Learned Cardinalities: Estimating Correlated Joins with Deep Learning

Andreas Kipf
Technical University of Munich
kipf@in.tum.de

Thomas Kipf
University of Amsterdam
t.n.kipf@uva.nl

Bernhard Radke
Technical University of Munich
radke@in.tum.de

Viktor Leis
Technical University of Munich
leis@in.tum.de

Peter Boncz
Centrum Wiskunde & Informatica
boncz@cwi.nl

Alfons Kemper
Technical University of Munich
kemper@in.tum.de

ABSTRACT

We describe a new deep learning approach to cardinality estimation. MSCN is a multi-set convolutional network, tailored to representing relational query plans, that employs set semantics to capture query features and true cardinalities. MSCN builds on sampling-based estimation, addressing its weaknesses when no sampled tuples qualify a predicate, and in capturing join-crossing correlations. Our evaluation of MSCN using a real-world dataset shows that deep learning significantly enhances the quality of cardinality estimation, which is the core problem in query optimization.

1 INTRODUCTION

Query optimization is fundamentally based on cardinality estimation. To be able to choose between different plan alternatives, the query optimizer must have reasonably good estimates for intermediate result sizes. It is well known, however, that the estimates produced by all widely-used database systems are routinely wrong by orders of magnitude—causing slow queries and unpredictable performance. The biggest challenge in cardinality estimation are join-crossing correlations [11]. For example, in the Internet Movie Database (IMDB), French actors are more likely to participate in romantic movies than actors of other nationalities.

The question of how to better deal with this is an open area of research. One state-of-the-art proposal in this area is Index-Based Join Sampling (IBJS) [12] that addresses this problem by probing qualifying base table samples against existing index structures. However, like other sampling-based techniques, IBJS fails when there are no qualifying samples to start with (i.e., under selective base table predicates) or when no suitable indexes are available. In such cases, these techniques usually fall back to an "educated" guess—causing large estimation errors.

The past decade has seen the widespread adoption of machine learning (ML), and specifically neural networks (deep learning), in many different applications and systems. The database community also has started to explore how machine learning can be leveraged within data management systems. Recent research therefore investigates ML for classical database problems like parameter tuning [2], query optimization [9, 17, 20], and even indexing [8].

We argue that machine learning is a highly promising technique for solving the cardinality estimation problem. Estimation can be formulated as a supervised learning problem, with the input being query features and the output being the estimated cardinality. In contrast to other problems where machine learning has been proposed like index structures [8] and join ordering [17], the current techniques based on basic per-table statistics are not very good. In other words, an estimator based on machine learning does not have to be perfect, it just needs to be better than the current, inaccurate baseline. Furthermore, the estimates produced by a machine learning model can directly be leveraged by existing, sophisticated enumeration algorithms and cost models without requiring any other changes to the database system.

In this paper, we propose a deep learning-based approach that learns to predict (join-crossing) correlations in the data and addresses the aforementioned weak spot of sampling-based techniques. Our approach is based on a specialized deep learning model called multi-set convolutional network (MSCN) allowing us to express query features using sets (e.g., both $(A \bowtie B) \bowtie C$ and $A \bowtie (B \bowtie C)$ are represented as $\{A, B, C\}$). Thus, our model does not waste any capacity for memorizing different permutations (all having the same cardinality but different costs) of a query's features, which results in smaller models and better predictions. The join enumeration and cost model are purposely left to the query optimizer.

We evaluate our approach using the real-world IMDB dataset [11] and show that our technique is more robust than sampling-based techniques and even is competitive in the sweet spot of these techniques. This is achieved using a (configurable) low footprint size of about 3 MB (whereas the sampling-based techniques have access to indexes covering the entire database). These results are highly promising and indicate that ML might indeed be the right hammer for the decades-old cardinality estimation job.

2 RELATED WORK

Deep learning has been applied to query optimization by three recent papers [9, 17, 20] that formulate join ordering as a *reinforcement learning* problem and use ML to find *query plans*. This work, in contrast, applies *supervised learning* to solve *cardinality estimation* in isolation. This focus is motivated by the fact that modern join enumeration algorithms can find the optimal join order for queries with dozens of relations [19]. Cardinality estimation, on the other hand, has been called the "Achilles heel" of query optimization [15] and causes most of its performance issues [11].

Twenty years ago the first approaches to use neural networks for cardinality estimation where published for UDF predicates [10]. Also, regression-based models have been used before for cardinality estimation [1]. A semi-automatic alternative for explicit machine

learning was presented in [16], where the feature space is partitioned using decision trees and for each split a different regression model was learned. These early approaches did not use deep learning nor included features derived from statistics, such as our sample-based bitmaps, which encode exactly which sample tuples were selected (and we therefore believe to be good starting points for learning correlations). The same holds for approaches that used machine learning to predict overall resource consumption: running time, memory footprint, I/O, network traffic [6, 13], although these models did include course-grained features (the estimated cardinality) based on statistics into the features. Liu et al. [14] used modern ML for cardinality estimation, but did not focus on joins, which are the key estimation challenge [11].

Our approach builds on sampling-based estimation by including bitmaps or cardinalities derived from samples into the training signal. Most sampling proposals create per-table samples/sketches and try to combine them intelligently in joins [3, 5, 22, 23]. While these approaches work well for single-table queries, they do not capture join-crossing correlations and are vulnerable to the 0-tuple problem (cf. Section 4.2). Recent work by Müller et al. [18] aims to reduce the 0-tuple problem for conjunctive predicates (albeit at high computational cost), but still cannot capture the basic case of a single predicate giving zero results. Our reasonably good estimates in 0-tuple situations make MSCN improve over sampling, including even the idea of estimation on materialized join samples (join synopses [21]), which still would not handle 0-tuple situations.

3 LEARNED CARDINALITIES

From a high-level perspective, applying machine learning to the cardinality estimation problem is straightforward: after training a supervised learning algorithm with query/output cardinality pairs, the model can be used as an estimator for other, unseen queries. There are, however, a number of challenges that determine whether the application of machine learning will be successful: the most important question is how to represent queries ("featurization") and which supervised learning algorithm should be used. Another issue is how to obtain the initial training dataset ("cold start problem"). In the remainder of this section, we first address these questions before discussing a key idea of our approach, which is to featurize information about materialized samples.

3.1 Set-Based Query Representation

We represent a query $q \in Q$ as a collection (T_q, J_q, P_q) of a set of tables $T_q \subset T$, a set of joins $J_q \subset J$ and a set of predicates $P_q \subset P$ participating in the specific query q. T, J, and P describe the sets of all available tables, joins, and predicates, respectively.

Each table $t \in T$ is represented by a unique *one-hot* vector v_t (a binary vector of length $|T|$ with a single non-zero entry, uniquely identifying a specific table) and optionally the number of qualifying base table samples or a bitmap indicating their positions. Similarly, we featurize joins $j \in J$ with a unique one-hot encoding. For predicates of the form (col, op, val), we featurize columns col and operators op using a categorical representation with respective unique one-hot vectors, and represent val as a normalized value $\in [0, 1]$, normalized using the minimum and maximum values of the respective column.

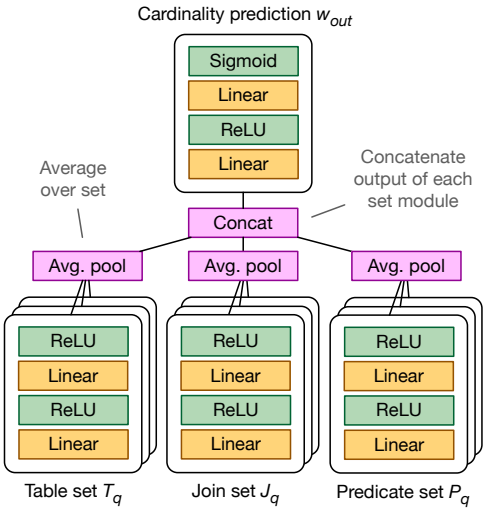

Figure 1: Architecture of our multi-set convolutional network. Tables, joins, and predicates are represented as separate modules, comprised of one two-layer neural network per set element with shared parameters. Module outputs are averaged, concatenated, and fed into a final output network.

Applied to the query representation (T_q, J_q, P_q), our MSCN model (cf. Figure 1) takes the following form:

$$\text{Table module:} \quad w_T = \frac{1}{|T_q|} \sum_{t \in T_q} \text{MLP}_T(v_t)$$

$$\text{Join module:} \quad w_J = \frac{1}{|J_q|} \sum_{j \in J_q} \text{MLP}_J(v_j)$$

$$\text{Predicate module:} \quad w_P = \frac{1}{|P_q|} \sum_{p \in P_q} \text{MLP}_P(v_p)$$

$$\text{Merge \& predict:} \quad w_{\text{out}} = \text{MLP}_{\text{out}}([w_T, w_J, w_P])$$

Figure 2 shows an example of a featurized query.

3.2 Model

Standard deep neural network architectures such as convolutional neural networks (CNNs), recurrent neural networks (RNNs), or simple multi-layer perceptrons (MLPs) are not directly applicable to this type of data structure, and would require *serialization*, i.e., conversion of the data structure to an ordered sequence of elements. This poses a fundamental limitation, as the model would have to spend capacity to learn to discover the symmetries and structure of the original representation. For example, it would have to learn to discover boundaries between different sets in a data structure consisting of multiple sets of different size, and that the order of elements in the serialization of a set is arbitrary.

Given that we know the underlying structure of the data *a priori*, we can bake this information into the architecture of our deep learning model and effectively provide it with an *inductive bias* that facilitates generalization to unseen instances of the same structure, e.g., combinations of sets with a different number of elements not seen during training.

Here, we introduce the *multi-set convolutional network* (MSCN) model. Our model architecture is inspired by recent work on *set*

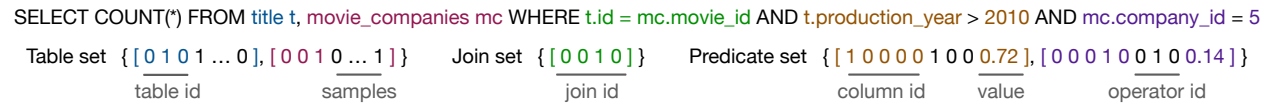

Figure 2: Query featurization as sets of feature vectors.

convolutions [24], a neural network module for operating on sets. A set convolution module rests on the observation that any function $f(S)$ on a set S that is permutation invariant to the elements in S can be decomposed into the form $\rho[\sum_{x \in S} \phi(x)]$ with appropriately chosen functions ρ and ϕ. For a more formal discussion and proof of this property, we refer to Zaheer et al. [24]. We choose simple fully-connected multi-layer neural networks (MLPs) to parameterize the functions ρ and ϕ and rely on their function approximation properties [4] to learn flexible mappings $f(S)$ for arbitrary sets S.

Our query representation consists of a collection of *multiple* sets, which motivates the following choice for our MSCN model architecture: for every set S, we learn a set-specific, per-element neural network $\text{MLP}_S(v_s)$, i.e., applied on every feature vector v_s for every element $s \in S$ individually. The final representation w_S for this set is then given by the average over the individual transformed representations of its elements, i.e., $w_S = 1/|S| \sum_{s \in S} \text{MLP}_S(v_s)$. We choose an average (instead of, e.g., a simple sum) to ease generalization to different numbers of elements in the set S, as otherwise the overall magnitude of the signal would vary depending on the number of elements in S.

Finally, we merge the individual set representations by concatenation and subsequently pass them through a final output MLP: $w_{\text{out}} = \text{MLP}_{\text{out}}([w_{S_1}, w_{S_2}, \ldots, w_{S_N}])$, where N is the total number of sets and $[\cdot, \cdot]$ denotes vector concatenation. Note that this representation includes the special case where each set representation w_S is transformed by a subsequent individual output function (as required by the original theorem in [24]). One could alternatively process each w_S individually first and only later merge and pass through another MLP. We decided to merge both steps into a single computation for computational efficiency.

Unless otherwise noted, all MLP modules are two-layer fully-connected neural networks with $\text{ReLU}(x) = \max(0, x)$ activation functions. For the output MLP, we use a $\text{sigmoid}(x) = 1/(1 + \exp(-x))$ activation function for the last layer instead and only output a scalar, so that $w_{\text{out}} \in [0, 1]$. All other representation vectors w_T, w_J, w_P, and hidden layer activations of the MLPs are chosen to be vectors of dimension d, where d is a hyperparameter, optimized on a separate validation set via grid search.

We normalize the target cardinalities c_{target} as follows: we first take the logarithm to more evenly distribute target values, and then normalize to the interval $[0, 1]$ using the minimum and maximum value after logarithmization obtained from the training set. The normalization is invertible, so we can recover the unnormalized cardinality from the prediction $w_{\text{out}} \in [0, 1]$ of our model.

We train our model to minimize the mean *q-error* q ($q \geq 1$). The q-error is the factor between an estimate and the true cardinality (or vice versa). We further explored using mean-squared error and geometric mean q-error as objectives (cf. Section 4.7). We make use of the Adam [7] optimizer for training.

3.3 Generating Training Data

One key challenge of all learning-based algorithms is the "cold start problem", i.e., how to train the model before having concrete information about the query workload. Our approach is to obtain an initial training corpus by generating random queries based on schema information and drawing literals from actual values in the database.

A training sample consists of table identifiers, join predicates, base table predicates, and the true cardinality of the query result. To avoid a combinatorial explosion, we only generate queries with up to two joins and let the model generalize to more joins. Our query generator first uniformly draws the number of joins $|J_q|$ ($0 \leq |J_q| \leq 2$) and then uniformly selects a table that is referenced by at least one table. For $|J_q| > 0$, it then uniformly selects a new table that can join with the current set of tables (initially only one), adds the corresponding join edge to the query and (overall) repeats this process $|J_q|$ times. For each base table t in the query, it then uniformly draws the number of predicates $|P_q^t|$ ($0 \leq |P_q^t| \leq$ num non-key columns). For each predicate, it uniformly draws the predicate type (=, <, or >) and selects a literal (an actual value) from the corresponding column. We configured our query generator to only generate *unique* queries. We then execute these queries to obtain their true result cardinalities, while skipping queries with empty results. Using this process we obtain the initial training set for our model.

3.4 Enriching the Training Data

A key idea of our approach is to enrich the training dataset with information about *materialized* base table samples. For each table in a query, we evaluate the corresponding predicates on a materialized sample and annotate the query with the *number* of qualifying samples s ($0 \leq s \leq 1000$ for 1000 materialized samples) for this table. We perform the same steps for an (unseen) test query at estimation time allowing the ML model to utilize this knowledge.

We even take this idea one step further and annotate each table in a query with the *positions* of the qualifying samples represented as bitmaps. As we show in Section 4, adding this feature has a significant impact on our join estimates since the ML model can now learn what it means if a certain sample qualifies (e.g., there might be some samples that usually have many join partners). In other words, the model can learn to use the patterns in the bitmaps to predict output cardinalities.

3.5 Training and Inference

Building our model involves three steps: i) generate random (uniformly distributed) queries using schema and data information, ii) execute queries to annotate them with their true cardinalities and information about qualifying materialized base table samples, and iii) feed this training data into an ML model. All of these steps are performed on an immutable snapshot of the database.

number of joins	0	1	2	3	4	overall
synthetic	1636	1407	1957	0	0	5000
scale	100	100	100	100	100	500
JOB-light	0	3	32	23	12	70

Table 1: Distribution of joins.

To predict the cardinality of a query, the query first needs to be transformed into its feature representation (cf. Section 3.1). Inference itself involves a certain number of matrix multiplications, and (optionally) querying materialized base table samples (cf. Section 3.4). Training the model with more query samples does not increase the prediction time. In that respect, the inference speed is largely independent from the quality of the predictions. This is in contrast to purely sampling-based approaches that can only increase the quality of their predictions by querying more samples.

4 EVALUATION

We evaluate our approach using the IMDB dataset which contains many correlations and therefore proves to be very challenging for cardinality estimators [11]. We use three different query workloads: i) a *synthetic* workload generated by the same query generator as our training data (using a different random seed) with 5,000 *unique* queries containing both (conjunctive) equality and range predicates on non-key columns with zero to two joins, ii) another synthetic workload *scale* with 500 queries designed to show how the model generalizes to more joins, and iii) *JOB-light*, a workload derived from the Join Order Benchmark (JOB) [11] containing 70 of the original 113 queries. In contrast to JOB, JOB-light does not contain any predicates on strings nor disjunctions and only contains queries with one to four joins. Most queries have equality predicates on dimension table attributes. The only range predicate is on `production_year`. Table 1 shows the distribution of queries with respect to the number of joins in the three test workloads. The non-uniform distribution in the synthetic workload is caused by our elimination of duplicate queries.

As competitors we use PostgreSQL version 10.3, Random Sampling (RS), and Index-Based Join Sampling (IBJS) [12]. RS executes base table predicates on materialized samples to estimate base table cardinalities and assumes independence for estimating joins. If there are no qualifying samples for a conjunctive predicate, it tries to evaluate the conjuncts individually and eventually falls back to using the number of distinct values of the corresponding columns to estimate the selectivity. IBJS represents the state-of-the-art for estimating joins and probes qualifying base table samples against existing index structures. Our IBJS implementation uses the same fallback mechanism as RS.

We train and test our model on a NVIDIA GeForce GTX 1050 Ti (4 GB GDDR5) GPU using the PyTorch framework. Unless stated otherwise, we use 100,000 random queries with zero to two joins and 1,000 materialized samples as training data (cf. Section 3.3). We split the training data into 90% training and 10% validation samples.

4.1 Estimation Quality

Figure 3 shows the q-error of MSCN compared to our competitors. While PostgreSQL's errors are more skewed towards the positive

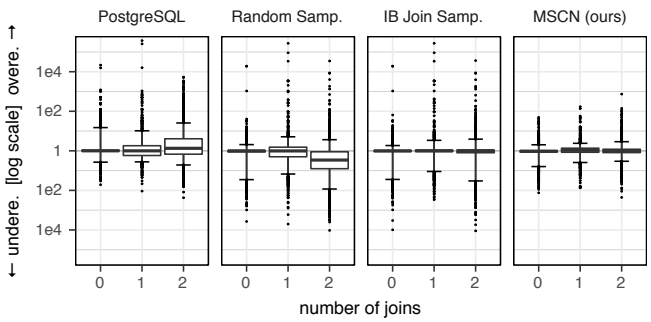

Figure 3: Estimation errors on the synthetic workload with 1,000 materialized samples. The box boundaries are at the 25th/75th percentiles and the horizontal "whisker" lines mark the 95th percentiles.

	median	90th	95th	99th	max	mean
PostgreSQL	1.69	9.57	23.9	465	373901	154
Random Samp.	1.89	19.2	53.4	587	272501	125
IB Join Samp.	**1.09**	9.93	33.2	295	272514	118
MSCN (ours)	1.19	**3.50**	**7.22**	**34.9**	**735**	**2.88**

Table 2: Estimation errors on the synthetic workload.

spectrum, RS tends to underestimate joins, which stems from the fact that it assumes independence. IBJS performs extremely well in the median and 75th percentile but (like RS) suffers from empty base table samples. MSCN is competitive with IBJS in the median while being significantly more robust. Considering that IBJS is using much more data—in the form of large primary and foreign key indexes—in contrast to the very small state MSCN is using (less than 3 MB), MSCN captures (join-crossing) correlations reasonably well and does not suffer as much from 0-tuple situations (cf. Section 4.2). To provide more details, we also show the median, percentiles, maximum, and mean q-errors in Table 2. While IBJS provides the best median estimates, MSCN outperforms the competitors by up to two orders of magnitude at the end of the distribution.

4.2 0-Tuple Situations

Purely sampling-based approaches suffer from empty base table samples (0-tuple situations) which can occur under selective predicates. While this situation can be mitigated using, e.g., more samples or employing more sophisticated—yet still sampling-based—techniques (e.g., [18]), it remains inherently difficult to address by these techniques. In this experiment, we show that deep learning, and MSCN in particular, can handle such situations fairly well.

In fact, 376 (22%) of the 1636 base table queries in the synthetic workload have empty samples. We will use this subset of queries to illustrate how MSCN deals with situations where it *cannot* build upon (runtime) sampling information (i.e., all bitmaps only contain zeros). We also include Random Sampling (which uses the same random seed—i.e., the same set of materialized samples as MSCN) and PostgreSQL in this experiment.

The results, shown in Table 3, demonstrate that MSCN addresses the weak spot of purely sampling-based techniques and therefore would complement them well.

	median	90th	95th	99th	max	mean
PostgreSQL	4.78	62.8	107	1141	21522	133
Random Samp.	9.13	80.1	173	993	19009	147
MSCN	**3.17**	**17.2**	**28.8**	**55.5**	**96.2**	**7.20**

Table 3: Estimation errors of 376 base table queries with empty samples in the synthetic workload.

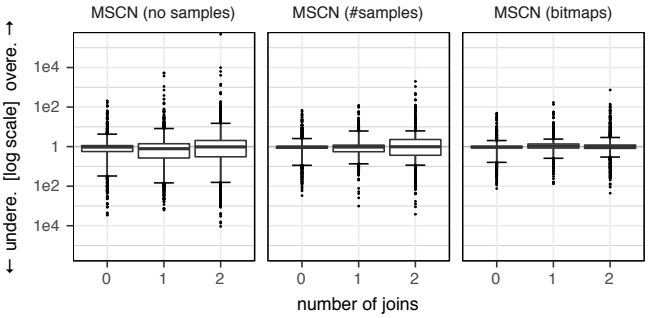

Figure 4: Estimation errors on the synthetic workload with different model variants.

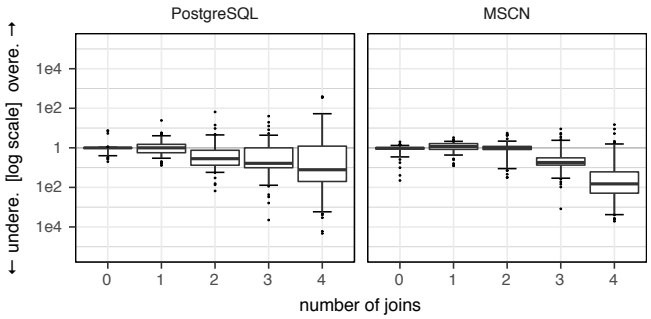

Figure 5: Estimation errors on the scale workload showing how MSCN generalizes to queries with more joins.

	median	90th	95th	99th	max	mean
PostgreSQL	7.93	164	1104	2912	3477	174
Random Samp.	11.5	198	4073	22748	23992	1046
IB Join Samp.	**1.59**	150	3198	14309	15775	590
MSCN	3.56	**31.5**	**77.6**	**415**	**676**	**27.5**

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

4.3 Removing Model Features

Next, we highlight the contributions of individual model features to the prediction quality (cf. Figure 4). MSCN (no samples) is the model without any (runtime) sampling features, MSCN (#samples) represents the model with one cardinality (i.e., the number of qualifying samples) per base table, and MSCN (bitmaps) denotes the full model with one bitmap per base table.

MSCN (no samples) produces reasonable estimates, purely relying on (inexpensive to obtain) query features. Adding sample cardinality information to the model improves both base table and join estimates. Replacing samples cardinalities with bitmaps does not have much impact on base table predictions (0 joins) but significantly improves the prediction quality for joins. This shows that the model can capture the correlations between the bitmaps and use this information to effectively address join-crossing correlations.

4.4 Generalizing to More Joins

To estimate a larger query, one can of course break the query down into smaller sub queries, estimate them individually using the model, and combine their selectivities. However, this means that we would need to assume independence between two sub queries which is known to deliver poor estimates with real datasets such as IMDB (cf. join estimates of Random Sampling in Section 4).

The question that we want to answer in this experiment is how MSCN can generalize to queries with more joins than it was trained on. For this purpose, we use the *scale* workload with 500 queries with zero to four joins (100 queries each). Recall that we trained the model only with queries between zero and two joins. Thus, this experiment shows how the model can estimate queries with three and four joins *without* having seen such queries in the training set (cf. Figure 5). From two to three joins, the 95th percentile q-error increases from 11.3 to 34.3. To give a point of reference, PostgreSQL has a 95th percentile q-error of 78.0 for the same queries. And finally, with four joins, MSCN's 95th percentile q-error increases further to 2,386 (PostgreSQL: 4,077).

4.5 JOB-light

To show how MSCN generalizes to a workload that was not generated by our query generator, we use JOB-light.

Table 4 shows the estimation errors. Recall that most queries in JOB-light have equality predicates on dimension table attributes. Considering that MSCN was trained with a uniform distribution between =, <, and > predicates, it performs reasonably well in this workload suggesting that it can take advantage of the range predicates it has seen during training to predict the effect of equality predicates. We verified this assumption with a modified training workload restricted to only equality predicates on dimension table attributes, which lead to a lower prediction quality. In summary, this experiment shows that MSCN can generalize to workloads with distributions different from the initial training data.

4.6 Model Costs

Next, we analyze the training, inference, and space costs of MSCN. By default, we train our model with 100 epochs (which is the number of iterations over the training set). The model requires fewer than 25 iterations (over the 90,000 training queries) to converge to a mean q-error of around 3 on the validation set.

The prediction time of our model is in the order of a few milliseconds, including the overhead introduced by the PyTorch framework. In theory (neglecting the PyTorch overhead), a prediction using a deep learning model (as stated earlier) is dominated by matrix multiplications which can be accelerated using modern GPUs. We thus expect performance-tuned implementations of our model to achieve very low prediction latencies. Since we incorporate sampling information, the end-to-end prediction time will be in the same order of magnitude as that of (per-table) sampling techniques.

The size of our model (when serialized to disk) is 1.1 MB, 1.6 MB, and 2.6 MB for MSCN (no samples), MSCN (#samples), and MSCN (bitmaps), respectively.

4.7 Further Experiments

We ran more experiments which we will only briefly mention here for space reasons. Besides optimizing the mean q-error, we also explored using mean-squared error and geometric mean q-error as optimization goals. Mean-squared error would optimize the *absolute* differences between the predicted and true cardinalities. Since we are more interested in minimizing the factor between the predicted and the true cardinalities (q-error) and use this metric for our evaluation, optimizing the q-error directly yielded better results. Optimizing the geometric mean of the q-error makes the model put less emphasis on heavy outliers (that would lead to large errors). While this approach looked promising at first, it turned out to be not as reliable as optimizing the mean q-error. We also experimented with 10,000 instead of 1,000 materialized samples but it turned out to be not as important: using 10× as many samples only decreased the 99th percentile and the mean q-error on the synthetic workload by 1.5× and 2×, respectively. Further, we tuned the hyperparameters of our model, including the batch size, number of hidden units, and the learning rate. More hidden units means larger model sizes and increased training and prediction costs with the upside of allowing the model to capture more data. We found a batch size of 64 samples, 256 hidden units, and a learning rate of 0.001 to perform well on the validation data.

5 CONCLUSION AND FUTURE WORK

We have introduced a new approach to cardinality estimation based on MSCN, a new deep learning model. We have trained MSCN with generated queries, uniformly distributed within a constrained search space. We have shown that it can learn (join-crossing) correlations and that it addresses the weak spot of sampling-based techniques, which is when no samples qualify. Our current model is a first step towards reliable ML-based cardinality estimation, which can be extended into multiple dimensions.

We have shown that MSCN can (to some extent) generalize to queries with more joins. Nevertheless, generalizing to queries that are not in the vicinity of the training data remains challenging. Of course, our model can be trained with queries from an actual workload or their structures. This would allow us to focus on the relevant joins and predicates. Another idea would be to *adaptively* generate training samples: based on the error distribution of queries in the validation set, we could generate new training samples that shine more light on difficult parts of the schema.

Currently, our model can only estimate queries with predicate types that it has seen during training. Complex predicates, such as LIKE or disjunctions, are not yet supported since we currently do not represent them in the model. An idea to allow for any complex predicate would be to purely rely on the sampling bitmaps in such cases. To increase the likelihood for qualifying samples, we could additionally use one bitmap per predicate. For example, for a query with two conjunctive base table predicates, we would have one bitmap for each predicate, and another bitmap representing the conjunction. We have already shown that MSCN can use the correlations between the bitmaps to make better predictions. We expect that it would significantly benefit from the patterns in these additional bitmaps. In addition, to be more robust in 0-tuple situations, we could featurize information from histograms (e.g., using bitmaps).

Another application of our set-based model is the prediction of the number of unique values in a column or in a combination of columns (i.e., estimating the result size of a group-by operator). This is another hard problem where current approaches achieve undesirable results and where machine learning seems promising.

6 ACKNOWLEDGEMENTS

This work has been partially supported by the German Federal Ministry of Education and Research (BMBF) grant `01IS12057` (FASTDATA). It is further part of the TUM Living Lab Connected Mobility (TUM LLCM) project and has been funded by the Bavarian Ministry of Economic Affairs, Energy and Technology (StMWi) through the Center Digitisation.Bavaria, an initiative of the Bavarian State Government. T.K. acknowledges funding by SAP SE.

REFERENCES

[1] M. Akdere, U. Çetintemel, M. Riondato, E. Upfal, and S. B. Zdonik. Learning-based query performance modeling and prediction. In *ICDE*, pages 390–401, 2012.
[2] D. V. Aken, A. Pavlo, G. J. Gordon, and B. Zhang. Automatic database management system tuning through large-scale machine learning. In *SIGMOD*, 2017.
[3] Y. Chen and K. Yi. Two-level sampling for join size estimation. In *SIGMOD*, 2017.
[4] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4), 1989.
[5] C. Estan and J. F. Naughton. End-biased samples for join cardinality estimation. In *ICDE*, 2006.
[6] A. Ganapathi, H. A. Kuno, U. Dayal, J. L. Wiener, A. Fox, M. I. Jordan, and D. A. Patterson. Predicting multiple metrics for queries: Better decisions enabled by machine learning. In *ICDE*, pages 592–603, 2009.
[7] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv:1412.6980*, 2014.
[8] T. Kraska, A. Beutel, E. H. Chi, J. Dean, and N. Polyzotis. The case for learned index structures. In *SIGMOD*, 2018.
[9] S. Krishnan, Z. Yang, K. Goldberg, J. Hellerstein, and I. Stoica. Learning to optimize join queries with deep reinforcement learning. *arXiv:1808.03196*, 2018.
[10] M. S. Lakshmi and S. Zhou. Selectivity estimation in extensible databases - A neural network approach. In *VLDB*, pages 623–627, 1998.
[11] V. Leis, A. Gubichev, A. Mirchev, P. Boncz, A. Kemper, and T. Neumann. How good are query optimizers, really? *PVLDB*, 9(3), 2015.
[12] V. Leis, B. Radke, A. Gubichev, A. Kemper, and T. Neumann. Cardinality estimation done right: Index-based join sampling. In *CIDR*, 2017.
[13] J. Li, A. C. König, V. R. Narasayya, and S. Chaudhuri. Robust estimation of resource consumption for SQL queries using statistical techniques. *PVLDB*, 5(11):1555–1566, 2012.
[14] H. Liu, M. Xu, Z. Yu, V. Corvinelli, and C. Zuzarte. Cardinality estimation using neural networks. In *CASCON*, 2015.
[15] G. Lohman. Is query optimization a solved problem? http://wp.sigmod.org/?p=1075, 2014.
[16] T. Malik, R. C. Burns, and N. V. Chawla. A black-box approach to query cardinality estimation. In *CIDR*, pages 56–67, 2007.
[17] R. Marcus and O. Papaemmanouil. Deep reinforcement learning for join order enumeration. In *International Workshop on Exploiting Artificial Intelligence Techniques for Data Management*, 2018.
[18] M. Müller, G. Moerkotte, and O. Kolb. Improved selectivity estimation by combining knowledge from sampling and synopses. *PVLDB*, 11(9):1016–1028, 2018.
[19] T. Neumann and B. Radke. Adaptive optimization of very large join queries. In *SIGMOD*, 2018.
[20] J. Ortiz, M. Balazinska, J. Gehrke, and S. S. Keerthi. Learning state representations for query optimization with deep reinforcement learning. In *Workshop on Data Management for End-To-End Machine Learning*, 2018.
[21] V. Poosala and Y. E. Ioannidis. Selectivity estimation without the attribute value independence assumption. In *VLDB*, 1997.
[22] D. Vengerov, A. C. Menck, M. Zaït, and S. Chakkappen. Join size estimation subject to filter conditions. *PVLDB*, 8(12):1530–1541, 2015.
[23] W. Wu, J. F. Naughton, and H. Singh. Sampling-based query re-optimization. In *SIGMOD*, pages 1721–1736, 2016.
[24] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov, and A. J. Smola. Deep sets. In *Advances in Neural Information Processing Systems*, 2017.