Cardinality Estimation with Local Deep Learning Models

Lucas Woltmann, Claudio Hartmann, Maik Thiele, Dirk Habich, and Wolfgang Lehner
firstname.lastname@tu-dresden.de
Technische Universität Dresden
Dresden, Germany

ABSTRACT

Cardinality estimation is a fundamental task in database query processing and optimization. Unfortunately, the accuracy of traditional estimation techniques is poor resulting in non-optimal query execution plans. With the recent expansion of machine learning into the field of data management, there is the general notion that data analysis, especially neural networks, can lead to better estimation accuracy. Up to now, all proposed neural network approaches for the cardinality estimation follow a global approach considering the whole database schema at once. These global models are prone to sparse data at training leading to misestimates for queries which were not represented in the sample space used for generating training queries. To overcome this issue, we introduce a novel local-oriented approach in this paper, therefore the local context is a specific sub-part of the schema. As we will show, this leads to better representation of data correlation and thus better estimation accuracy. Compared to global approaches, our novel approach achieves an improvement by two orders of magnitude in accuracy and by a factor of four in training time performance for local models.

CCS CONCEPTS

• **Information systems** → **Query optimization**; • **Computing methodologies** → **Neural networks**; *Supervised learning by regression*; *Ensemble methods.*

aiDM'19, July 5, 2019, Amsterdam, Netherlands

ACM ISBN 978-1-4503-6802-5/19/07…$15.00
https://doi.org/10.1145/3329859.3329875

ACM Reference Format:
Lucas Woltmann, Claudio Hartmann, Maik Thiele, Dirk Habich, and Wolfgang Lehner. 2019. Cardinality Estimation with Local Deep Learning Models. In *International Workshop on Exploiting Artificial Intelligence Techniques for Data Management (aiDM'19), July 5, 2019, Amsterdam, Netherlands.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3329859.3329875

1 INTRODUCTION

Query optimization is still an important challenge due to ever-increasing data sizes, whereby most query optimization techniques are cost-based [2, 19]. In this cost-based approach, cardinality estimation plays a dominant role with the task to approximate the number of returned tuples for every query operator within a query execution plan [2, 19, 21]. These estimations are used within different optimization techniques for various decisions such as determining the right join order [3], choosing the optimal operator variant [27], or finding the optimal placement within a heterogeneous hardware environment [9, 10]. For this reason, it is important to have cardinality estimations with high accuracy.

Unfortunately, most traditional estimation approaches, which are based on statistical models with strong assumptions, are not accurate enough [15]. Here, the main critical assumptions are uniformity and data independence [19]. For example, the color *red* is usually uniformly distributed over all car brands but its distribution for the manufacturer *Ferrari* is rather skewed since most of them are red. In this case, the color and the manufacturer are highly correlated in non-uniform way leading to erroneous cardinality estimates using traditional approaches.

A promising way to overcome these limitations is the use of machine learning, including neural networks, for cardinality estimation [11, 16, 17, 23]. Here, the main assumption is that a sufficiently deep neural network can model the very complex data dependencies and correlations. In this regard, available techniques are based on a *global approach* by creating a single neural network (*global model*) over the entire database schema. This concept is detailed in Figure 1, where Tables `R`, `S`, `T`, `U`, `V`, and `W` are a complete schema

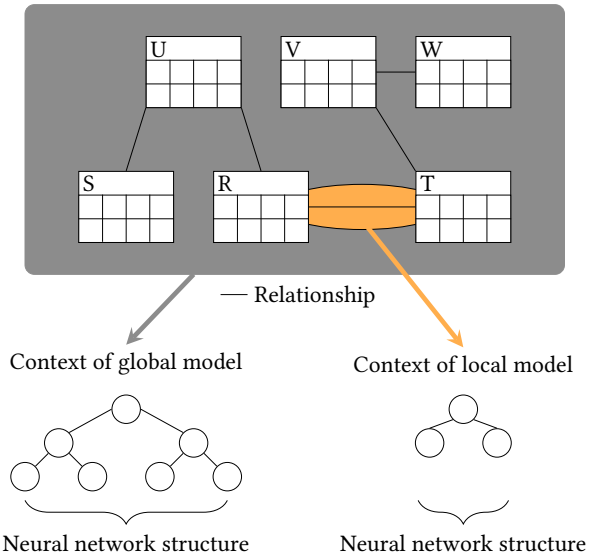

Figure 1: Global vs. local approach.

in a database. The resulting global model for cardinality estimation can become large in structure to model as many aspects of the schema as possible and arbitrary cardinality estimates are possible. The global model is trained on sampled queries from the entire schema. However, this leads to sparse query sampling because even for a limited number of tables and predicates, the number of possible training queries becomes extremely large. For example, our six tables with five columns each containing 1,000 possible values and three possible predicate operators (<, =, >) would generate a sample space of $(2^6 - 1) \cdot 2^5 \cdot 1{,}000 \cdot 3 = 6{,}048{,}000$ queries.

Our Contribution and Outline

To tackle this issue, we propose a cardinality estimation technique with a focus on smaller neural network structures in this paper. We call this a *local* approach because each *local model* focuses on a small part of the schema instead of the whole schema. An example for a model on a sub-part of a schema (i.e. a single join) is depicted in Figure 1. The schema sub-part for a local model can be any number and combination of joins. That means, each local model is always specialized to a specific schema sub-part. Without loss of generalization, we focus on equi-joins in this paper. The advantage of our local approach is that the query sampling gets less sparse. Given our example from before concentrating on the join between **R** and **T**, we get a sample space of $(2^2 - 1) \cdot 2^5 \cdot 1000 \cdot 3 = 288{,}000$ queries. Thus, the coverage of a sample would increase because the same amount of sampled queries would be less sparse for the local sample space. If we sample 100,000 queries in both sample spaces, we cover ca. 1.6% of the global sample space but ca. 35% of the local sample space. Sampling 100,000 queries needs the same amount of time in both scenarios but the coverage of seen queries with different cardinalities is higher for our local approach. This gives a learned model the possibility to generalize its prediction making it more accurate.

In detail, we make the following contributions in this paper: The contributions of our work are the following:

- In Section 2, we outline the current research context of our approach.
- We introduce an approach for learned cardinalities with local models, i.e. neural networks in Section 3.
- Section 3 also provides details on our vectorization process transforming SQL queries into numerical vectors to be utilizable by the neural network as input.
- We provide a comprehensive evaluation in Section 4. In particular, we show that our *local* approach shows an accuracy improvement by two orders of magnitude and a speed-up by three orders of magnitude for the forward pass compared to a global model.

Finally, we close the paper with an outlook onto our future research in Section 5 and a brief summary in Section 6.

2 RELATED WORK

Traditional approaches of cardinality estimation in relational data management systems (RDBMS) rely on statistics that include one-dimensional equi-depth or equi-width histograms on each column in a table [5–7, 25], a list of most frequent values and their frequencies, the number of distinct values, and min and max values [24]. However, the main assumptions of uniformity and data independence have shown to be a major problem for these statistical approaches leading to non-optimal query execution plans in the end [15, 26].

To solve this issue in a more sophisticated way, there have been works on formulating cardinality estimation as a supervised learning problem [11, 14, 16]. On the one hand, the authors in [16] proposed learned cardinality estimators for single tables only. On the other hand, Kipf et al. introduced a universal approach called multi-set convolutional neural network (MSCN) which models cardinality estimation as a global model. The neural network processes three inputs from an SQL query independently. These are the used tables, the join keys, and the chosen predicates on the used tables. Additionally, MSCN uses samples of the first 1,000 rows of a query as a bitmap given the truth values of the query's predicates. MSCN is capable of modeling joins and predicates over several tables and hence can cover correlations in the data. A very complex network structure estimates the cardinality of the given query. The large sampling space to cover the whole schema leads to a sparse representation of different queries and their cardinalities. If a query without a representative is passed to the network, the network's interpolation capability fails and the estimate is erroneous. The complex network

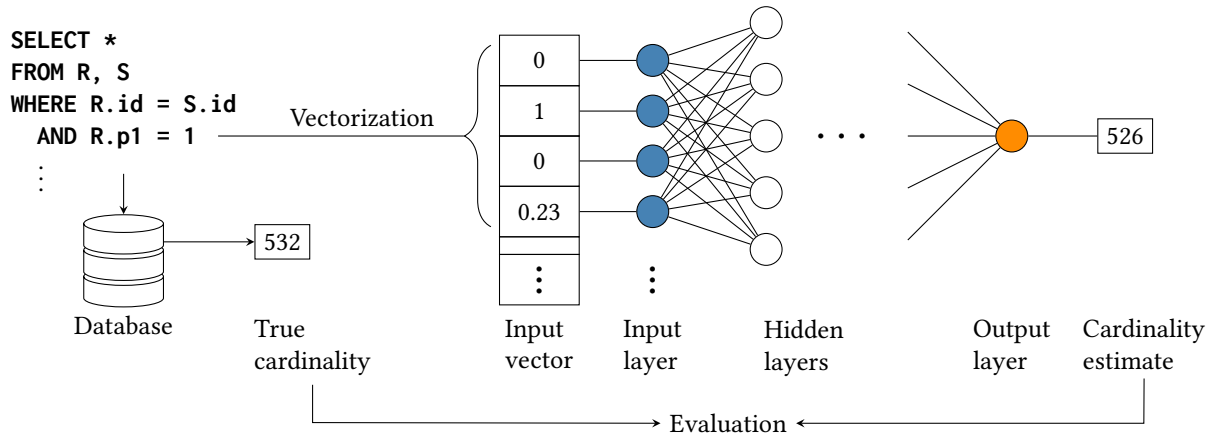

Figure 2: Process for cardinality estimation with learned models.

structure also results in an increased learning time of the model and slower cardinality estimation itself.

Besides cardinality estimation, machine learning techniques have been also used to optimize other RDMS internals. For example, reinforcement learning has been applied for query optimization as well as to solve the join ordering challenge [17, 23]. Moreover, Kraska et al. proposed a learned index structure as novel indexing approach [12]. Here, a model learns the sort order or the structure of lookup keys and uses these signals to effectively predict the position or existence of tuples.

3 LOCAL CARDINALITY ESTIMATION

As already demonstrated by [11, 14, 16], the application of deep learning techniques to the cardinality estimation task enhances the accuracy compared to traditional approaches. Nevertheless, all recently introduced approaches have shortcomings in terms of accuracy, network size or effort for training as mentioned in the previous section. To overcome these issues, we propose a novel *local model approach* in this paper. Our local model approach is characterized by the fact that we build different neural networks (models) for various subparts of the database schema instead of having one global neural network for the whole schema at once. Specifically, we build our models at the granularity of n-ary joins and their corresponding filter predicates that occur in a given workload. Without loss of generality, we restrict ourselves to equi-joins at the moment with different filter predicates opportunities.

The exact structure of the network, i.e. number of layers and the number of neurons, is subject to the adaption of our local model to the problem, also known as hyperparameter tuning. We explain these structural properties of the neural network and their influences in Section 4.3 as part of the evaluation. In machine learning, cardinality estimation can be seen as a regression problem using SQL queries with relation and attribute constraints as input and the cardinality as the objective. A machine learning based estimator takes any vectorized query as input and returns a cardinality for said query. This assumption holds for both global and local models. In Section 3.1, we detail how a neural network can be used for regression. The featurization of queries is described in Section 3.2.

3.1 Regression with Neural Networks

A standard way for modeling regression with neural networks are multi-layer perceptrons (MLP). This kind of neural network is defined by three types of layers: an input layer, hidden layers, and an output layer. Figure 2 shows an example of such a neural network. Each layer contains a configurable but fixed number of neurons. Each neuron is connected to all neurons in both the previous and following layer. Therefore, these layers are called fully-connected. When applied to regression, MLPs use an n-dimensional vector as input and produce a floating point or an integer number as output. The input vector dictates the numbers of neurons in the input layer of the neural net. The output of a regression neural network is a scalar. While training, example queries with known cardinalities will be passed through the network. Such example queries are called the *ground truth*. The neural network self-optimizes its prediction of the output (i.e. cardinalities) based on the ground truth. Once trained, a neural network can be used to estimate cardinalities for both known queries which were already in the workload or unknown queries which are newly introduced to the workload. Our approach uses such a regression neural network. The input vector is the vectorized query (see Section 3.2) and the output is the estimated cardinality of the query.

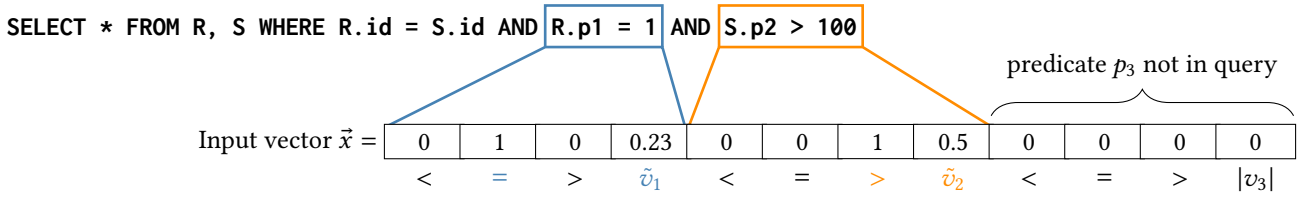

Figure 3: Vectorization of a query

3.2 Vectorization

In order to model a join and the resulting correlations in the data, we need to instantiate the given join in the database. This generates a *join table* of width n where $n = \#cols(R) + \#cols(S)$. These assumptions also hold for more complex joins over three or more tables.

A neural network can only take numerical values as its input vector. An SQL query has to be transformed from its string representation to a numerical vector. This process is called *vectorization*. We use the predicates on the two tables on which the join is based as a foundation for our vectorization. Every possible predicate $p_i \in \{p_1, \ldots, p_n\}$ on those two tables generates 4 entries in the input vector $\vec{x} \in \mathbb{R}^{4n}$. We are only allowing selections on non-key predicates because we argue that there are no useful selections on key predicates in our scenario.

We differentiate between two encodings in our vectorization: the *operator encoding* and the *value encoding*. The operator encoding vectorizes the choice of operator for the predicate p_i. We use *one-hot encoding* to transform a number of choices to a vector. With three singular operators <, =, and >, we need a vector of length three to model all possible operators. The presence of an operator dictates a 1 at the corresponding position in the vector. We have chosen to use the order (<, =, >) for our purposes. For example, < generates the vector (1, 0, 0) and <= generates the vector (1, 1, 0). The operator encoding takes up the first three entries for each predicate. Next, we need to encode the chosen value for the predicate p_i. Usually, this is the numeric or character value v_i on the right hand side of the operator. The value encoding is a single floating point number representing this value. If this value is non-numeric, we use dictionary encoding to transform it to an integer. Neural networks are usually used with min-max-normalized input vectors ranging from 0 to 1 to enhance their accuracy [8]. We normalize our predicate value v_i to this range as shown in Equation (1).

$$\tilde{v}_i = \frac{v_i - \min(p_i)}{\max(p_i) - \min(p_i)} \tag{1}$$

The minimum and maximum values are the boundaries of the range of the predicate p_i where v_i is the value of p_i in the query. These can be obtained directly from the database. For the purpose of our model, we only take those predicates into account which select directly on the join table.

From Figure 2, we assume that the join of **R** and **S** has three columns $\{p_1, p_2, p_3\}$ excluding the join predicate. The vectorization generates an input vector $\vec{x}$ of length 12, four entries for each predicate. Note that the predicates are chosen from different relations, i.e. p_1 comes from **R** and p_2 and p_3 come from **S**. Figure 3 shows the encoding for the example query. We mark the direct translation of query predicates to parts of the input vector.

4 EVALUATION

We conduct an experimental study to evaluate the performance of our approach. We begin with the experimental setup (Section 4.1), including the evaluation data and the comparison techniques. This is followed by a detailed description of the experiments and the discussion of their results.

The process of generating data according to the experimental setup of Kipf et al. is described in Section 4.1. We compare our results to two other estimators, one traditional estimator and one learned estimator, in Section 4.2. In Section 4.3, we evaluate different configurations of our network topology. We perform experiments on how many queries are necessary for a stable model in Section 4.4. Last, we show the performance of all tested models and compare their training time and test time in Section 4.5.

4.1 Experimental Setup

The base for our evaluation is the IMDB data set[1]. We focused on two joins: (1) **title⋈movie_info** and (2) **movie_companies⋈movie_info⋈title**. The first join results in approximately 29 million tuples and the second one in 134 million tuples. Further information about the data can be found in Table 1. All properties are chosen in order to ensure comparability to other approaches. We selected all columns from each join which can be represented as integers. All experiments are executed five times and their results are averaged. For each of the five complete training runs in every experiment, we randomly split our data set of 105,000 queries in 90,000 queries for training, 10,000 queries for validation,

[1] ftp://ftp.fu-berlin.de/pub/misc/movies/database/frozendata/

table	column	number of distinct values
title	kind_id	8
title	production_year	145
movie_info	info_type_id	78
movie_companies	company_id	362,131
movie_companies	company_type_id	2

Table 1: Properties of IMDB tables and columns.

and 5,000 queries for testing. All sets are disjunctive from each other. This gives a good assessment of the model's generalization and prevents overfitting. Note that in real-world scenarios, there can be an overlap between training and test sets but this would only improve the performance of our neural network. The neural network sees different queries and cardinalities each training run. Thus, we prevent it from just remembering specific queries through repetition.

As the evaluation metric, we choose the q-error or multiplicative error [20]. This measurement is the factor of misjudgment between the ground truth value y_{true} and the estimated value y_{est}. It ranges from 1 for $y_{est} = y_{true}$ to ∞ for misestimates because it has no upper limit.

$$q(y_{true}, y_{est}) = \frac{\max(y_{true}, y_{est})}{\min(y_{true}, y_{est})} \tag{2}$$

We use the sampling technique for the generation of queries as proposed in [11]. We only need to sample the predicates because the number of joins is fixed for each of our neural network. For this, we sample the number of predicates l uniformly from $[1, p_n]$ with $l \in \mathbb{N}$. This is done coherently to Kipf et al. We use the same operator set (<, =, >) as well. Every generated query is executed through a database to obtain the correct cardinality. We generate a total of 105,000 queries for each join table. This is a strict subset of the sampling space proposed by Kipf et al. This set of queries only assesses the example joins but they are still covered in the original sample space. So, the global MSCN should be able to generalize from its training data to estimate the cardinalities of these queries.

model	mean q-error	median q-error	max q-error	95th mean q-error
2-way join				
Posgres	212,311.8	62.6	23,913,682.0	638,227.4
MSCN	1,353.8	3.4	194,170.7	1,973.6
Local NN	4.9	1.4	1,569.7	11.7
3-way join				
MSCN	4748.5	5.0	1315521.7	2041.1
Local NN	26.4	1.9	24304.8	32.4

Table 2: Resulting q-errors of different approaches.

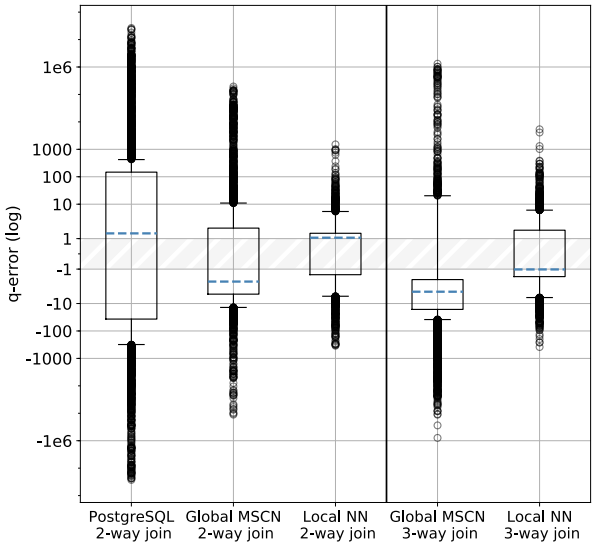

Figure 4: Evaluation results of different cardinality estimators.

All experiments are executed on an AMD A10-7870K Radeon R7 with 32GB RAM. All neural networks, including networks from other publications, are trained with CUDA capability on an NVIDIA Tesla K20c with keras[2].

4.2 Accuracy

To evaluate the estimation accuracy of our local model, we compare our approach with two other techniques. First, we select a baseline estimator which does not use machine learning. Postgres uses traditional estimations based on histograms or most frequent values[3]. This approach is not capable to model correlations between predicates because it assumes data independence. Next, we choose multi-set convolutional neural network (MSCN) as a state-of-the-art approach using deep learning for cardinality estimation. The MSCN model is trained on the data set from the original publication. These queries are sampled from the same IMDB tables, we use throughout our evaluation. Last, we train our local estimator on the sampled queries mentioned in Section 4.1. For this, we choose the network structure which performs best on our data sets. An evaluation of different tested structures can be found in Section 4.3.

Figure 4 and Table 2 detail the accuracy results for all estimators on join (1) and (2). The q-error of each estimate in our test set is scaled to a symmetric log-space on the y-axis. Whereas underestimates get a negative value and

[2]https://keras.io

[3]https://www.postgresql.org/docs/10/row-estimation-examples.html

overestimates a positive value, both between 1 and ∞. The blue line is the tendency of an estimator to overestimate or underestimate. The box for each model details the confidence interval (CI) around the median q-error. The number of q-errors which are outside the CI but inside 1.5 times the CI are represented with whiskers. Outliers are plotted as points outside the whiskers. The closer all elements are around the q-error range between −1 and 1 the better the accuracy of a model. Table 2 shows the mean q-error, the median q-error, the maximum q-error, and the mean q-error of the 95th percentile. All q-errors refer to the accuracy of the models on 5,000 test queries.

It can be observed that our model can estimate data correlation much more precisely. We gain an improvement in accuracy of four orders of magnitude to traditional estimators and a factor of 275 to the global model regarding the mean q-errors from Table 2. With a smaller focus, our neural network has the advantage of using more of its estimation quality on queries which access similar data. The neural network does not need to generalize over a large data context like a complete schema but learns local data correlations which are easier to model in general. This leads to two improvements as shown by the fliers and whiskers in Figure 4: 1) Our model is not as susceptible to misestimates. 2) The variance in our estimates is much smaller.

The assumption that MSCNs can generalize from a general workload of sample queries to a specific one does not hold. By using our local approach different joins can be evaluated with different specialized local models (see Section 5). This also allows for using other approaches such as histograms or sketches where a neural network would introduce too much overhead.

Our model can not only be used for joins over two tables but also for an arbitrary number of tables. To show the accuracy of a larger join, we use join (2) mentioned in Section 4.1. The last two box plots in Figure 4 and the last lines in Table 3 show the performance of MSCN and our model on join (2). The accuracy improves by a factor of 180 compared to the global approach when comparing the mean q-errors. Note that this model is trained on only 50,000 sample queries as motivated in Section 4.4. This shows, that our approach can be generalized to n-way joins be keeping higher accuracy compared to global models.

4.3 Network Structure

There are two main components which have the most significant influence on the accuracy of a neural network: width and depth. The width of a fully-connected neural network is the number of neurons in the first hidden layer[4]. The depth describes the number of consecutive hidden layers

[4]All following layers have half the number of neurons as their predecessor.

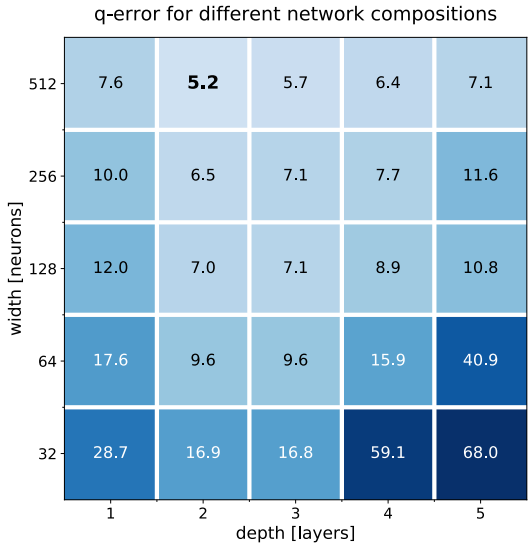

Figure 5: Neural network structure evaluation.

in the network. To tune the network for our purposes, we use an exhaustive grid search with depth and width as the search space axes. The width varies between the values (32, 64, 128, 256, 512) and the depth varies between the values (1, 2, 3, 4, 5). For each combination of a given depth and width, a network is instantiated with these parameters, trained on the train data with 100 epochs and tested on the test data. We have chosen the 2-way join (1) for this experiment with predicates on three attributes depicted in Table 3. Figure 5 shows the q-errors for all network combinations. The x-axis describes the depth parameter, whereas the y-axis details the width parameter. It is clearly visible that the best combination is a network with two hidden layers and 512 neurons in the first layer. Another result is that width has more impact on the q-error than the depth. Shallow models with broad layers perform better than deep models with narrow layers up to a factor of ten. For all other experiments, we therefore decide to use a network with the following configuration: input layer ($4n$ neurons), hidden layer 1 (512 neurons), hidden layer 2 (256 neurons), output layer (1 neuron).

4.4 Number of Sampled Queries

The *cold start problem* in machine learning means that if a system is freshly installed, a lot of training data is required for a first model. Most of the time, one does not have a sufficient number of samples for training in this scenario and compute expensive data generation (see Section 4.1) is required. In data management and especially in cardinality estimation this usually leads to random sampling of example queries representing an artificial workload. To mitigate the cold start problem, we assess how fast our network converges given a

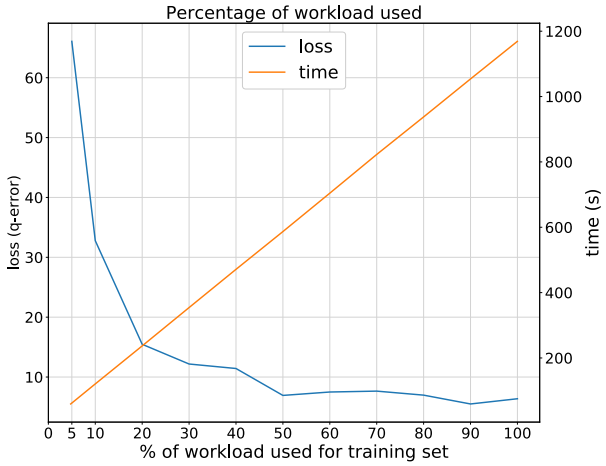

Figure 6: Number of samples needed.

specific number of input queries from the workload as training data. We look at the number of required queries for an artificial start-up workload to result in a stable estimation for join (1). The number of queries used for training is increased by 10,000 in each step and a sample of 5,000 queries is chosen from the remaining queries for testing. The neural network is trained with a batch size of 32. The results are shown in Figure 6.

First of all, the runtime of a neural network in training scales linearly with the number of training queries if parameters like width, depth and batch size are kept constant on a single machine. It shows that we reach a mean q-error of 6.9 when using 50% of queries. We argue that therefore 50,000 queries are enough to build a first robust model for this join. This leaves us with 50% of runtime for model training (i.e. 598s) compared to the full sample set.

4.5 Runtime Performance

Since performance is a key feature for using learned cardinality estimators in database management systems, we detail the runtime evaluation of all models. For evaluating the runtimes of different models, we use join (1) and 105,000 sampled queries accessing this join. The experiment is carried out as described in Section 4.1. Additionally, our model is also trained with the number of samples necessary for a stable model (see Section 4.4). The MSCN and our model are trained with GPU capability and a batch size of 32.

Table 3 details the training and test time for all model-based approach and the runtime for the Postgres query analyzer. The training time consists of the 100 epochs needed for each model to be fitted to the 100,000 train queries. The test time is the evaluation time for a single query, i.e. the time for the **`EXPLAIN ANALYZE`** statement for Postgres or the time for a forward pass through the neural network. One could

Model	Training time	Testing time (per sample)
Postgres	–	1.8s
MSCN	4945s	33ms
Local NN (full sample set)	1159s	29µs
Local NN (sufficient samples)	598s	29µs

Table 3: Runtime performance of models.

argue that histogram construction in Postgres is equivalent to a training process but it uses a different type of data, the data distribution of each attribute. Therefore, a comparison would be insufficient.

For model training, we can achieve a speed-up of factor four and a factor of eight if only a reduced number of queries is used. Our neural network estimator is faster than Postgres by a factor of 62,000. Compared to MSCN, the forward pass of the local model is three orders of magnitude faster since the structure is much smaller. When we apply our model is fast enough to have little to zero impact on the query's execution time.

5 FUTURE WORK

The approach of using neural network for cardinality estimation is still in its infancy and there is a multitude of challenges ahead that needs to be solved. The most important issues that have to be tackled are the *cold start problem*, the number of samples needed to train the network, the maintenance of the models, and their interplay and integration with statistical approaches based on histograms.

Curriculum Learning. We want to mitigate the cold start problem for cardinality estimation by applying curriculum learning. The basis notion is that in human learning the supervision often follows a curriculum where the teacher presents the examples not randomly but in a specific order. The approach of curriculum learning [1] transfers this principle to supervised learning. It is an extension of stochastic gradient descent where easy examples are over-sampled at the beginning of training. This gives a higher probability to escape a low quality local minimum, since the variance of the gradient direction increases with the difficulty of samples [28]. In [13, 28] it was shown, that by applying curriculum learning the training converges much faster but also improves the generalization performance of the learned models. For learning cardinality estimations, the key question is how to rank samples according to their difficulty. This can be done either by using the estimation error of the underlying database, by investigating existing models or by using operator embeddings [18]. Operator embeddings [18] map a query operator to a low-dimensional vector space that captures much information about the operator. The authors of [18] showed that operator embeddings can be used to classify

operators for which the query optimizer's cardinality estimate is correct, too high, or too low. This could be used as an approximation of the difficulty.

Model Maintenance. Currently, we are assuming that our database schema as well as the underlying data is static. While the learned cardinality estimation models should be general enough to tolerate minor shifts in data distribution and correlation, there will also be cases where these models need to be adapted or learned from scratch. Therefore, we need to monitor the cardinality estimation error and trigger model maintenance if needed. The notion of model maintenance shares some aspects with the approach of transfer learning. Transfer learning is the improvement of learning in a new task through the transfer of knowledge from a related task that has already been learned [22]. The core question that has to be investigated is whether it is possible to transfer the knowledge from a given model to another setup with different data characteristics.

Integration with Statistical Approaches. Moreover, we want to combine existing statistical approaches with learned models. Since the computation of histograms is pretty cheap compared to learning neural networks, they provide a plain-vanilla approach that could be used to mitigate the cold start problem. Additionally, they remain applicable for all cases where the independence assumption holds. To support a decision-making process in choosing either statistical or neural-network approaches, we plan to discover functional dependencies and correlations between pairs of column [4]

6 CONCLUSION

In this paper, we present a neural network approach for cardinality estimation that focuses on local models instead of global models. The evaluation shows that our local models outperform global models in terms of accuracy as well as training and testing time. Beside this significant improvement, we believe that local models have the advantage to be much easier to maintain when it comes to drifts in data or schema and workload changes compared to global models

REFERENCES

[1] Yoshua Bengio, Jerome Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum Learning. In *ICML*. 41–48.

[2] Nicolas Bruno and Surajit Chaudhuri. 2002. Exploiting statistics on query expressions for optimization. In *SIGMOD*. 263–274.

[3] Pit Fender and Guido Moerkotte. 2011. A new, highly efficient, and easy to implement top-down join enumeration algorithm. In *ICDE*. 864–875.

[4] Ihab F Ilyas, Volker Markl, Peter Haas, Paul Brown, and Ashraf Aboulnaga. 2004. CORDS: Automatic Discovery of Correlations and Soft Functional Dependencies. In *SIGMOD*. 647–658.

[5] Yannis E Ioannidis and Stavros Christodoulakis. 1993. Optimal Histograms for Limiting Worst-case Error Propagation in the Size of Join Results. *ACM Trans. Database Syst.* 18, 4 (Dec. 1993), 709–748.

[6] Yannis E Ioannidis and Viswanath Poosala. 1995. Balancing Histogram Optimality and Practicality for Query Result Size Estimation. In *SIGMOD*. 233–244.

[7] H V Jagadish, Nick Koudas, S Muthukrishnan, Viswanath Poosala, Ken Sevcik, and Torsten Suel. 1998. Optimal Histograms with Quality Guarantees. In *VLDB*. 275–286.

[8] T Jayalakshmi and A Santhakumaran. 2011. Statistical normalization and back propagation for classification. *IJCTE* 3, 1 (2011), 1793–8201.

[9] Tomas Karnagel, Dirk Habich, and Wolfgang Lehner. 2015. Local vs. Global Optimization: Operator Placement Strategies in Heterogeneous Environments. In *EDBT/ICDT Workshops*. 48–55.

[10] Tomas Karnagel, Dirk Habich, and Wolfgang Lehner. 2017. Adaptive Work Placement for Query Processing on Heterogeneous Computing Resources. *PVLDB* 10, 7 (2017), 733–744.

[11] Andreas Kipf, Thomas Kipf, Bernhard Radke, Viktor Leis, Peter A. Boncz, and Alfons Kemper. 2019. Learned Cardinalities: Estimating Correlated Joins with Deep Learning. In *CIDR*.

[12] Tim Kraska, Alex Beutel, Ed H Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The case for learned index structures. In *SIGMOD*. 489–504.

[13] Kai A Krueger and Peter Dayan. 2009. Flexible shaping: How learning in small steps helps. *Cognition* 110, 3 (2009), 380 – 394.

[14] Seetha Lakshmi and Shaoyu Zhou. 1998. Selectivity estimation in extensible databases-A neural network approach. In *VLDB*. 623–627.

[15] Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, and Thomas Neumann. 2015. How good are query optimizers, really? *PVLDB* 9, 3 (2015), 204–215.

[16] Henry Liu, Mingbin Xu, Ziting Yu, Vincent Corvinelli, and Calisto Zuzarte. 2015. Cardinality Estimation Using Neural Networks. In *CASCON*. 53–59.

[17] Ryan Marcus and Olga Papaemmanouil. 2018. Deep reinforcement learning for join order enumeration. In *aiDM*. ACM, 3.

[18] Ryan Marcus and Olga Papaemmanouil. 2019. Flexible Operator Embeddings via Deep Learning. *arXiv:1901.09090* (2019).

[19] Volker Markl, Vijayshankar Raman, David E. Simmen, Guy M. Lohman, and Hamid Pirahesh. 2004. Robust Query Processing through Progressive Optimization. In *SIGMOD*. 659–670.

[20] Guido Moerkotte, Thomas Neumann, and Gabriele Steidl. 2009. Preventing bad plans by bounding the impact of cardinality estimation errors. *VLDB* 2, 1 (2009), 982–993.

[21] Ravi Mukkamala and Sushil Jajodia. 1991. A Note on Estimating the Cardinality of the Projection of a Database Relation. *ACM Trans. Database Syst.* 16, 3 (1991), 564–566.

[22] E S Olivas, J D M Guerrero, M M Sober, J R M Benedito, and A J S Lopez. 2009. *Handbook Of Research On Machine Learning Applications and Trends: Algorithms, Methods and Techniques*.

[23] Jennifer Ortiz, Magdalena Balazinska, Johannes Gehrke, and S Sathiya Keerthi. 2018. Learning state representations for query optimization with deep reinforcement learning. *arXiv:1803.08604* (2018).

[24] G Piatetsky-Shapiro and C Connell. 1984. Accurate Estimation of the Number of Tuples Satisfying a Condition. In *SIGMOD*. 256–276.

[25] Viswanath Poosala, Peter J Haas, Yannis E Ioannidis, and Eugene J Shekita. 1996. Improved Histograms for Selectivity Estimation of Range Predicates. In *SIGMOD*. 294–305.

[26] Viswanath Poosala and Yannis E Ioannidis. 1997. Selectivity estimation without the attribute value independence assumption. In *VLDB*, Vol. 97. 486–495.

[27] Viktor Rosenfeld, Max Heimel, Christoph Viebig, and Volker Markl. 2015. The Operator Variant Selection Problem on Heterogeneous Hardware. In *ADMS@VLDB*. 1–12.

[28] Daphna Weinshall, Gad Cohen, and Dan Amir. 2018. Curriculum Learning by Transfer Learning: Theory and Experiments with Deep Networks. *arXiv:1802.03796* (2018).