Mathematical Problems of Computer Science 54, 41–52, 2020.

UDC 004.65, 004.8

Cardinality Estimation of an SQL Query Using Recursive Neural Networks

Davit S. Karamyan

Information Systems
Yerevan State University, Armenia
e-mail: dkaramyan@krisp.ai

Abstract

To learn complex interactions between predicates and accurately estimate the cardinality of an SQL query, we develop a novel framework based on recursive tree-structured neural networks, which take into account the natural properties of logical expressions: compositionality and n-ary associativity. The proposed architecture is an extension of MSCN (multi-set convolutional network) for queries containing both conjunction and disjunction operators. The goal is to represent an arbitrary logical expression in a continuous vector space by combining sub-expression vectors according to the operator type. We compared the proposed approach with the histogram-based approach on the real-world dataset and showed that our approach significantly outperforms histograms with a large margin.

Keywords: Recursive neural networks, Representation learning, Cardinality estimation, Compositionality, Complex logical expressions.

1. Introduction

Cardinality estimation is a key unit in query optimization. To choose the best execution plan, the query optimizer should precisely estimate the cardinality of an SQL query the number of rows in the table selected by the query without actual execution. Existing query optimizers in today's database management systems still select poor execution strategies. Their main drawback is that query optimizers make simplifying assumptions (e.g., column independence) about the underlying distribution of the relational table. These simplifying assumptions are helping to factorize the joint distribution into some low-dimensional representation using per column statistics, which are cheap to construct and store. When these assumptions do not hold, cardinality estimation errors occur, and the problem gets more complicated as the number of columns grows, leading to sub-optimal plan selections.

Recently, a number of machine and deep learning techniques have been successfully applied in many RDBMS (Relational Database Management System) applications, including

query optimization [1] - [6], indexing [7]. Attempts are being made to replace programmed heuristics with learned models. The advantage of these methods concluded in their ability to learn latent patterns of the dataset that are hard to find using rule-based heuristics.

This work was originally inspired by the recent work done by Kipf et al., 2018 [1]. The authors have developed a new deep learning approach (a multi-set convolutional network) to cardinality estimation. They show that deep learning models can learn complex interactions between predicates and even can capture join-crossing correlations.

Nevertheless, there remain challenging problems with complex queries, which can contain an arbitrary number of predicates connected by one of the Boolean operators (conjunction or disjunction). We use a recursive neural model [8] to approach this challenge by representing a logical expression with a dense vector which can be considered as a non-linear composition of sub-expression vectors.

Another challenge is related to the evaluation metric: mean q-error [9], which is the ratio between an estimate and the true cardinality. The relative factor can be the same for big and small cardinalities. For example, if the true and predicted cardinalities are equal to 10 and 5, the q-error will be 2. The same q-error will be obtained if the true and predicted cardinalities are equal to 10000 and 5000. It will be better to distinguish between mistakes made on big cardinalities mistakes made on small cardinalities. That is why we introduce two novel evaluation metrics for cardinality estimation: mean absolute interval error ($MAIR$) and interval accuracy (IA), which take into account not only the relative factor but also the absolute difference.

The major contributions of our work are as the followings: [1]

- We present a novel deep learning framework for cardinality estimation, which can handle complex queries.
- We introduce new evaluation metrics as well as a new objective function for cardinality estimation to distinguish between big and small mistakes.
- We compare the proposed model with a histogram-based approach and show the superiority of the proposed model.

The rest of this paper is organized as follows: A literature review on cardinality estimation is presented in Section 2. This is followed by the introduction of the proposed model and its essential aspects. Experimental evaluations and training details are presented in Section 4. This is followed by discussion and conclusions.

2. Related Work

Ivanov et al. [10] used query execution statistics of the previously executed queries to train famous machine learning models (e.g., KNN). Despite the simplicity, their approach does not take into account the semantic similarity between clauses, i.e., predicates are atomic objects.

Kipf et al. introduced [1] the multi-set convolutional network for cardinality estimation. They used set convolution to represent a set of tables, a set of joins and a set of predicates with fixed-length vectors, which are then fed as input to a regression module. They have

[1] https://github.com/naymaraq/SQL-Cardinality-Estimation.git

shown that it can learn (join-crossing) correlations pretty well. However, there are still open questions on dealing with complex predicates, which may include both conjunction and disjunction.

Another supervised method was proposed by Wang et al. [11], where an attempt was made to build a consistent and interpretable cardinality estimator by introducing a monotonic regression model w.r.t the query threshold.

To capture the multivariate distributions of relational tables, Yang et al. [6] proposed an unsupervised approach based on the deep autoregressive model combined with Monte Carlo sampling. The model approximates conditional densities, which are then used to evaluate the joint distribution.

In [12], the authors focused on more general queries, which may include DISTINCT, AND, OR, and NOT operators. They described a recursive algorithm to extend any cardinality estimation method that only handles conjunctive queries to one that works for more general queries, without changing the method itself. However, the proposed method has exponential complexity and it is tractable for queries, which have relatively small number of predicates.

Marcus et al. [13] presented flexible operator embeddings, to automatically map query operators to useful features tailored to a particular database which can be combined with machine learning models.

3. Proposed Approach

In this chapter, we will describe the key aspects of the proposed approach and explain how to use natural properties of logical expressions (compositionality and n-ary associativity) and inject these inductive properties into deep learning architecture, which will allow them to learn composable, permutations invariant functions. We will introduce the parsing algorithm (Permutation Invariant Parse Tree) to achieve compositionality as well as chosen deep learning architecture. Also, we briefly remind how to represent queries with multi-hot vectors as it was done in previous works [1, 5, 12].

3.1. Query Representation

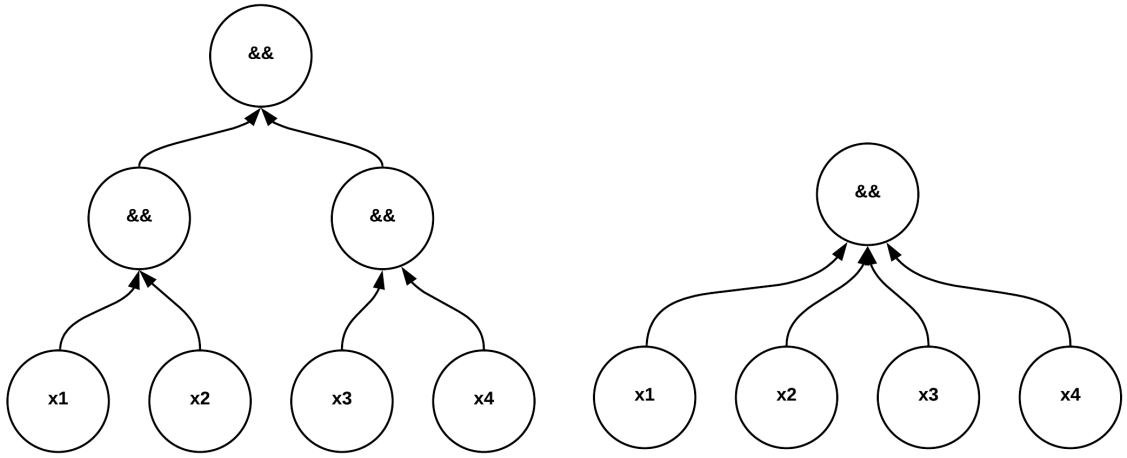
From now on, we focus only on SQL queries, which have the following form:

```
select count(*)
from table
where  logical_expression
```

where *logical_expression* consists of an arbitrary number of predicates, which may be connected with one of the Boolean operators $(\&\&, ||)$ and may be grouped using parentheses. Predicates are tuples with the form (c, op, v), where c corresponds to column name, op can be any comparison operator: $op \in \{<, \leq, =, \neq, \geq, >\}$, v is taken from the column domain.

Like the related works, we also represent queries with multi-hot vectors, according to which every predicate of the form (c, op, v) is encoded with multiple one-hot vectors one for each part of the predicate. More formally, the feature vector of predicate p is a concatenation of one-hot vectors of its parts: $x_p = [x_c, x_{op}, x_v]$. Here we assume that x_v is a one-hot vector, but in general, it could also be a numeric value. These vectors are used as leaf nodes and serve as inputs for TreeRNN (see Section 3.3.).

In addition, we will restrict our attention to single-table queries, i.e., in from field, there is always one table. That is why we do not need to encode tables and join attributes. It is worth noting that the proposed architecture can be easily extended for queries containing joins and multiple-relations, as was done in [1].



(a) Expression tree. (b) Permutation invariant parse tree.

Fig. 1. Expression and permutation invariant trees for $(x_1\&\&x_2)\&\&(x_3\&\&x_4)$.

3.2. Parsing

In this section, we would like to study the structure of functions operating on logical expressions. These functions take any valid logical expression as an input, and the output response range is a continuous space $\mathbb{R}$, as in the case of regression. A function f acting on logical expressions needs to be invariant with respect to a permissible permutation of predicates. In other words, the permutation invariability of function f can be defined recursively as follows: if x and y are sub-expressions, then

- $f(x||y) = f(y||x)$
- $f(x\&\&y) = f(y\&\&x)$

Thus, to construct the proper structure of function f, we extend the traditional expression tree[2] to an alternative and equivalent one: Permutation Invariant Parse Tree, where every node might have an arbitrary number of children. For example, a permutation invariant parse tree for $(x_1\&\&x_2)\&\&(x_3\&\&x_4)$ will look like just as the example shown in Fig. 1b. All the children of the internal node form a set, and the permutation of this set should not lead to any changes in the final output.

The algorithm of construction of such a tree is pretty straightforward: in the first step, the expression tree is being constructed from a prefix/postfix form of the logical expression. In the second step, we traverse from leaves to root by merging two internal nodes if they both have the same operator.

[2]An expression tree is a binary tree, in which each internal node corresponds to the operator. Each leaf node corresponds to an operand.

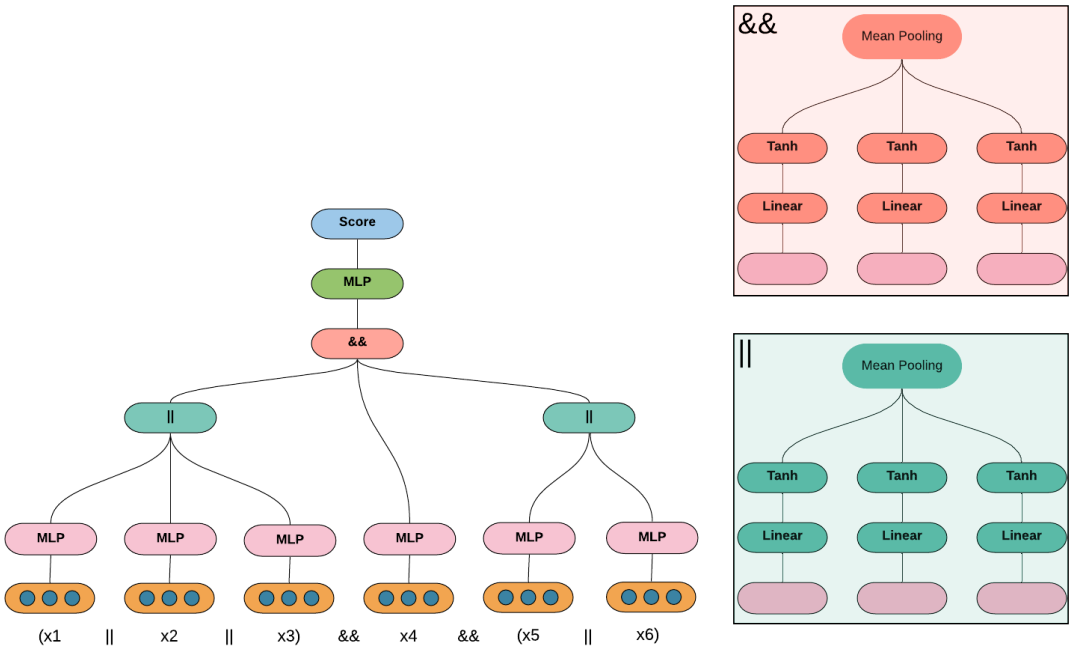


Fig. 2. The architecture of the model: During the forward propagation, the corresponding subnetwork adjusted to each internal node according to the operator (&&, ||). The activation of the root node acts as an input for the regression module.

3.3. Model Architecture

Due to the compositional nature of logical expression, we choose a tree-structured recursive neural network (TreeRNN) model [8], which has been successful in a number of NLP tasks, including sentiment analysis [14, 15], paraphrase detection [16], natural language inference [17], etc. Their success highly depends on their ability to capture semantic compositionality. In NLP, compositionality is the ability to express sentences of arbitrary length by combining phrases and words. Besides their ability to express compositionality with fixed-length representations, tree-structured models have justified themselves to learn logical deduction from reasonably-sized training sets. It is shown that these models can learn to identify logical relationships such as entailment and contradiction[18]. Also, TreeRNN models have been used to find the equivalence of arbitrary symbolic and boolean expressions by forcing the model to cluster its output to one location per equivalence class[19].

Given a tree described in Section 3.2., let $C(j)$ denote the set of children of node j and $N(j)$ denote the number of children of node j. The following equations describe the forward propagation of the model:

$$\tilde{h}_j = \frac{1}{N(j)} \sum_{k \in C(j)} \phi(h_k),$$

$$\phi(h_k) = \begin{cases} \phi_{\&\&}(h_k), & \text{if } type(j) = \&\& \\ \phi_{||}(h_k), & \text{if } type(j) = || \end{cases},$$

$$s = R(\tilde{h}_{\text{root}}). \tag{1}$$

where in (1), $\tilde{h}_{\text{root}}$ is a vector representation of root node, R is a shallow neural network that estimates the final cardinality score.

In Fig. 2, we illustrate the architecture of the proposed TreeNN model on a particular input query: $(x_1||x_2||x_3)\&\&x_4\&\&(x_5||x_6)$. During the forward propagation, the model uses a different type of subnetwork according to the operator ($\phi_{\&\&}$ or $\phi_{||}$). These two types of subnetworks are just shallow multilayer perceptrons (MLP) with the linear activation functions for the hidden layers and the hyperbolic tangent activation functions for the output layer. The outputs from MLP are averaged by mean pooling layer, making the activations of each tree node invariant to the corresponding child nodes permutations. The motivation of choosing mean pooling comes from Deep Sets [20], where Zaheer et al. introduce a network architecture, which can operate on sets. In particular, they prove a theorem, which claims that any function $f(X)$ operating on a set X is a valid set function, i.e., invariant to the permutation of instances in X, if it can be decomposed in the form $\rho(\sum_{x\in X}\phi(x))$, for suitable transformations ϕ and ρ. Finally, the activations of the root node ($\tilde{h}_{\text{root}}$) act as an input for another multilayer perceptron (R), which is responsible for cardinality estimation.

4. Experiments

4.1. Data Collection

First, we generate random queries based on the database schema and the values in it. Next, the true cardinalities are obtained by executing queries on the whole database. As we have mentioned before, we only examine single-table queries, i.e., a training sample consists of a logical expression defined in `where` field and the true cardinality. An example of a logical expression is shown in Fig. 3.

To generate predicates, we uniformly select a column from the database schema, then select the corresponding operation from $\{<, \leq, =, \neq, \geq, >\}$ and, finally, select a value from the column values.

We perform parallel query execution in Hadoop infrastructure, which gives us the fastest way to collect the training data.

```
(age>19 && age<30) && gender=female &&

&& (interest=17_1 || interest=c2_9 || interest=c16_6) &&

&& interest≠c1 && language=ru
```

Fig. 3. An example of generated *logical_expression*.

4.2. Evaluation Metrics

The traditional evaluation metric for cardinality estimation is the *mean q-error* [20], which shows how many times the predicted and the true cardinalities differ from each other. The drawback of q-error is that it scores big and small cardinalities in the same way. For instance, if the true and predicted cardinalities are equal to 10 and 5, respectively, the q-error will be

2. We will get the same q-error if the true and predicted cardinalities are equal to 10000 and 5000. Hence, it would be preferable to score these cases by considering also the absolute difference. That is why we introduce two novel metrics for cardinality estimation: *Mean Absolute Interval Error (MAIR)* and *Interval Accuracy (IA)*. To define these metrics, we first split the output range into K consecutive intervals. Let $G(x)$ denote the index of the interval, which contains x. Based on these intervals, $MAIR$ and IA are calculated as follows:

$$MAIR = \frac{1}{N}\sum_q |G(y) - G(\hat{y})|,$$

$$IA = \frac{\sum_q 1_{G(y)=G(\hat{y})}}{N},$$

where N is the number of samples, y and $\hat{y}$ are the true and predicted cardinalities. The higher the value of K, i.e., the smaller the intervals, the more reliable these metrics are.

Besides, we also report other evaluation metrics such as RMSE (root mean squared error), MAE (mean absolute error), MQE (mean q-error).

4.3. Training Details

4.3.1. Training Data

The training was conducted on $100K$ generated queries, from which $10K$ queries were used as a development set, and another $10K$ were used as a final evaluation (test) set. We also removed 0-cardinality examples from the development and test sets to have a fair evaluation. The log-normalization was used to normalize the outputs.

As each query has its unique parse tree, we cannot easily organize mini-batch training. So, in all our experiments, the training was conducted with a mini-batch size equal to 1, which leads to slow training.

4.3.2. Hyperparameter Search

Hyperparameter search was performed on the development set to choose the dimensionality of node vectors h_j and the dimensionality of layers in regression network R. Because of slow training, hyperparameter search was performed on the first 20 epochs using early stopping. The best result was achieved for $d_{h_j} = 32$ and $d_{l_R} = 80$.

4.3.3. Loss Function and Optimization

To overcome the drawback mentioned in Section 4.2., we construct the loss function, which consists of two parts: the relative factor and the squared difference. The first part is responsible for optimizing the relative difference, i.e., the q-error, and the second part is responsible for differentiating big and small mistakes.

$$L_q(y, \hat{y}) = \frac{\alpha}{2}(\frac{y}{\hat{y}} + \frac{\hat{y}}{y}) + (1-\alpha)(y-\hat{y})^2,$$

$$L = \frac{1}{N}\sum_q L_q(y, \hat{y}). \tag{2}$$

(a) Convergence of loss function with the number of epochs.

(b) MAIR and IA changes with the number of epochs.

Fig. 4. Model performance with the number of epochs.

The final training objective is to minimize the loss function written in (2). The loss is a convex combination of relative and squared difference terms. We set α equal to 0.99 to concentrate the central part of the loss on the relative part, plus an additional penalization score when huge mistakes were made.

We used Adam optimizer [21] with $\beta_1 = 0.9, \beta_2 = 0.99$ and $\epsilon = 10^{-8}$. We set the learning rate equal to 10^{-4}. The parameters, which achieve the smallest absolute interval error on the development set will be chosen for final evaluation.

Fig. 4a shows how the training and validation losses have decreased over epochs. It takes 30-35 epochs to converge. On the other hand, Fig. 4b shows how $MAIR$ and IA have changed over epochs. Both metrics are evaluated on the development set. It can be seen from Fig. 4 that minimizing the loss function leads to the smaller mean absolute interval error and the larger interval accuracy, which means that the TreeRNN is generalized successfully on the development set.

4.4. Results

4.4.1. Histogram

The histogram approach makes a simplifying assumption about column independence. The independence assumption is helping to factorize the joint distribution. In particular, one can derive inference rules based on that assumption:

$$P(A \wedge B) = P(A)P(B),$$

$$P(\neg A) = 1 - P(A),$$

$$P(A \vee B) = 1 - (1 - P(A))(1 - P(B)),$$

where A and B are predicates, P is the probability of a single predicate.

4.4.2. Quantitative Analysis

The performance of the proposed permutation invariant TreeRNN is compared with the baseline model. The appropriate performance metrics are given in Table 1. To calculate $MAIR$ and IA, we divide the output range into 15 intervals: $0 < 50 < 500 < 1000 < 2500 < 5000 < ... < 250000 < 400000 < 600000$. The expression is considered *short* if it is a combination of at most five predicates. The expression is considered *middle* if it is a combination of at least six but not more than 12 predicates. And, finally, the expression is considered *long* if it is a combination of at least 13 predicates.

One can see from Table 1 that the proposed model is superior to the baseline model with a clear margin. Due to their simple architecture, histograms are fast and accurate when dealing with short expressions. Long expressions are better learned with the TreeRNN model, as it takes into account the natural properties of logical expressions and can learn more complex interactions between predicates.

Table 1: Comparison of Histogram against the proposed TreeRNN. The integer in parentheses indicates the number of evaluated samples corresponding to each type.

Query Type	TreeRNN					Histogram				
	MQE	MAIR	IA	MAE	RMSE	MQE	MAIR	IA	MAE	RMSE
Short (313)	1.616	**0.294**	**0.719**	7811	14927	**1.36**	0.3	0.715	6077	11294
Middle (1353)	**1.718**	**0.363**	**0.66**	3661	9400	2.685	0.5	0.59	4846	13121
Long (4093)	**1.768**	**0.2**	**0.81**	594	2652	3.833	0.393	0.649	1006	4625
Overall (5759)	**1.747**	**0.243**	**0.77**	1707	6154	3.43	0.413	0.64	2184	7911

4.4.3. Monotonic Test

The ideal cardinality estimation system must be not only accurate but also consistent and interpretable. To explain it, let us look at the example: if query A is a subset of B, then the cardinality estimator should give more scores to B rather than A.

We make a simple experiment to understand the behaviour of the proposed model. We randomly remove one or two predicates from logical expressions in the test set and see how the output changes. If removed predicates are connected to others with && (||) operator, the output must be increased (decreased). We report the accuracy of successful outcomes.

Table 2 shows the results. Although we did not even demand that TreeRNN must be monotonous, it performs very well, especially for removed predicates, which are connected to others with && operator. Histograms, on the other side, are designed to be monotonous.

Table 2: Monotonic test result.

#Removed	TreeRNN		Histogram	
	&&	||	&&	||
1	92%	67%	100%	100%
2	94%	78%	100%	100%

5. Conclusion

We have presented a novel cardinality estimation system using recursive tree-structured neural networks. The proposed TreeRNN model is capable of taking complex tree-structured data and is able to learn underlying patterns of relational tables. Experimental evaluations show that our method outperforms the well-known histogram method with a clear margin. This is achieved by injecting inductive biases into the model architecture, which allows us to learn composable and permutations invariant functions.

References

[1] A. Kipf, T. Kipf, B. Radke, V. Leis, P. Boncz and A. Kemper, *Learned Cardinalities: Estimating Correlated Joins with Deep Learning*, arXiv preprint arXiv:1809.00677, 2018.

[2] S. Krishnan, Z. Yang, K. Goldberg, J. Hellerstein and I. Stoica, *Learning to Optimize Loin Queries with Deep Reinforcement Learning*, arXiv preprint arXiv:1808.03196, 2018.

[3] R. Marcus and O. Papaemmanouil, "Deep reinforcement learning for join order enumeration", *In Proceedings of the First International Workshop on Exploiting Artificial Intelligence Techniques for Data Management*, pp. 1-4, 2018.

[4] J. Ortiz, M. Balazinska, J. Gehrke, and S. S. Keerthi, "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.

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

[6] Z. Yang, E. Liang, A. Kamsetty, C. Wu, Y. Duan, X. Chen, P. Abbeel, J. M. Hellerstein, S. Krishnan, and I. Stoica, *Deep unsupervised cardinality estimation*, arXiv preprint arXiv:1905.04278, 2019.

[7] T. Kraska, A. Beutel, E. H. Chi, J. Dean and N. Polyzotis, "The case for learned index structures", *In Proceedings of the 2018 International Conference on Management of Data*, pp. 489-504, 2018.

[8] C. Goller and A. Kuchler, "Learning task-dependent distributed representations by backpropagation through structure", *In Proceedings of International Conference on Neural Networks (ICNN96)*, IEEE,vol. 1, pp. 347-352, 1996.

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

[10] O. Ivanov and S. Bartunov, *Adaptive cardinality estimation*, arXiv preprint arXiv:1711.08330, 2017.

[11] Y.Wang, C.Xiao, J.Qin, X.Cao, Y.Sun, W.Wang, and M.Onizuka, "Monotonic cardinality estimation of similarity selection: A deep learning approach", *In Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pp. 1197-1212, 2020.

[12] R. Hayek and O. Shmueli, *Nn-based transformation of any sql cardinality estimator for handling distinct, and, or and not*, arXiv preprint arXiv:2004.07009, 2020.

[13] R.Marcusand O.Papaemmanouil, *Flexible operator embeddings via deep learning*, arXiv preprint arXiv:1901.09090, 2019.

[14] O. Irsoy and C. Cardie, "Deep recursive neural networks for compositionality in language", *In Advances in Neural Information Processing Systems*, pp. 2096-2104, 2014.

[15] R. Socher, J. Pennington, E. H. Huang, A. Y. Ng and C. D. Manning, "Semi-supervised recursive autoencoders for predicting sentiment distributions, *In Proceedings of the 2011 conference on empirical methods in natural language processing*, pp. 151-161, 2011.

[16] R. Socher, E. H. Huang, J. Pennin, C. D. Manning and A. Y. Ng, "Dynamic pooling and unfolding recursive autoencoders for paraphrase detection", *In Advances in Neural Information Processing Systems*, pp. 801-809, 2011.

[17] I. Dagan, O. Glickman and B. Magnini, "The pascal recognising textual entailment challenge", *in Machine Learning Challenges Workshop*, Springer, pp. 177-190, 2005.

[18] S. Bowman, C. Potts and C. D. Manning, "Recursive neural networks can learn logical semantics", *In Proceedings of the 3rd workshop on continuous vector space models and their compositionality*, pp. 12-21, 2015.

[19] M. Allamanis, P. Chanthirasegaran, P. Kohli and C. Sutton, "Learning continuous semantic representations of symbolic expressions", *In International Conference on Machine Learning*, pp. 80-88, 2017.

[20] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov and A. J. Smola, "Deep sets", *In Advances in Neural Information Processing Systems*, pp. 3391-3401, 2017.

[21] D.P.Kingma and J.Ba, *Adam:A method for stochastic optimization*, arXiv preprint arXiv:1412.6980, 2014.

Submitted 10.06.2020, accepted 04.11.2020.

Հարցման հզորության մոտարկումը ռեկուրսիվ նեյրոնային ցանցերի միջոցով

Դավիթ Ս. Քարամյան

Երևանի պետական համալսարան
e-mail: davkar98@gmail.com

Ամփոփում

Պրեդիկատների միջև բարդ կապերը սովորելու և Էս-Քյու-Էլ հարցումների հզորությունը ճշգրիտ գնահատելու համար առաջարկվում է նոր մոտեցում, որը հիմնված է ռեկուրսիվ նեյրոնային ցանցերի վրա, որոնք հաշվի են առնում տրամաբանական արտահայտությունների բնական հատկությունները, բաղադրականությունը և ասոցիատիվությունը: Առաջարկվող ճարտարապետությունը MSCN-ի (Multi-Set Convolutional Network) ընդլայնումն է դիզունկցիա և կոնյուկցիա օպերատորներ պարունակող հարցումների համար: Նպատակն է ներկայացնել կամայական տրամաբանական արտահայտություն անընդհատ վեկտորային տարածության մեջ՝ համատեղելով ենթաարտահայտությունների վեկտորները՝ ըստ օպերատորի տեսակի: Առաջարկվող մոտեցումը համեմատվել է հիստոգրամային մոտեցման հետ և ցույց է տրվել, որ այն զգալիորեն գերազանցում է հիստոգրամային մոտեցմանը

Բանալի բառեր՝ ռեկուրսիվ նեյրոնային ցանցեր, ներկայացուցչական ուսուցում, հարցման հզորություն, բաղադրականություն, բարդ տրամաբանական արտահայտություններ:

Оценка мощности SQL-запроса с помощью рекурсивных нейронных сетей

Давид С. Карамян

Ереванский государственный университет
e-mail: davkar98@gmail.com

Аннотация

Чтобы изучить сложные взаимодействия между предикатами и точно оценить мощность SQL-запроса, была сконструирована новая структура, основанная на рекурсивных нейронных сетях с древовидной архитектурой, которые учитывают естественные свойства логических выражений: композиционность и n-арную ассоциативность. Предлагаемая архитектура является расширением MSCN(Multi-Set Convolutional Network) для запросов, содержащих операторы конъюнкции и дизъюнкции. Цель состоит в том, чтобы представить произвольное логическое выражение в непрерывном векторном пространстве путем объединения векторов подвыражений в соответствии с типом оператора. Предлагаемый подход был сравнен с подходом, основанным на гистограммах, на реальном наборе данных и было показано, что предложенный подход значительно превосходит гистограммы.

Ключевые слова: рекурсивные нейронные сети, репрезентативное обучение, мощность SQL-запросов, композиционность, сложные логические выражения.