NEURAL ENQUIRER: LEARNING TO QUERY TABLES IN NATURAL LANGUAGE

Pengcheng Yin[1], **Zhengdong Lu**[2], **Hang Li**[2] **& Ben Kao**[1]
[1]Department of Computer Science, The University of Hong Kong
{pcyin,kao}@cs.hku.hk
[2]Noahs Ark Lab, Huawei Technologies
{Lu.Zhengdong,HangLi.HL}@huawei.com

ABSTRACT

We propose NEURAL ENQUIRER — a neural network architecture for answering natural language (NL) questions given a knowledge base (KB) table. Unlike previous work on end-to-end training of semantic parsers, NEURAL ENQUIRER is fully "neuralized": it gives distributed representations of queries and KB tables, and executes queries through a series of differentiable operations. The model can be trained with gradient descent using both end-to-end and step-by-step supervision. During training the representations of queries and the KB table are jointly optimized with the query execution logic. Our experiments show that the model can learn to execute complex NL queries on KB tables with rich structures.

1 INTRODUCTION

Natural language dialogue and question answering often involve querying a knowledge base (Wen et al., 2015; Berant et al., 2013). The traditional approach involves two steps: First, a given query $\tilde{Q}$ is semantically parsed into an "executable" representation, which is often expressed in certain logical form $\tilde{Z}$ (e.g., SQL-like queries). Second, the representation is executed against a KB from which an answer is obtained. For queries that involve complex semantics and logic (e.g., "*Which city hosted the longest Olympic game before the game in Beijing?*"), semantic parsing and query execution become extremely complex. For example, carefully hand-crafted features and rules are needed to correctly parse a complex query into its logical form (see example in the lower-left corner of Figure 1). To partially overcome this complexity, recent works (Clarke et al., 2010; Liang et al., 2011; Pasupat & Liang, 2015) attempt to "backpropagate" query execution results to revise the semantic representation of a query. This approach, however, is greatly hindered by the fact that traditional semantic parsing mostly involves rule-based features and symbolic manipulation, leaving only a handful of tunable parameters to cater to the great flexibility of natural language.

In this paper we propose NEURAL ENQUIRER — a neural network system that learns to understand NL queries and execute them on a KB table from examples of queries and answers. Unlike similar efforts along this line of research (Neelakantan et al., 2015), NEURAL ENQUIRER is a fully neuralized, end-to-end differentiable network that jointly models semantic parsing and query execution. It encodes queries and KB tables into distributed representations, and executes compositional queries against the KB through a series of differentiable operations. The model is trained using query-answer pairs, where the distributed representations of queries and the KB are optimized together with the query execution logic in an end-to-end fashion. We demonstrate using a synthetic QA task that NEURAL ENQUIRER is capable of learning to execute complex compositional NL questions.

2 MODEL

Given an NL query Q and a KB table $\mathcal{T}$, NEURAL ENQUIRER executes Q against $\mathcal{T}$ and outputs a ranked list of answers. The execution is done by first using *Encoders* to encode the query and table into distributed representations, which are then sent to a cascaded pipeline of *Executors* to derive the answer. Figure 1 gives an illustrative example. It consists of the following components:

Query Encoder abstracts the semantics of an NL query Q and encodes it into a query embedding $\mathbf{q}$. Let $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T\}$ be the embeddings of the words in Q, where $\mathbf{x}_t \in \mathbb{R}^{d_\mathcal{W}}$ is from an embedding matrix $\mathbf{L}$. We employ a bidirectional GRU to summarize the sequence of word embeddings in forward and reverse orders. $\mathbf{q}$ is formed by concatenating the last hidden states in the two directions.

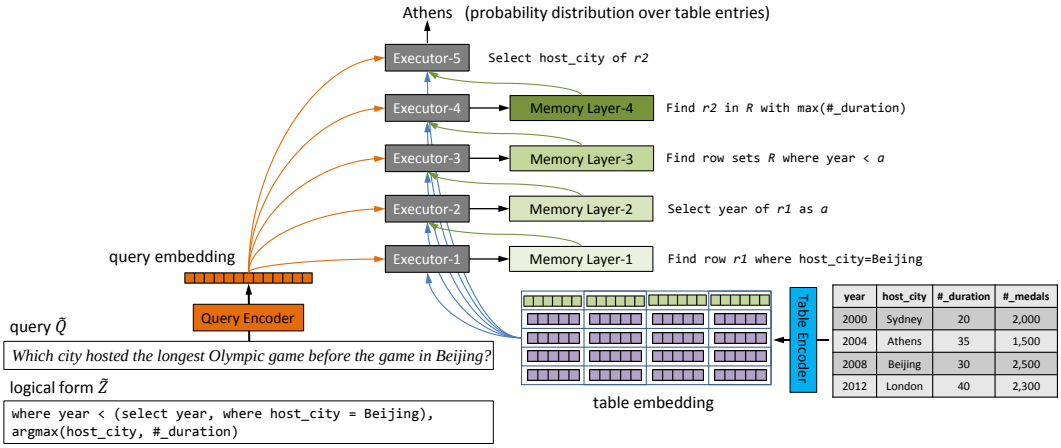

Figure 1: An overview of NEURAL ENQUIRER with five executors

Table Encoder derives a table embedding by encoding entries in the table $\mathcal{T}$ into distributed vectors. Suppose $\mathcal{T}$ has M rows and N columns. In our model, the n-th column is associated with a *field name* (e.g., `host_city`). Each cell value is a word (e.g., *Beijing*) in the vocabulary. We use w_{mn} to denote the cell value in row m column n, and $\mathbf{w}_{mn}$ to denote its embedding. Let $\mathbf{f}_n$ be the embedding of the field name for column n. For each entry (cell) w_{mn}, Table Encoder computes a $\langle$field, value$\rangle$ composite embedding $\mathbf{e}_{mn} \in \mathbb{R}^{d\varepsilon}$ by fusing $\mathbf{f}_n$ and $\mathbf{w}_{mn}$ using a Deep Neural Network (DNN):

$$\mathbf{e}_{mn} = \mathsf{DNN}_0(\mathbf{f}_n, \mathbf{w}_{mn}) = \tanh(\mathbf{W} \cdot [\mathbf{f}_n; \mathbf{w}_{mn}] + \mathbf{b}),$$

where $[\cdot;\cdot]$ denotes vector concatenation.

Executor executes a query against the table and outputs *annotations* that encode intermediate execution results. Each executor models a specific type of operation conditioned on the query. Figure 1 illustrates the operation each executor is assumed to perform in answering $\tilde{Q}$. The query is executed sequentially through a stack of executors. Such a cascaded architecture enables the model to answer complex compositional queries. An executor at Layer-ℓ (denoted as Executor-ℓ) consists of two major neural network components: a *Reader* and an *Annotator*. The executor processes a table row-by-row. For the m-th row with N $\langle$field, value$\rangle$ composite embeddings $\mathcal{R}_m = \{\mathbf{e}_{m1}, \mathbf{e}_{m2}, \ldots, \mathbf{e}_{mN}\}$, the Reader fetches a read vector $\mathbf{r}_m^\ell$ from $\mathcal{R}_m$ via an attentive reading operation. The read vector is then sent to the Annotator to perform the actual execution. The output of the Annotator is a *row annotation* $\mathbf{a}_m^\ell$, which captures the row-wise local computation result. Once all row annotations are obtained, Executor-ℓ generates a *table annotation* $\mathbf{g}^\ell$ to summarize the global computation result on the whole table by pooling all row annotations. All the row and table annotations are saved in the memory Layer-ℓ: $\mathcal{M}^\ell = \{\mathbf{a}_1^\ell, \mathbf{a}_2^\ell, \ldots, \mathbf{a}_M^\ell, \mathbf{g}^\ell\}$, which is accessible to Executor-$(\ell+1)$. Specifically:

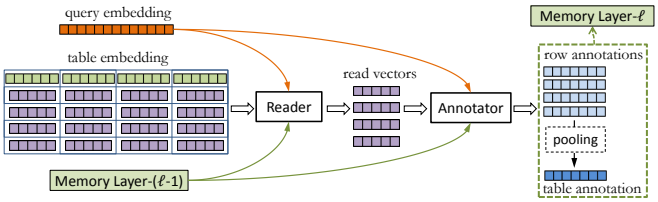

Figure 2: Overview of an Executor-ℓ

$$\text{Read Vector:} \quad \mathbf{r}_m^\ell = f_{\text{R}}^\ell(\mathcal{F}_\mathcal{T}, \mathbf{q}, \mathcal{M}^{\ell-1}, \mathcal{R}_m) = \textstyle\sum_{n=1}^N \tilde{\omega}(\mathbf{f}_n, \mathbf{q}, \mathbf{g}^{\ell-1})\mathbf{e}_{mn} \tag{1}$$

$$\text{Row Annotation:} \quad \mathbf{a}_m^\ell = f_{\text{A}}^\ell(\mathbf{r}_m^\ell, \mathbf{q}, \mathcal{M}^{\ell-1}) = \mathsf{DNN}_1^{(\ell)}([\mathbf{r}_m^\ell; \mathbf{q}; \mathbf{a}_m^{\ell-1}; \mathbf{g}^{\ell-1}]) \tag{2}$$

$$\text{Table Annotation:} \quad \mathbf{g}^\ell = f_{\text{MAXPOOLING}}(\mathbf{a}_1^\ell, \mathbf{a}_2^\ell, \ldots, \mathbf{a}_M^\ell) \tag{3}$$

where $\tilde{\omega}(\cdot)$ is the attention weight. Intuitively, row annotations handle operations that require only row-wise, local information (e.g., `select`, `where`), while table annotations model superlative operations (e.g., `max`, `min`) by aggregating table-wise, global execution results. A combination of row and table annotations enables the model to perform a wide variety of real-world query operations.

Instead of computing annotations based on read vectors, the last executor in NEURAL ENQUIRER directly outputs the probability of an entry w_{mn} in table $\mathcal{T}$ being the answer a:

$$p(a = w_{mn}|Q, \mathcal{T}) = \frac{\exp(\mathsf{DNN}_2^{(\ell)}([\mathbf{e}_{mn}, \mathbf{q}, \mathbf{a}_m^{\ell-1}, \mathbf{g}^{\ell-1}]))}{\sum_{m'=1,n'=1}^{M,N} \exp(\mathsf{DNN}_2^{(\ell)}([\mathbf{e}_{m'n'}, \mathbf{q}, \mathbf{a}_{m'}^{\ell-1}, \mathbf{g}^{\ell-1}]))} \tag{4}$$

① SELECT_WHERE [select F_a, where F_b = w_b]	③ WHERE_SUPERLATIVE [where F_a >|< w_a, argmax/min(F_b, F_c)]
▷ *How many people participated in the game in Beijing?*	▷ *How long was the game with the most medals that had fewer than 3,000 participants?*
▷ *In which city was the game hosted in 2012?*	▷ *How many medals were in the first game after 2008?*
② SUPERLATIVE [argmax/min(F_a, F_b)]	④ NEST [where F_a >|< (select F_a,where F_b=w_b), argmax/min(F_c, F_d)]
▷ *When was the latest game hosted?*	▷ *Which country hosted the longest game before the game in Athens?*
▷ *How big is the country which hosted the shortest game?*	▷ *How many people watched the earliest game that lasts for more days than the game in 1956?*

Table 1: Example queries for each query type, with annotated SQL-like logical form templates

	MIXTURED-25K			MIXTURED-100K	
	SEMPRE	N2N	SbS	N2N	SbS
SELECT_WHERE	93.8%	96.2%	99.7%	99.3%	100.0%
SUPERLATIVE	97.8%	98.9%	99.5%	99.9%	100.0%
WHERE_SUPERLATIVE	34.8%	80.4%	94.3%	98.5%	99.8%
NEST	34.4%	60.5%	92.1%	64.7%	99.7%
Overall Accuracy	65.2%	84.0%	96.4%	90.6%	99.9%

Table 2: Accuracies on MIXTURED datasets

Learning NEURAL ENQUIRER can be trained in an *end-to-end* (N2N) fashion. Given a set of $N_{\mathcal{D}}$ query-table-answer triples $\mathcal{D} = \{(Q^{(i)}, \mathcal{T}^{(i)}, y^{(i)})\}$, the model is optimized by maximizing the log-likelihood of gold-standard answers:

$$\mathcal{L}_{\text{N2N}}(\mathcal{D}) = \sum\nolimits_{i=1}^{N_{\mathcal{D}}} \log p(a = y^{(i)} | Q^{(i)}, \mathcal{T}^{(i)}) \tag{5}$$

The training can also be carried out with stronger guidance, i.e., *step-by-step* (SbS) supervision, by softly guiding the learning process via controlling the attention weights $\tilde{w}(\cdot)$ in Eq. (1). As an example, for Executor-1 in Figure 1, by biasing the attention weight of the `host_city` field towards 1.0, only the value of `host_city` will be fetched and sent to the Annotator. In this way we can "force" the executor to learn the `where` operation to find the row whose `host_city` is *Beijing*. Formally, this is done by introducing additional supervision signal to Eq. (5):

$$\mathcal{L}_{\text{SbS}}(\mathcal{D}) = \sum\nolimits_{i=1}^{N_{\mathcal{D}}} [\log p(a = y^{(i)} | Q^{(i)}, \mathcal{T}^{(i)}) + \alpha \sum\nolimits_{\ell=1}^{L-1} \log \tilde{w}(\mathbf{f}^{\star}_{i,\ell}, \cdot, \cdot)] \tag{6}$$

where α is a tuning weight, and L is the number of executors. $\mathbf{f}^{\star}_{i,\ell}$ is the embedding of the field known *a priori* to be used by Executor-ℓ in answering the i-th example.

3 A SYNTHETIC QA TASK

We present a synthetic QA task with a large number of QA examples at various levels of complexity. We generate two synthetic datasets, MIXTURED-25K and MIXTURED-100K. Each dataset consists of query-table-answer triples $\{(Q^{(i)}, \mathcal{T}^{(i)}, y^{(i)})\}$. Tables are in size 10×10, and are sampled from a synthetic schema of Olympic Games. We generate four types of NL queries at various compositional depths (see Table 1), ranging from simple SELECT_WHERE queries to more complex NEST ones.

We compare the performance of NEURAL ENQUIRER under N2N and SbS training settings with a state-of-the-art semantic parser, SEMPRE (Pasupat & Liang, 2015). Results are listed in Table 2. On MIXTURED-25K, the relatively low performance of SEMPRE indicates that our QA task, although synthetic, is highly nontrivial. Under N2N setting, NEURAL ENQUIRER outperforms SEMPRE on all types of queries, with significant gain on complex queries like WHERE_SUPERLATIVE and NEST. Under SbS setting, with stronger supervision, our model achieves nearly perfect accuracy.

To better understand the query execution process, we study the attention weights $\tilde{w}(\cdot)$ of Readers (Eq. 1) for intermediate executors, and the answer probability (Eq. 4) given by the last executor. Figure 3 visualizes the values for an example query Q_1 in N2N setting. The reference logical form Z_1 indicates that Q_1 involves five steps of execution. Each executor focuses on a specific type of operation, similar as the example in Figure 1. However, the attention weights for the first two executors are a bit obscure. We then investigate their corresponding values in SbS setting, and find that they are perfectly centered on the ideal fields as highlighted in red dashed rectangles.

Q_1: *Which country hosted the longest game before the game in Athens?*

Z_1: `where year < (select year,where host_city=Athens), argmax(host_country,#_duration)`

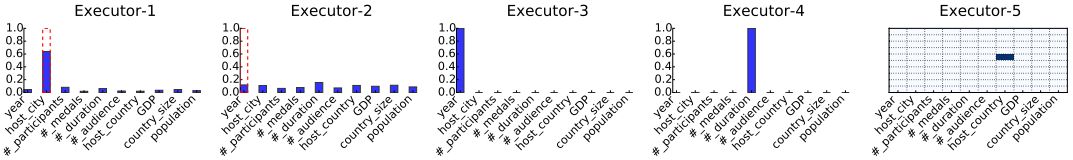

Figure 3: Weights visualization of query Q_1

REFERENCES

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. Semantic parsing on freebase from question-answer pairs. In *EMNLP*, pp. 1533–1544, 2013.

James Clarke, Dan Goldwasser, Ming-Wei Chang, and Dan Roth. Driving semantic parsing from the world's response. In *CoNLL*, pp. 18–27, 2010.

Percy Liang, Michael I. Jordan, and Dan Klein. Learning dependency-based compositional semantics. In *ACL (1)*, pp. 590–599, 2011.

Arvind Neelakantan, Quoc V. Le, and Ilya Sutskever. Neural programmer: Inducing latent programs with gradient descent. *CoRR*, abs/1511.04834, 2015.

Panupong Pasupat and Percy Liang. Compositional semantic parsing on semi-structured tables. In *ACL (1)*, pp. 1470–1480, 2015.

Tsung-Hsien Wen, Milica Gasic, Nikola Mrksic, Pei hao Su, David Vandyke, and Steve J. Young. Semantically conditioned lstm-based natural language generation for spoken dialogue systems. In *EMNLP*, pp. 1711–1721, 2015.