CELA: An Accurate Learned Cardinality Estimator with Strong Generalization Ability and Dimensional Adaptability

Weiqing Zhou, Siyu Zhan(✉), Lei Guo, and Bo Dai

University of Electronic Science and Technology of China, No. 2006, Xiyuan Ave, West Hi-Tech Zone, Chengdu 611731, Sichuan, China
wqzhou@std.uestc.edu.cn, {zhansy,leiguo,daibo}@uestc.edu.cn

Abstract. Accurate cardinality estimation contributes significantly to query optimization whereas traditional approaches such as histogram-based or sketching-based approaches relying on assumption of uniform distribution of data and appropriate pre-set parameters, often leading to dilemma in practical applications. In this paper, an accurate lightweight, dimensionally adaptive, strongly generalizable learned cardinality estimator for multi-dimensional range queries, CELA is proposed reflecting on the characteristics of desirable cardinality estimators. For the purpose of capturing relationship between dimensions, CELA raises a query-oriented approach of constructing constraint matrices to apply convolution. Experiments illustrates that CELA performs superbly on each defined indicators far superior to PostgreSQL. Furthermore, the strong generalization ability of CELA is demonstrated by the excellent performance trained with continuously scaled-down training set.

Keywords: Cardinality estimation · Dynamic architecture · Deep learning · Generalization ability · Dimensional adaptability

1 Introduction

There are many methods for database query optimization [3,9,10], and cost estimation is an inextricable part of them all [7], in which cardinality estimation is an important component. Cardinality estimation specifically refers to predicting the number of records in the query results. The cardinality estimation is so critical that it has been studied by researchers in the field of databases for decades, but there is still much room for improvement. There are already many widely used traditional methods. The performance of histogram-based methods [13] depends on whether the data conform to the assumption of a uniform distribution. What's more, advance settings for width or depth are also required. Sketching-based methods are often applied to estimate the number of different data, but requires extra space to store the bitmaps and carefully designed hash

Supported by the National Key R&D Program of China (grant No. 2019YFB1705601) and the Natural Science Foundation of China (grant No. 62072075).

W. Zhang et al. (Eds.): WISE 2021, LNCS 13080, pp. 111–118, 2021.
https://doi.org/10.1007/978-3-030-90888-1_9

functions for different workloads. Machine learning and deep learning are rapidly evolving in research and industry [1,5,6,14]. With the explosion of this technology, some learned cardinality estimators emerges which have shown promising results.

An exquisite, lightweight, and easy-to-train cardinality estimation model with high generalization ability and dimensional adaptability, CELA is proposed in this paper. Specifically, the approach for dynamic vectorization and dynamic model architecture is proposed for adaption to queries with different dimensions. In addition, an efficient algorithm for data generation is explored. What's more, three indicators for measuring the model are presented, and the performance of CELA is tested and compared with other estimators.

2 Related Work

Cardinality estimation is a crucial part of query optimization. However, the traditional methods are helpless to cope with multi-dimensional constraints. With the excellence of machine learning and deep learning, learned cardinality estimators emerge continuously. Andreas Kipf [4] extracts features of tables, joins, and predicates separately with fully connected networks and learn the cardinality with MLPs after aggregation. Furthermore Lucas Woltmann [12] transforms the query into a one-dimensional vector as input to the neural networks. Hasan [2] seeks to estimate the joint probability distribution from samples. Nevertheless, the researches have not yet explored in-depth specifically on the generalization ability of models and adaptability to queries with various dimensions.

3 Vectorization

For instance, a query q, "select a_1 from A where $a_1 > b_1$ and $a_2 = b_2$ and ... and $a_k < b_k$;", "$a_i > b_i$" is defined as the constraint on the data in dimension w_i of the query, in which the predicate p_i and the qualifier b_i need to be parsed. For a query with k dimensions, k constraint triplets like $<a_i, p_i, b_i>$ are obtained.

Each constraint triplet is refined from the query q. Firstly, a_i is processed and a sequence is maintained to specify their order. In the constraint vector, p_i is encoded with 3 bits using one-hot and normalized b_i occupies 1 bit. The 4-bit vector is then passed through a fully connected layer containing n neurons and a ReLU layer resulting in a vectorized representation of n bits on w_i. The constraint matrix of size $[k, n]$ is obtained by splicing constraint vectors in order and adjusting the size of the matrix. Such a multi-dimensional representation, instead of a $[1, k * n]$ single-dimensional vector, allows better utilization of convolution to capture the correlation between different attributes.

4 Model

4.1 Considerations

Cardinality estimation is a part of physical execution plan generation. The time taken for a query from reaching the storage engine to returning the final result,

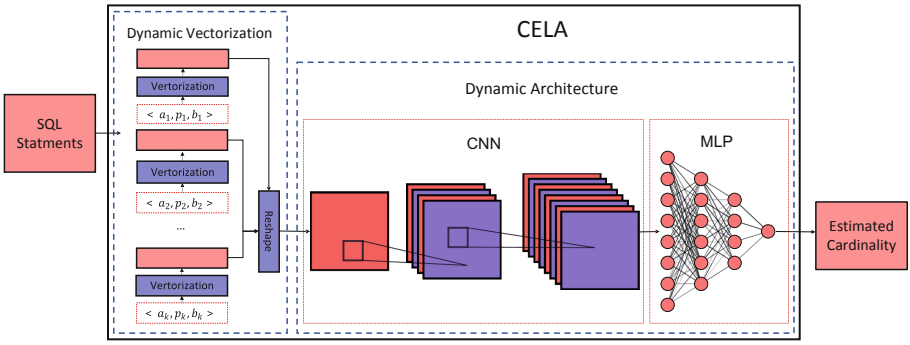


Fig. 1. The architecture of CELA.

includes the time for generating the physical plan and the time for executing the plan in the storage engine. Therefore, the key for learned estimators is to produce the accurate estimated results in a short time. From the above issues, a dynamic model architecture is designed, which is lightweight and can capture inter-dimensional relationship and have strong generalization ability.

4.2 Dynamic Vectorization

The number of bits encoded by one-hot for each constraint triplet in the query q is denoted as EB. The number of neurons in the single-layer MLP, n is related to the dimension k and the number of encoding bits EB. In terms of popularity, the more dimensions, the more neurons. When the encoding is determined, n is expected to be monotone polynomial with only one variable k. In this work, n is specifically equal to $EB * K$ and EB is 4. For high computing performance, the size of the constraint matrix is adjusted to match data with different dimensions. The number of elements in the constraint matrix can be calculated as $EB * K^2$. Provided that EB is a perfect square number, constraint matrix can be reshaped to $[k\sqrt{EB}, k\sqrt{EB}]$ as further input.

4.3 Dynamic Architecture

For the dynamic input scale brought by dynamic vectorization, a dynamic network architecture is designed to accommodate varying dimensions, as well as to avoid excessive overhead. The architecture is shown in Fig. 1, which contains two parts, CNNs with two convolutional layers and MLPs with four layers. The output channels of the two convolutional layers, $Channel_{conv1}$ and $Channel_{conv2}$ are set adaptively and separately shown as Formula 1 and Formula 2.

$$Channel_{conv1} = EB * k \tag{1}$$

$$Channel_{conv2} = EB^{3/2} * k \tag{2}$$

The output of CNNs is reshaped to a one-dimensional vector of size $[1, EB^{5/2} * k^3]$ for further computation. The MLPs are also with adaptive layers according to k. The number of neurons in each layer of the network is expressed as Formula 3–5, and the output layer contains only one neuron.

$$Numel_{mlp1} = EB * \mathrm{FN}(EB, k) = EB^2 * k^2 \tag{3}$$

$$Numel_{mlp2} = EB^{1/2} * \mathrm{FN}(EB, k) = EB^{1/2} * k^2 \tag{4}$$

$$Numel_{mlp3} = k \tag{5}$$

5 Experiment

5.1 Generating Data

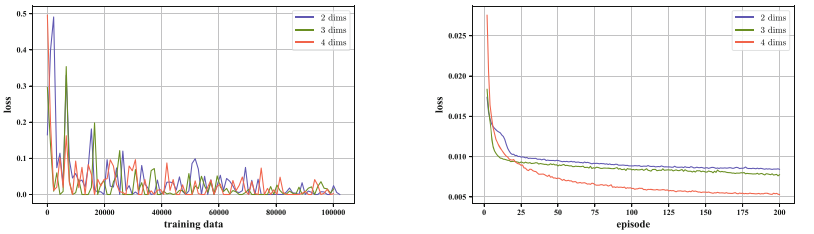
The following way of generating queries is often considered. For the constraints in the queries on each dimension, values are taken evenly within the range of the attributes in the database. This is followed by a random selection of predicates for that dimension, but the values obtained may not exist in the database and skewed distribution may bring troubles. Therefore, the proposed algorithm of generating data is illustrated as Table 1. The experiments are conducted on the 4GB TPC-H workload [11] in PostgreSQL [8] and data is generated separately according to Table 1. 211,721 samples including 102,789 training samples, 215,636 samples including 101,032 training samples and 200,936 samples including 100,869 training samples are separately generated with 2, 3 and 4 dimensions. It is of interest that in this work, both the training set and the test set is close to 50%, revealing the strong generalization ability.

Table 1. Generating data.

The algorithm of generating data	
Step 1	Sample randomly in the database to obtain the real records in the table;
Step 2	Extract the constraint values $b_1, b_2, b_3, \ldots, b_k$ for each of the attributes
Step 3	Select randomly one of the three predicates ,“>”, “<”and “=”, as p_i, which forms a constraint triplet $<a_i, p_i, b_i>$ on dimension w_i for a query;
Step 4	Repeat Step 2, 3 until the constraint triplet on all dimensions is formed.;
Step 5	Repeat Steps 1–4 until a sufficient amount of data is generated

5.2 Model Training

For all three dimensions of data L1Loss is set as the loss function and the loss in one training episode is shown in Fig. 2(a). It illustrates that the model converges fast without seeing much data no matter which dimension it is trained on. Figure 2(b) illustrates the average loss for 200 continuous episodes. Figure 2(b) demonstrates that the loss is reduced to a low level after just one training episode, although there is a steep drop during approximately 25 more training episodes.



(a) Loss of one training episode (b) Average loss of 200 training episodes

Fig. 2. Training loss.

5.3 Evaluation

In this work, three quantitative indicators are proposed as follow to measure the performance of the model. First, The total number of queries for which the estimated cardinality in the test set is exactly equal to the true cardinality expressed as Formula 6. Second, the average error cardinality for all queries in the test set expressed as Formula 7. Third, the average error rate of all queries in the test set expressed as Formula 8.

$$EQ = \sum_{i=1}^{QT} x_i, x_i = \begin{cases} 1, Car_{ie} = Car_{it} \\ 0, Car_{ie} \neq Car_{it} \end{cases} \tag{6}$$

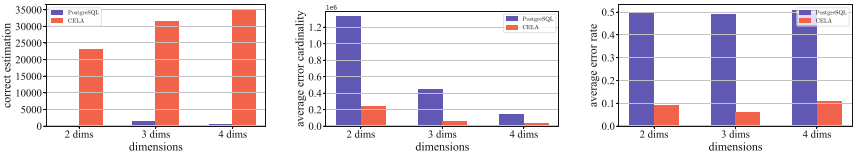
$$Avg_{ec} = \frac{\sum_{i=1}^{QT} |Car_{ie} - Car_{it}|}{QT} \tag{7}$$

$$Avg_{er} = \frac{\sum_{i=1}^{QT} |Car_{ie} - Car_{it}|}{\sum_{i=1}^{QT} Car_{it}} \tag{8}$$

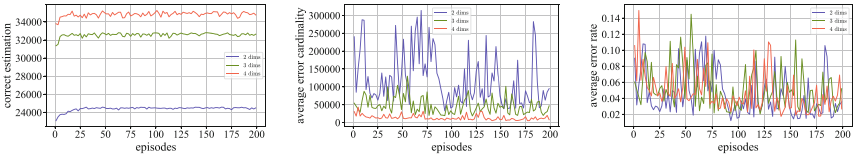
From Fig. 3(a)–(c) it is seen that the proposed model outperforms the cardinality estimator in PostgreSQL for each indicators a lot. The Fig. 3(d)–(f) shows that there is only a small improvement in each dimensions after more episodes of training not matching the overhead of training.

To demonstrate the strong generalization ability of CELA, we keep the test set constant, continuously add 200 queries into training set, and then test the models. From Fig. 3(g)–(i), the performance of the first few dozen models on EQ can even approach the best performance. Shown in Fig. 3(j)–(o), about the $300th$ model outperforms PostgreSQL in all three dimensions, and the training set is about 600,000 queries, **37.5**% of the total data set. The fantastic results are full proof the strong generalization ability of CELA.

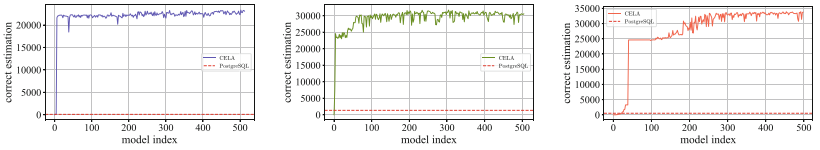
Overall, the performance of the models on varying dimensions is shown in Table 2 retaining three decimals. MSCN [4] only take predicate sets as input to adapt to our workload, so there is no need for concatenation after average pooling. Only the attributes need to be encoded in vectorization of LocalNN [12], so it can be applied directly. The two recent learned estimators are selected to



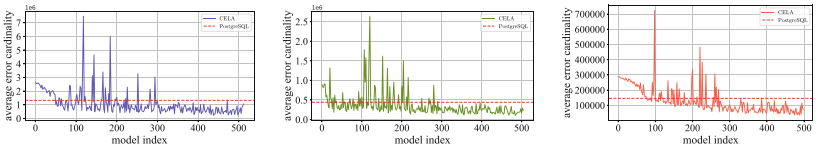
(a) EQ for CELA in one training episode. (b) Avg_{ec} for CELA in one training episode. (c) Avg_{er} for CELA in one training episode.



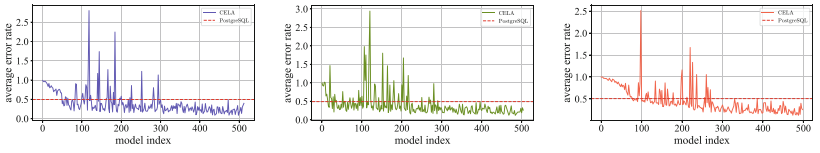
(d) EQ for CELA in 200 training episodes. (e) Avg_{ec} for CELA in 200 training episodes. (f) Avg_{er} for CELA in 200 training episodes.



(g) EQ for CELA with 2 dimensions on different training set. (h) EQ for CELA with 3 dimensions on different training set. (i) EQ for CELA with 4 dimensions on different training set.



(j) Avg_{ec} for CELA with 2 dimensions on different training set. (k) Avg_{ec} for CELA with 3 dimensions on different training set. (l) Avg_{ec} for CELA with 4 dimensions on different training set.



(m) Avg_{er} for CELA with 2 dimensions on different training set. (n) Avg_{er} for CELA with 3 dimensions on different training set. (o) Avg_{er} for CELA with 4 dimensions on different training set.

Fig. 3. Performance of CELA.

Table 2. Performance on different dimensions.

	Cardinality estimator	EQ	Avg_{ec}	Avg_{er}
Performance on 2 dims	PostgreSQL	91	1,326,542.005	0.497
	LocalNN	22,433	221,571.788	0.083
	MSCN	22,358	184,191.273	0.069
	CELA	**24,585**	**32,362.293**	**0.012**
Performance on 3 dims	PostgreSQL	1,341	441,115.440	0.491
	LocalNN	30,768	243,296.303	0.271
	MSCN	28,614	158,685.886	0.177
	CELA	**32,810**	**18,378.845**	**0.020**
Performance on 4 dims	PostgreSQL	519	145,902.568	0.506
	LocalNN	32,022	165,388.694	0.574
	MSCN	30,897	79,541.116	0.276
	CELA	**35,423**	**4,998.175**	**0.017**

compare with CELA. The performance of PostgreSQL on EQ is unstable, while CELA's performance continuously improves with dimensions and exceeds other three estimators a lot. On Avg_{ec} and Avg_{er}, the performance of PostgreSQL is relatively stable but poor. As the dimensions increases, the performance of both MSCN and LocalNN decreases rapidly, and LocalNN even performs worse than PostgreSQL, while CELA is quite stable as the dimensions increase. In summary, there is tremendous progress brought by CELA with approximately $\mathbf{2.4}\% - -\mathbf{4}\%$ error of PostgreSQL, $\mathbf{6.2}\% - -\mathbf{17.4}\%$ error of MSCN, $\mathbf{3}\% - -\mathbf{14.5}\%$ error of LocalNN.

6 Conclusions

In this work, an accurate lightweight learned model for multi-dimensional range queries, CELA is proposed with dynamic vectorization and model architecture adapted to the various dimensions allowing the complexity of CELA to match the input size. TPC-H workloads are implemented on PostgreSQL to generate abundant data based on real records. After only one training episode, CELA exceeds PostgreSQL, MSCN and LocalNN by far on all defined indicators and the performance is even improved after more training episodes. For further research, the strong generalization ability of the model is demonstrated by holding the test set constant and reducing the training set. In conclusion, our work is valuable and provides constructive ideas for other researches.

References

1. Ding, J., et al.: Alex: an updatable adaptive learned index. In: Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, pp. 969–984 (2020)

2. Hasan, S., Thirumuruganathan, S., Augustine, J., Koudas, N., Das, G.: Deep learning models for selectivity estimation of multi-attribute queries. In: Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, pp. 1035–1050 (2020)
3. Kamatkar, S.J., Kamble, A., Viloria, A., Hernández-Fernandez, L., Cali, E.G.: Database performance tuning and query optimization. In: Tan, Y., Shi, Y., Tang, Q. (eds.) DMBD 2018. LNCS, vol. 10943, pp. 3–11. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-93803-5_1
4. Kipf, A., Kipf, T., Radke, B., Leis, V., Boncz, P., Kemper, A.: Learned cardinalities: estimating correlated joins with deep learning. arXiv preprint arXiv:1809.00677 (2018)
5. Marcus, R., et al.: Neo: a learned query optimizer. arXiv preprint arXiv:1904.03711 (2019)
6. Marcus, R., Papaemmanouil, O.: Towards a hands-free query optimizer through deep learning. arXiv preprint arXiv:1809.10212 (2018)
7. Markl, V., Raman, V., Simmen, D., Lohman, G., Pirahesh, H., Cilimdzic, M.: Robust query processing through progressive optimization. In: Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data, pp. 659–670 (2004)
8. Momjian, B.: PostgreSQL: Introduction and Concepts, vol. 192. Addison-Wesley, New York (2001)
9. Ortiz, J., Balazinska, M., Gehrke, J., Keerthi, S.S.: Learning state representations for query optimization with deep reinforcement learning. In: Proceedings of the Second Workshop on Data Management for End-To-End Machine Learning, pp. 1–4 (2018)
10. Panahi, V., Navimipour, N.J.: Join query optimization in the distributed database system using an artificial bee colony algorithm and genetic operators. Concurr. Comput. Pract. Exp. **31**(17), e5218 (2019)
11. Poess, M., Floyd, C.: New TPC benchmarks for decision support and web commerce. ACM Sigmod Rec. **29**(4), 64–71 (2000)
12. Woltmann, L., Hartmann, C., Thiele, M., Habich, D., Lehner, W.: Cardinality estimation with local deep learning models. In: Proceedings of the Second International Workshop on Exploiting Artificial Intelligence Techniques for Data Management, pp. 1–8 (2019)
13. Xu, J., Zhang, Z., Xiao, X., Yang, Y., Yu, G., Winslett, M.: Differentially private histogram publication. VLDB J. **22**(6), 797–822 (2013). https://doi.org/10.1007/s00778-013-0309-y
14. Yu, X., Li, G., Chai, C., Tang, N.: Reinforcement learning with tree-LSTM for join order selection. In: 2020 IEEE 36th International Conference on Data Engineering (ICDE), pp. 1297–1308. IEEE (2020)