite resulting in query plans that are significantly faster.* This highlights the fact that prediction error alone is unsuitable to evaluate cardinality estimators.

In Section II, we describe the notion of Plan-Error and our cost-guided learning approach. We present experimental results in Section III. Related works are presented in Section IV. Concluding remarks and directions for future work are given in Section V.

II. Cost-Guided Learning

Given a query, Q, an estimator should output a cardinality estimate for each of its sub-queries. Most cardinality estimation work focus on producing estimators with low QError (multiplicative error) [7]:

$$\text{QError}(y, \hat{y}) = \max(y/\hat{y}, \hat{y}/y), \tag{1}$$

where y and $\hat{y}$ are the true and predicted cardinalities, respectively. In many learned cardinality estimation works, QError is directly minimized via gradient descent [5], decision tree induction [8], or other means [3].

Moerkotte et al. theoretically justify why QError is a better loss metric than relative error or absolute error for the purposes of query optimization [7]. Intuitively, this is because small cardinality mis-estimates can multiply and become much larger after joins. But this is an indirect way to achieve our goal: minimize the runtime of the final query plan produced due to the estimated cardinalities.

A. Plan-Error

Perhaps the most obvious way to train a cardinality estimator that optimizes for query runtime, instead of just for QError, is to give greater importance to those queries that have worse runtimes due to the estimator's predictions. However, this would require frequently re-evaluating the queries throughout training, which is prohibitively expensive. Therefore, we define *Plan-Cost*, which is a proxy for runtimes based on the optimizer's cost model.

For any query, let $\mathbf{C}$ be true cardinalities and $\hat{C}$ be estimated cardinalities for all its sub-queries, and P be a particular query plan. Here, the query plan consists of the join order, join operators, and the indexes output by the query optimizer.

For an arbitrary cost model, such as the one in PostgreSQL, a set of estimated cardinalities, $\hat{C}$, and a query plan P, let $Cost\ (\hat{C}, P)$ denote the cost of plan P. A query optimizer uses a search algorithm to find the optimal query plan:

$$P^{\star}(\hat{C}) = \arg\min_{P} \mathrm{Cost}(\hat{C}, P)$$

We define the *plan cost* PC for any set of cardinality estimates $\hat{C}$ as the cost of the optimal plan $P^{*}(\hat{C})$ with respect to *the true cardinalities* $\mathbf{C}$:

$$\text{Plan-Cost}\ (\hat{C}) = \mathrm{Cost}(\mathbf{C}, P^{\star}(\hat{C})).$$

Notice that Plan-Cost($\mathbf{C}$) will be the the lowest possible cost for a given query, since $P^{\star}(\mathbf{C})$ is the optimal plan with respect to the true cardinalities. Finally, we can define the Plan-Error as the difference in plan cost with the estimated cardinalities versus the true cardinalities:

$$\text{Plan-Error} = \text{Plan-Cost}(\hat{C}) - \text{Plan-Cost}(\mathbf{C})$$

Unlike the runtime of a query plan [9, 10], the Plan-Error gives us a metric that we can compute easily during training - since all the true cardinalities would have already been computed to generate the dataset before training. However, unlike QError, the Plan-Error is not differentiable, since it uses the dynamic programming algorithm of the query optimizer, so we can not directly optimize for it during training using gradient descent. We use an idea, inspired by Schaul et al. [11], to focus our learner's capacity on samples which are the most important for reducing Plan-Error.

B. Prioritized Training with Plan-Error

We consider the standard setting of training a parametric model (e.g., a neural network) with supervised learning using batch stochastic gradient descent. We use QError as the loss function similar to existing approaches. However, we sample sub-queries used in each training epoch non-uniformly based on the Plan-Error. Intuitively, if a particular query has a high Plan-Error, we train on that query more frequently, causing the model to adjust its weights to achieve a lower QError on that query. On the other hand, if a particular query has a low Plan-Error, we train on that query less frequently, de-emphasizing the query and freeing up model capacity for other queries.

Let $q_i > 0$ be the priority for each sample (sub-query) in the training set. After every training epoch, the priority q_i is set to be the Plan-Error of the query which that sample belongs to. To prevent sudden shifts in priorities causing unstable learning behaviour, we use a moving average of the last four epochs to calculate the priority.

During a training epoch, we compute the probability of selecting sample i according to:

$$P(i) = \frac{{q_i}^{\alpha}}{\sum_k q_k^{\alpha}}$$

Here, α controls how much weight we give to the priority (and hence the Plan-Error). With $\alpha = 0$, we get the usual case, in which the training samples are chosen uniformly. In other words, with $\alpha = 0$, our approach is exactly the same as that used in previous work [4, 5]. Higher values of α increase the impact of the Plan-Error. We manually tuned the α parameter, and set it to 2 for all our experiments.

III. Experiments

In this section, we present preliminary experimental results demonstrating the efficacy of our method on the MSCN model [5] and the FCNN model [4]. We additionally introduce a new query dataset specifically tailored to evaluating learned cardinality estimators.

Dataset We created a new set of queries [12] for the purpose of evaluating various cardinality estimators by their effect on query performance. It is based on ten templates, and contains 8746 unique queries, with almost three million sub-queries. We required such a large scale dataset for these evaluations to address the following limitations in the queries used in previous work [5, 6].

- **Number of tables:** "JOB-Light", the query set used by Kipf et al. [5], only used joins with upto 4 tables. Ortiz et al. generated workloads with joins of upto 6 tables [6]. In contrast, our workload contains ten templates, with the number of tables ranging from 5 to 16.
- **Number of samples per template:** Our dataset contains templates in the style of JOB, with relevant real world interpretations (i.e., they are not just random join graphs), but it uses an automated method [12] to generate the predicate filters, which lets us create hundreds of samples per template class.

Due to computational constraints for collecting ground truth data, we exclude cross-joins when evaluating any of the classifiers. Table 1 summarizes the key properties of each of the templates in this dataset. More details about the query generation process can be found at our github page [12].

Featurization Here, we briefly explain the featurization scheme used for each classifier.

- **Query structure** As described by Kipf et al., [5], each sample (sub-query) is mapped to three input vectors: T_q (one-hot vector for the tables in the query), J_q (one hot vector for the joins in the query), and P_q, which is a mapping of all the predicates in a query to a feature vector.
- **Heuristics** Dutt et al., used heuristic features, such as cardinality estimates from a traditional optimizer to drastically improve performance [4]. Thus, for each query and predicate, we provide the cardinality estimate from PostgreSQL for it as input to the neural network.

Template	Samples	Tables	Sub-queries	Optimal Plans
1a	3000	9	107	624
2a,b,c	1686	11	290	787
3a	1383	10	230	117
4a	516	6	37	10
5a	1014	10	219	101
6a	465	14	1413	203
7a	167	16	3120	115
8a	515	12	469	251

TABLE I: Samples refers to the number of query instances of a template. Sub-queries refers to the number of smaller queries in a single query instance (excluding cross-joins). Optimal Plans refers to the number of unique query plans generated for all sample instances when true cardinalities are provided to the PostgreSQL optimizer.

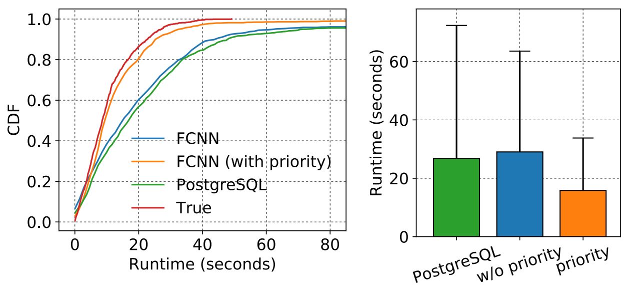
- **Query predicates** For range predicates over continuous columns, we normalize them to $[0, 1]$ using the minimum and maximum values, which has been established as a standard technique [5]. For `IN` clauses over discrete columns, we indicate the presence or absence of the predicate as a 1 or 0. In both cases, we also provide the PostgreSQL selectivity estimate as a feature.

Models We use the default estimates from PostgreSQL and the true cardinalities as baselines for each of our evaluation metrics. The training technique we propose should benefit any gradient-descent based learning model, thus we evaluate two of the state of the art neural network architectures proposed for cardinality estimation.

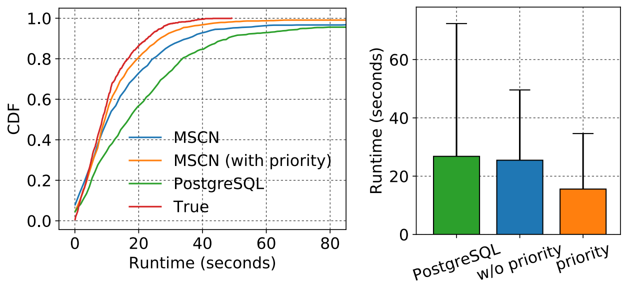
- **FCNN**: This is an extension of the architecture proposed by Dutt et al. [4] that concatenates the individual feature vectors, T_q, J_q, and P_q into an input vector, and uses a two hidden-layer fully connected neural network.
- **MSCN**: Each of the feature vectors, T_q, J_q, and P_q, gets its own module: a one hidden layer fully connected neural network, which is then pooled together into a combined, one-hidden layer, fully connected neural network. For more details, see Kipf et al., [5].

We use hidden layers of size 100 for both the FCNN and MSCN models. We believe that other learned models, such as the ones proposed by Ortiz et al., and Woltmann et al., [2, 6], should also benefit from our approach, and plan to evaluate them in future work.

Methodology We use PostgreSQL for all our experiments, using its exhaustive search algorithm and default cost model to compute the Plan-Error. The evaluation setup and code can be found at our github page [12]. All the results here are presented on the test set, which comprises of half of all the queries in the data set. Each model is trained for twenty epochs. In each epoch of training, the models see almost 1.5 million sub-queries in randomized order — in the non-prioritized case, this would involve seeing every sub-query in the training set. In the prioritized versions, sub-queries will be sampled in a weighted manner, so not every sub-query may be seen in an epoch. For the runtime experiments, we used an



(a) FCNN model



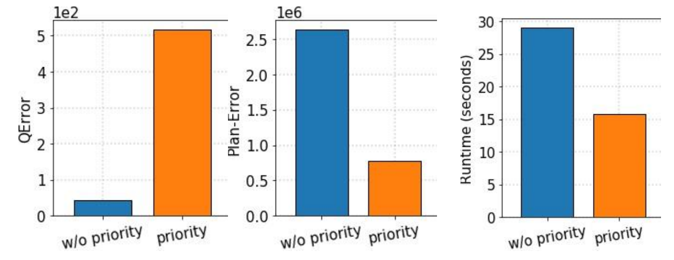
(b) MSCN model

Fig. 1: Cost-guided learning improves the actual query execution time under different models. The bar plots show the mean of query runtimes and the error bar indicate 95 percentile.

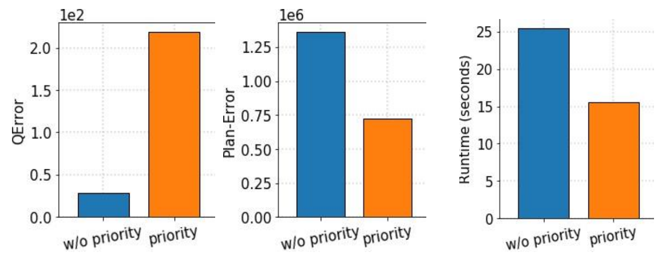
Amazon EC2 instance with a NVME SSD hard drive, and 8GB RAM. To reduce variance, we disabled the parallel operators in PostgreSQL. Two queries in the same template may have the same optimal plan. To reduce the run time overhead, we selected a representative set of 1112 queries of the test set to execute, such that each optimal plan was seen once.

Runtime In Figures 1a and 1b we look at the cumulative distribution function of the runtimes. When true cardinalities are provided, no query runs for over 50 seconds, while in any other case, there is a long tail of much worse performing queries. With the cost-guided training method, there are significantly fewer queries that last over 50 seconds — thus our training method helps in improving the tail performance. This general trend appears to hold for both the FCNN and MSCN architectures. *When using cost-guided learning, query performance improves significantly, both in the mean and the tails.* In particular, the mean query runtime improves by $1.66\times$ for MSCN and $2\times$ for FCNN. The gains at the tail are even larger: for the 10 worst queries (not shown in the figure), there is a more than $6\times$ improvement in runtime. It is also interesting to note that even though each of the models improves the QError over PostgreSQL by many orders of magnitude, in this workload, neither of the models trained without prioritization led to much improvements over PostgreSQL in terms of runtime.

QError and Plan-Error In Figures 2a and 2b we look at the mean of each of the cost criterion we have considered. Surprisingly, models trained with prioritization do much worse in terms of QError, but better on the query optimization goals we care about: improving the Plan-Error and average runtimes



(a) FCNN model



(b) MSCN model

Fig. 2: Comparison of the QError, Plan-Error, and Runtime for the FCNN and the MSCN models

by almost a factor of two. The more structured MSCN model also seems to do better than FCNN, but when we train them with prioritization, their performance on the Plan-Error and runtimes seem to converge to similar values.

IV. Related work

Learned query optimization Query optimization has a long research history [13]. Our work most closely follows the setup used by Ortiz et al., [6]. In comparison, they measured the impact of cardinality estimates on runtimes on a smaller data set of queries, but they did not use any signal besides the estimation errors to train their models. We additionally introduced the notion of Plan-Error, which lets us utilize the query performance in improving our cardinality estimator. Besides this, Kipf et al., Leis et al., and Dutt et al., were other works which introduced central ideas on top of which we further develop our work [4, 5, 14].

Neural networks Using non-uniform weights to select items while training has seen applications in various Machine Learning domains. In Reinforcement Learning, various prioritization schemes for selecting more useful training samples from a replay buffer have been proposed [15, 16]. In Supervised Learning, assigning higher priorities to harder tasks was found to be helpful in multi-task learning systems [17], or when the number of samples from each class are not balanced, such as in heartbeat detection models [18].

V. Conclusions and Future work

We introduced cost-guided learning for cardinality estimation, a technique that helps learned cardinality estimation models "focus" on predictions that have the largest impact on query plans. Experimentally, we show that our cost-guided approach can improve average query runtimes by nearly a factor of two. Surprisingly, our approach resulted in neural networks with increased query performance, despite having lower cardinality prediction accuracy. This suggests that directly minimizing prediction error is not a good training regime for learned cardinality estimators.

In future work, we plan to expand our experimental study to include more recent cardinality estimation models, as well as measuring training time and investigating trends in the Plan-Error metric. There are also several hyper-parameters related to prioritization and training that we have yet to fully evaluate. We also plan on investigating approximate query processing to more quickly gather training data for learned estimators.

Acknowledgement

This research was supported by Intel, Google and Microsoft as part of the MIT Data Systems and AI Lab (DSAIL), and NSF grant CNS-1751009.

References

[1] V. Leis, A. Gubichev, A. Mirchev, P. Boncz, A. Kemper, and T. Neumann, "How good are query optimizers, really?" *Proceedings of the VLDB Endowment*, vol. 9, no. 3, pp. 204–215, 2015.

[2] L. Woltmann, C. Hartmann, M. Thiele, D. Habich, and W. Lehner, "Cardinality estimation with local deep learning models," in *Proceedings of the Second International Workshop on Exploiting Artificial Intelligence Techniques for Data Management*, 2019, pp. 1–8.

[3] Y. Park, S. Zhong, and B. Mozafari, "Quicksel: Quick selectivity learning with mixture models," *arXiv preprint arXiv:1812.10568*, 2018.

[4] A. Dutt, C. Wang, A. Nazi, S. Kandula, V. Narasayya, and S. Chaudhuri, "Selectivity estimation for range predicates using lightweight models," *Proceedings of the VLDB Endowment*, vol. 12, no. 9, pp. 1044–1057, 2019.

[5] A. Kipf, T. Kipf, B. Radke, V. Leis, P. Boncz, and A. Kemper, "Learned cardinalities: Estimating correlated joins with deep learning," *arXiv preprint arXiv:1809.00677*, 2018.

[6] J. Ortiz, M. Balazinska, J. Gehrke, and S. S. Keerthi, "An empirical analysis of deep learning for cardinality estimation," *arXiv preprint arXiv:1905.06425*, 2019.

[7] G. Moerkotte, T. Neumann, and G. Steidl, "Preventing bad plans by bounding the impact of cardinality estimation errors," *Proceedings of the VLDB Endowment*, vol. 2, no. 1, pp. 982–993, 2009.

[8] C. Wu, A. Jindal, S. Amizadeh, H. Patel, W. Le, S. Qiao, and S. Rao, "Towards a learning optimizer for shared clouds," *Proceedings of the VLDB Endowment*, vol. 12, no. 3, pp. 210–222, Nov. 2018.

[9] R. Marcus and O. Papaemmanouil, "Towards a hands-free query optimizer through deep learning," *CIDR*, 2019.

[10] R. Marcus, P. Negi, H. Mao, C. Zhang, M. Alizadeh, T. Kraska, O. Papaemmanouil, and N. Tatbul, "Neo: A learned query optimizer," *Proceedings of the VLDB Endowment*, vol. 12, no. 11, pp. 1705–1718, 2019.

[11] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.

[12] P. Negi and R. Marcus. (2020) Cardinality estimation dataset. https://github.com/parimarjan/learned-cardinalities. [Online;].

[13] P. G. Selinger *et al.*, "Access Path Selection in a Relational Database Management System," in *SIGMOD '79*.

[14] V. Leis, B. Radke, A. Gubichev, A. Kemper, and T. Neumann, "Cardinality estimation done right: Index-based join sampling." in *CIDR*, 2017.

[15] J. Zhai, Q. Liu, Z. Zhang, S. Zhong, H. Zhu, P. Zhang, and C. Sun, "Deep Q-Learning with Prioritized Sampling," in *Neural Information Processing*, ser. Lecture Notes in Computer Science, A. Hirose, S. Ozawa, K. Doya, K. Ikeda, M. Lee, and D. Liu, Eds. Cham: Springer International Publishing, 2016, pp. 13–22.

[16] D. Horgan, J. Quan, D. Budden, G. Barth-Maron, M. Hessel, H. van Hasselt, and D. Silver, "Distributed Prioritized Experience Replay," *arXiv:1803.00933 [cs]*, Mar. 2018.

[17] M. Guo, A. Haque, D.-A. Huang, S. Yeung, and L. Fei-Fei, "Dynamic Task Prioritization for Multitask Learning," in *Proceedings of the European Conference on Computer Vision (ECCV)*, ser. ECCV '18, 2018, pp. 270–287.

[18] A. Sellami and H. Hwang, "A robust deep convolutional neural network with batch-weighted loss for heartbeat classification," *Expert Systems with Applications*, vol. 122, pp. 75–84, May 2019.