COUPLING DISTRIBUTED AND SYMBOLIC EXECUTION FOR NATURAL LANGUAGE QUERIES

Lili Mou,[1] **Zhengdong Lu,**[2] **Hang Li,**[3] **Zhi Jin**[1]
[1]Key Laboratory of High Confidence Software Technologies (Peking University), MoE
Institute of Software, Peking University, China
[2]DeeplyCurious.ai [3]Noah's Ark Lab, Huawei Technologies
doublepower.mou@gmail.com, luz@DeeplyCurious.ai
HangLi.HL@huawei.com, zhijin@sei.pku.edu.cn

ABSTRACT

In this paper, we propose to combine neural execution and symbolic execution to query a table with natural languages. Our approach makes use the differentiability of neural networks and transfers (imperfect) knowledge to the symbolic executor before reinforcement learning. Experiments show our approach achieves high learning efficiency, high execution efficiency, high interpretability, as well as high performance.[1]

1 INTRODUCTION

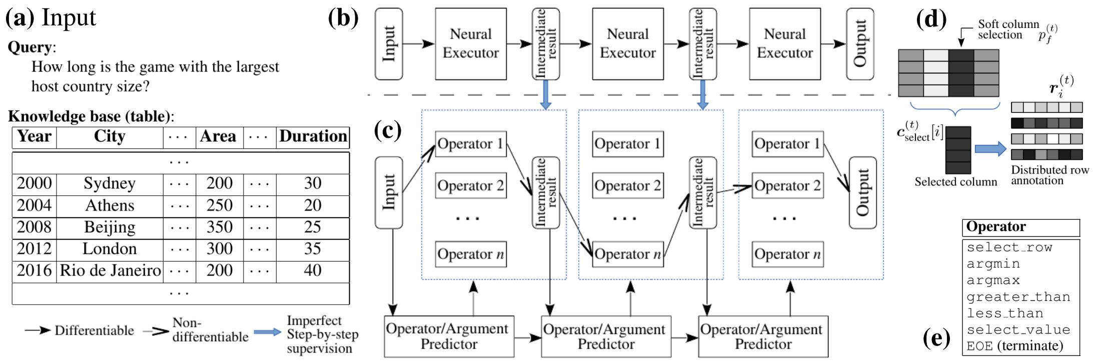
Using natural language to query a knowledge base has wide applications in question answering (Yin et al., 2016a), human-computer conversation (Wen et al., 2016), etc. Fig. 1a illustrates an example. Such task is also known as *semantic parsing*. An emerging research topic in semantic parsing is to query a knowledge base by neural networks. Because queries can be composited in a highly complicated manner, neural enquirers necessitate multiple steps of execution. The difficulty then lies in the lack of step-by-step supervision. In other words, we only assume groundtruth denotations are available in realistic settings; that we do not know execution sequences or intermediate results.

Yin et al. (2016b) propose a fully distributed neural enquirer, comprising several neuralized execution layers of field attention, row annotation, etc. The model is differentiable and end-to-end learnable, but it lacks explicit interpretation and is not efficient in execution. Neelakantan et al. (2016) propose a neural programmer by defining a set of symbolic operators (e.g., `argmax`); at each step, all possible execution results are fused by softmax. The step-by-step fusion is accomplished by weighted sum and the model is trained with mean square error. Such approaches work with numeric tables, but not with other operations like string matching; it also suffers from the problem of "exponential numbers of combinatorial states." Liang et al. (2016) train a symbolic executor by REINFORCE, but it is known that REINFORCE is sensitive to the initial policy. It could be very difficult to get started with a random initial policy; thus it only works with simple cases.

In this paper, we propose to couple distributed and symbolic execution for natural language queries. Our intuition rises from the observation that a fully distributed/neuralized executor also exhibits some (imperfect) symbolic interpretation. For example, the field attention gadget in Yin et al. (2016b) generally aligns with column selection. We therefore use the distributed model's intermediate execution results as supervision signals to pretrain a symbolic executor. Guided by such imperfect step-by-step supervision, the symbolic executor learns a fairly meaningful initial policy, which largely alleviates the cold start problem of the REINFORCE algorithm. Moreover, the improved policy can be fed back to the distributed enquirer to improve the neural network's performance.

To the best of our knowledge, we are the first to couple distributed and symbolic execution for semantic parsing. Our work is related to (but different from) several other studies of incorporating neural networks with external mechanisms Ling et al. (2015); Hu et al. (2016); Lei et al. (2016); Mi et al. (2016), where additional knowledge is used to improve neural networks' performance. Our

[1]The full version of this paper is available at https://arxiv.org/pdf/1612.02741.pdf



(a) Input

Query:
How long is the game with the largest host country size?

Knowledge base (table):

Year	City	···	Area	···	Duration
		···			
2000	Sydney	···	200	···	30
2004	Athens	···	250	···	20
2008	Beijing	···	350	···	25
2012	London	···	300	···	35
2016	Rio de Janeiro	···	200	···	40
		···			

Differentiable
Non-differentiable
Imperfect Step-by-step supervision
(b)
Input
Neural Executor
Intermediate result
Output
(c)
Operator 1
Operator 2
Operator n
Operator/Argument Predictor
(d)
Soft column selection $p_f^{(t)}$
$\boldsymbol{r}_i^{(t)}$
$\boldsymbol{c}_{\text{select}}^{(t)}[i]$
Selected column
Distributed row annotation
(e)

Operator
`select_row`
`argmin`
`argmax`
`greater_than`
`less_than`
`select_value`
`EOE` (terminate)

Figure 1: (a) Input, i.e., a query and a table. (b) Distributed enquirer. (c) Symbolic executor. (d) An execution layer in the distributed enquirer. (e) Primitive operators for symbolic execution.

main idea works in an opposite way: we first utilize the differentiability of neural networks to learn meaningful (although imperfect) intermediate execution steps, and then guide an external symbolic system, which is more natural to the semantic parsing task. Our study also sheds light on neural sequence prediction in general.

2 APPROACH

• **Distributed Enquirer.** The distributed enquirer makes full use of neural networks for table querying, where all semantic units (words, tables entries, etc.) are represented as real-valued vectors and processed by neural networks. It consists of the following main components.

- *Query encoder*. A bi-directional recurrent neural net (RNN) goes through word embeddings in a sentence. Bi-RNNs' last states in both directions are concatenated as the query representation $\boldsymbol{q}$.
- *Table encoder*. For a cell c with its column name being f, the cell vector $\boldsymbol{c}$ is the concatenation of the embeddings of c and f, further processed by a multi-layer perceptron (MLP).
- *Executor*. The distributed neural enquirer comprises several layers of execution. In each execution step, the neural network annotates each row with a vector, i.e., an embedding (Fig. 1d). The row vector can be intuitively thought of as row selection in query execution, but is represented by distributed semantics here. In the final execution layer, a softmax classifier is applied to the entire table to select a cell as the answer. We further describe the executor as follows.

Let $\boldsymbol{r}_i^{(t-1)}$ be the previous step's row annotation results, where the subscript i indexes a particular row. We summarize global execution information (denoted as $\boldsymbol{g}^{(t-1)}$) by max-pooling the row annotation $\boldsymbol{r}^{(t-1)}$, i.e., $\boldsymbol{g}^{(t-1)} = \text{MaxPool}_i\{\boldsymbol{r}_i^{(t-1)}\}$.

In the current execution step, we first compute a distribution $p_f^{(t)}$ over all fields as "soft" field selction. The computation is based on the query $\boldsymbol{q}$, the previous global information $\boldsymbol{g}^{(t-1)}$, and the field name embeddings $\boldsymbol{f}$, i.e.,

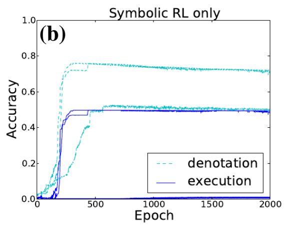
$$p_{f_j}^{(t)} = \text{softmax}\left(\exp\{\text{MLP}([\boldsymbol{q}; \boldsymbol{f}_j; \boldsymbol{g}^{(t-1)}])\}\right) \tag{1}$$

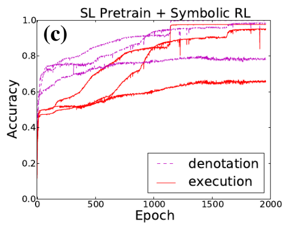
We represent the selected cell in each row as the sum of all cells in that row, weighted by soft field selection, i.e., $\boldsymbol{c}_{\text{select}}^{(t)}[i] = \sum_j p_{f_j}^{(t)} \boldsymbol{c}_{ij}$. The current row annotation is computed by another MLP, given by $\boldsymbol{r}_i^{(t)} = \text{MLP}\left([\boldsymbol{q}, \boldsymbol{g}^{(t-1)}, \boldsymbol{r}^{(t-1)}, \boldsymbol{c}_{\text{select}}^{(t)}[i]]\right)$. As said, the last execution layer applies a softmax classifier over all cells to select an answer. The probability of choosing the i-th row, j-th column is $p_{ij} = \text{softmax}\left(\exp\{\text{MLP}(\boldsymbol{q}, \boldsymbol{g}^{(t-1)}, \boldsymbol{r}_i^{(t-1)}, \boldsymbol{c}_{ij})\}\right)$.

• **Symbolic Executor.** The methodology of designing a symbolic executor is to define a set of primitive operators for the task, and then to use a machine learning model to predict the operator sequence and its arguments.

- *Primitive Operators*. We design six operators for symbolic execution. The result of one-step execution is a boolean scalar, indicating whether a row is selected after a step of execution. The symbolic execution takes previous results as input, with a column/field being the argument. Blue boxes in Fig. 1c illustrate the process and Fig. 1e summarizes our primitive operator set.

(a) Query type	SEMPRE•	Denotation Distributed•	Denotation Symbolic	Denotation Coupled	Execution Distributed	Execution Symbolic	Execution Coupled
`SelectWhere`	93.8	96.2	99.2	**99.6**	–	99.1	**99.6**
`Superlative`	97.8	98.9	**100.0**	**100.0**	–	**100.0**	**100.0**
`WhereSuperlative`	34.8	80.4	51.9	**99.9**	–	0.0	**91.0**
`NestQuery`	34.4	60.5	52.5	**100.0**	–	0.0	**100.0**
Overall	65.2	84.0	75.8	**99.8**	–	49.5	**97.6**





(d)	Fully Distributed	Our approach Op/Arg Pred.	Our approach Symbolic Exe.	Our approach Total
CPU	13.86	2.65	0.002	2.65
GPU	1.05	0.44		0.44

(e) Training Method	Accuracy (%)
End-to-end (w/ denotation labels)•	84.0
Step-by-step (w/ execution labels)•	96.4
Feeding back	96.5

Figure 2: Results. (a) Denotation and execution test accuracy, where Distributed and Symbolic columns refer to the results within a single world. (b) Learning curves of REINFORCE only. (c) Learning curves of REINFORCE with imperfect supervised pretraining. (d) Execution time (in seconds). (e) Test accuracy, when we feed back the well-trained symbolic executor's knowledge to the distributed one. "•" refers to results reported in Yin et al. (2016b).

- *Operator/Argument Predictors.* We also leverage RNNs to predict the operator and its argument (a column). The predicted probability of an operator i is $p_{\text{op}_i}^{(t)} = \text{softmax}\{\boldsymbol{w}_{\text{op}_i}^{(\text{out})\top}\boldsymbol{h}_{\text{op}}^{(t-1)}\}$; the operator with the largest predicted probability is selected for execution. Likewise, another RNN predicts the field selection, i.e.,

$$\boldsymbol{h}_{\text{field}}^{(t-1)} = \text{sigmoid}(W_{\text{field}}^{(\text{rec})}\boldsymbol{h}_{\text{field}}^{(t-1)}), \qquad p_{f_j}^{(t)} = \text{softmax}\left\{\boldsymbol{f}_j^\top \boldsymbol{h}_{\text{field}}^{(t-1)}\right\} \tag{2}$$

• **A Unified View.** We observe that the field attention in Eqn. 1 generally aligns with column selection in Eqn. 2. We therefore pretrain the column selector in the symbolic enquirer with labels predicted by a fully neuralized enquirer. Such pretraining can obtain up to 70% accurate field selection and largely reduce the search space during reinforcement learning. After obtaining a meaningful, albeit imperfect, initial policy, we apply REINFORCE (Sutton & Barto, 1998) to improve the policy.

After policy improvement by REINFORCE, we could further feed back the symbolic executor's intermediate results to the distributed one, akin to step-by-step supervised training. The loss is a combination of denotation cross entropy loss and field attention cross entropy loss.

3 EXPERIMENTS

Dataset. We evaluated our approach on a QA dataset in Yin et al. (2016b). The dataset comprises 25k different tables and queries. The validation and test sets contain 10k samples, respectively, and do not overlap with the training data. Each table is of size 10×10; the queries can be divided into four types: `SelectWhere`, `Superlative`, `WhereSuperlative`, and `NestQuery`, requiring 2–4 execution steps (`EOE` excluded).

Results. As we see in Fig. 2a, both distributed and symbolic enquirers outperform the traditional SEMPRE system; the coupled approach also significantly outperforms either of them. If trained solely by reinforcement learning, the symbolic executor can recover the execution sequences for simple questions (50%). However, for more complicated queries, it only learns last one or two steps of execution and has trouble in recovering early steps. This results in low execution accuracy but near 50% denotation accuracy. In our scenario, we still have half chance to obtain an accurate denotation even if the nested (early) execution is wrong—the ultimate result is either in the candidate list or not, given a wrong where-clause execution.

Figs. 2b and 2c show that our coupling approach largely accelerates the symbolic executor's learning process, whereas Fig. 2d presents the time consumption of execution of the test test. When we feed back the symbolic executor's knowledge to the distributed one, we also obtain a higher performance than end-to-end learning by neural networks (Fig. 2e), showing further evidence that the neural and symbolic worlds can be coupled well.

In summary, our approach makes use of both the distributed and symbolic worlds, and achieves high learning efficiency, high execution efficiency, high interpretability, as well as high performance.

References

Zhiting Hu, Xuezhe Ma, Zhengzhong Liu, Eduard Hovy, and Eric Xing. Harnessing deep neural networks with logic rules. In *ACL*, pp. 2410–2420, 2016.

Tao Lei, Regina Barzilay, and Tommi Jaakkola. Rationalizing neural predictions. In *EMNLP*, pp. 107–117, 2016.

Chen Liang, Jonathan Berant, Quoc Le, Kenneth D Forbus, and Ni Lao. Neural symbolic machines: Learning semantic parsers on freebase with weak supervision. *arXiv preprint arXiv:1611.00020*, 2016.

Wang Ling, Isabel Trancoso, Chris Dyer, and Alan W Black. Character-based neural machine translation. *arXiv preprint arXiv:1511.04586*, 2015.

Haitao Mi, Baskaran Sankaran, Zhiguo Wang, and Abe Ittycheriah. Coverage embedding models for neural machine translation. In *EMNLP*, pp. 955–960, 2016.

Arvind Neelakantan, Quoc V Le, and Ilya Sutskever. Neural programmer: Inducing latent programs with gradient descent. In *ICLR*, 2016.

Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*, volume 1. MIT Press Cambridge, 1998.

Tsung-Hsien Wen, Milica Gasic, Nikola Mrksic, Lina M Rojas-Barahona, Pei-Hao Su, Stefan Ultes, David Vandyke, and Steve Young. A network-based end-to-end trainable task-oriented dialogue system. *arXiv preprint arXiv:1604.04562*, 2016.

Jun Yin, Xin Jiang, Zhengdong Lu, Lifeng Shang, Hang Li, and Xiaoming Li. Neural generative question answering. In *IJCAI*, pp. 2972–2978, 2016a.

Pengcheng Yin, Zhengdong Lu, Hang Li, and Ben Kao. Neural enquirer: Learning to query tables with natural language. In *IJCAI*, pp. 2308–2314, 2016b.