Neural Enquirer: Learning to Query Tables in Natural Language

Pengcheng Yin
The University of Hong Kong
pcyin@cs.hku.hk

Zhengdong Lu
Huawei Noah's Ark Lab
Lu.Zhengdong@huawei.com

Hang Li
Huawei Noah's Ark Lab
HangLi.HL@huawei.com

Ben Kao
The University of Hong Kong
kao@cs.hku.hk

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 Games before the Games 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 and 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, and is subject to intractable search space incurred by 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

Proceedings of 2016 NAACL Human-Computer Question Answering Workshop, pages 29–35,
San Diego, California, June 12-17, 2016.

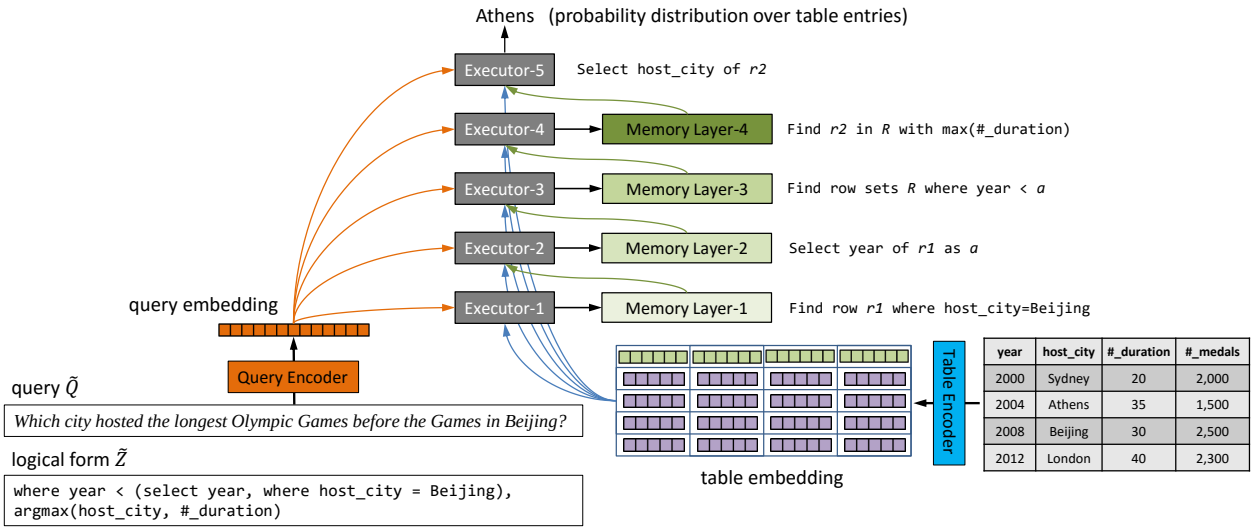


Figure 1: An overview of NEURAL ENQUIRER with five executors

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:

2.1 Query Encoder

Query Encoder abstracts the semantics of an NL query Q and encodes it into a query embedding $\mathbf{q} \in \mathbb{R}^{d_\mathcal{Q}}$. 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 Gated Recurrent Unit (GRU) (Bahdanau et al., 2015) 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.

We remark that Query Encoder can find the representation of a rather general class of symbol sequences, agnostic to the actual representation of the query (e.g., natural language, SQL, etc). The model is able to learn the semantics of input queries through end-to-end training, making it a generic model for query understanding and query execution.

2.2 Table Encoder

Table Encoder converts a KB table $\mathcal{T}$ into a distributed representation, which is used as an input to executors. 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 $\mathbf{e}_{mn} \in \mathbb{R}^{d_\mathcal{E}}$ by fusing $\mathbf{f}_n$ and $\mathbf{w}_{mn}$ through a non-linear transformation:

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

where $[\cdot;\cdot]$ denotes vector concatenation. The output of Table Encoder is an $M \times N \times d_\mathcal{E}$ tensor that consists of $M \times N$ embeddings, each of length $d_\mathcal{E}$.

2.3 Executor

NEURAL ENQUIRER executes an input query on a KB table through layers of execution. Each layer consists of an executor that, after learning, performs certain operation (e.g., `select`, `max`) relevant to the input query. An executor outputs intermediate execution results, referred to as *annotations*, which are saved in the external memory of the executor. A query is executed sequentially through a stack of executors. Such a cascaded architecture enables the model to answer complex, compositional queries. An example is given in Figure 1 in which descriptions of the operation each executor is assumed to perform for the query $\tilde{Q}$ are shown. We will demonstrate in Section 4 that the model is capable of learning the operation logic of each executor via end-to-end training.

As illustrated in Figure 2, 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. The

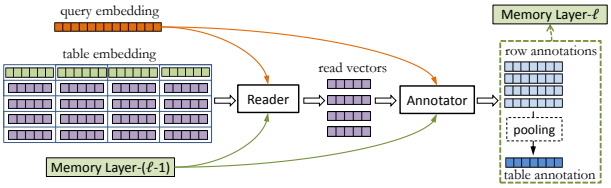


Figure 2: Overview of an Executor-ℓ

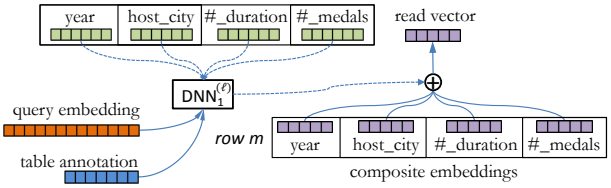


Figure 3: Illustration of the Reader in Executor-ℓ.

Reader reads in data from each row m in the form of a *read vector* $\mathbf{r}_m^\ell$, which 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 of Layer-ℓ: $\mathcal{M}^\ell = \{\mathbf{a}_1^\ell, \mathbf{a}_2^\ell, \ldots, \mathbf{a}_M^\ell, \mathbf{g}^\ell\}$. 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 query operations in real world scenarios.

2.3.1 Reader

As illustrated in Figure 3, for the m-th row with N ⟨field, value⟩ 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:

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

where $\mathcal{M}^{\ell-1}$ denotes the content of memory Layer-$(\ell-1)$, and $\mathcal{F}_\mathcal{T} = \{\mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_N\}$ is the set of field name embeddings. $\tilde{\omega}(\cdot)$ is the normalized attention weights given by:

$$\tilde{\omega}(\mathbf{f}_n, \mathbf{q}, \mathbf{g}^{\ell-1}) = \frac{\exp(\omega(\mathbf{f}_n, \mathbf{q}, \mathbf{g}^{\ell-1}))}{\sum_{n'=1}^{N} \exp(\omega(\mathbf{f}_{n'}, \mathbf{q}, \mathbf{g}^{\ell-1}))} \quad (1)$$

where $\omega(\cdot)$ is modeled as a Deep Neural Network (denoted as DNN$_1^{(\ell)}$). Since each executor models a specific type of computation, it should only attend to a subset of entries that are pertinent to its execution. This is modeled by the Reader. Our approach is related to the content-based addressing of Neural Turing Machines (Graves et al., 2014) and the attention mechanism in neural machine translation models (Bahdanau et al., 2015).

2.3.2 Annotator

The Annotator of Executor-ℓ computes row and table annotations based on read vectors fetched by the Reader. The results are stored in the ℓ-th memory layer $\mathcal{M}^\ell$ accessible to Executor-$(\ell+1)$. The last executor is the only exception, which outputs the final answer.

[Row annotations] Capturing row-wise execution result, the annotation $\mathbf{a}_m^\ell$ for row m in Executor-ℓ is given by

$$\mathbf{a}_m^\ell = f_{\text{A}}^\ell(\mathbf{r}_m^\ell, \mathbf{q}, \mathcal{M}^{\ell-1}) = \text{DNN}_2^{(\ell)}([\mathbf{r}_m^\ell; \mathbf{q}; \mathbf{a}_m^{\ell-1}; \mathbf{g}^{\ell-1}]).$$

DNN$_2^{(\ell)}$ fuses the corresponding read vector $\mathbf{r}_m^\ell$, the results saved in the previous memory layer (row and table annotations $\mathbf{a}_m^{\ell-1}$, $\mathbf{g}^{\ell-1}$), and the query embedding $\mathbf{q}$. Specifically,

- row annotation $\mathbf{a}_m^{\ell-1}$ represents the local status of execution before Layer-ℓ;
- table annotation $\mathbf{g}^{\ell-1}$ summarizes the global status of execution before Layer-ℓ;
- read vector $\mathbf{r}_m^\ell$ stores the value of attentive reading;
- query embedding $\mathbf{q}$ encodes the overall execution agenda.

All the above values are combined through DNN$_2^{(\ell)}$ to form the annotation of row m in the current layer.

[Table annotations] Capturing global execution state, a table annotation summarizes all row annotations via a global max pooling operation:

$$\mathbf{g}^\ell = f_{\text{MaxPool}}(\mathbf{a}_1^\ell, \mathbf{a}_2^\ell, \ldots, \mathbf{a}_M^\ell) = [g_1, g_2, \ldots, g_{d_\mathcal{G}}]^\top$$

year	host_city	#_participants	#_medals	#_duration	#_audience	host_country	GDP	country_size	population
2008	Beijing	4,200	2,500	30	67,000	China	2,300	960	130

Figure 4: An example table in the synthetic QA task (only one row shown)

① SELECT_WHERE `[select Fa, where Fb = wb]`	③ WHERE_SUPERLATIVE `[where Fa >|< wa, argmax/min(Fb, Fc)]`
▷ *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(Fa, Fb)]`	④ NEST `[where Fa >|< (select Fa,where Fb=wb), argmax/min(Fc, Fd)]`
▷ *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

where $g_k = \max(\{\mathbf{a}_1^\ell(k), \mathbf{a}_2^\ell(k), \ldots, \mathbf{a}_M^\ell(k)\})$ is the maximum value among the k-th elements of all row annotations.

2.3.3 Last Layer Executor

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:

$$p(a = w_{mn}|Q, \mathcal{T}) = \frac{\exp(f_{\text{ANS}}^\ell(\mathbf{e}_{mn}, \mathbf{q}, \mathbf{a}_m^{\ell-1}, \mathbf{g}^{\ell-1}))}{\sum_{m'=1,n'=1}^{M,N} \exp(f_{\text{ANS}}^\ell(\mathbf{e}_{m'n'}, \mathbf{q}, \mathbf{a}_{m'}^{\ell-1}, \mathbf{g}^{\ell-1}))} \quad (2)$$

where $f_{\text{ANS}}^\ell(\cdot)$ is modeled as a DNN ($\mathsf{DNN}_3^{(\ell)}$). Note that the last executor, which is devoted to returning answers, could still carry out execution in $\mathsf{DNN}_3^{(\ell)}$.

3 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:

$$\mathcal{L}_{\text{N2N}}(\mathcal{D}) = \sum_{i=1}^{N_\mathcal{D}} \log p(a = y^{(i)}|Q^{(i)}, \mathcal{T}^{(i)}) \quad (3)$$

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. (3):

$$\mathcal{L}_{\text{SbS}}(\mathcal{D}) = \sum_{i=1}^{N_\mathcal{D}} [\log p(a = y^{(i)}|Q^{(i)}, \mathcal{T}^{(i)}) + \alpha \sum_{\ell=1}^{L-1} \log \tilde{w}(\mathbf{f}_{i,\ell}^\star, \cdot, \cdot)] \quad (4)$$

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

4 Experiments

In this section we evaluate NEURAL ENQUIRER on synthetic QA tasks with NL queries of varying compositional depths.

4.1 Synthetic QA Task

We present a synthetic QA task with a large number of QA examples at various levels of complexity to evaluate the performance of NEURAL ENQUIRER. Starting with "artificial" tasks accelerates the development of novel deep models (Weston et al., 2015), and has gained increasing popularity in recent research on modeling symbolic computation using DNNs (Graves et al., 2014; Zaremba and Sutskever, 2014).

Our synthetic dataset consists of query-table-answer triples $\{(Q^{(i)}, \mathcal{T}^{(i)}, y^{(i)})\}$. To generate a triple, we first randomly sample a table $\mathcal{T}^{(i)}$ of size 10×10 from a synthetic schema of Olympic Games. The cell values of $\mathcal{T}^{(i)}$ are drawn from a vocabulary of 120 location names and 120 numbers. Figure 4 gives an example table. Next, we sample a query $Q^{(i)}$ generated using NL templates, and obtain its gold-standard answer $y^{(i)}$ on $\mathcal{T}^{(i)}$. Our task consists of four types of NL queries, with examples given in Table 1. We also give the logical form template

	MIXED-25K			MIXED-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 MIXED datasets

for each type of query. The templates define the semantics and compositionality of queries. We generate queries at various compositional depths, ranging from simple SELECT_WHERE queries to more complex NEST ones. This makes the dataset have similar complexity as a real-world one, except for the relatively small vocabulary. The queries are flexible enough to involve complex matching between NL phrases and logical constituents, which makes query understanding nontrivial: (1) the same field is described by different NL phrases (e.g., *"How big is the country ..."* and *"What is the size of the country ..."* for the `country_size` field); (2) different fields may be referred to by the same NL pattern (e.g, *"in China"* for `host_country` and *"in Beijing"* for `host_city`); (3) simple NL constituents may be grounded to complex logical operations (e.g., *"after the game in Beijing"* implies comparing between the values of `year` fields).

To simulate the read-world scenario where queries of various types are issued to the model, we construct two MIXED datasets, with $25K$ and $100K$ training examples respectively, where four types of queries are sampled with the ratio $1:1:1:2$. Both datasets share the same testing set of $20K$ examples, $5K$ for each type of query. We enforce that no tables and queries are shared between training/testing sets.

4.2 Setup

[Tuning] We adopt a model with five executors. The lengths of hidden states for GRU and DNNs are 150, 50. The numbers of layers for $\mathsf{DNN}_1^{(\ell)}$, $\mathsf{DNN}_2^{(\ell)}$ and $\mathsf{DNN}_3^{(\ell)}$ are 2, 3, 3. The length of word embeddings and annotations is 20. α is 0.2. We train the model using ADADELTA (Zeiler, 2012) on a Tesla K40 GPU. The training converges fast within 2 hours.

[Metric] We evaluate in terms of accuracy, defined as the fraction of correctly answered queries.

[Models] We compare the results of the following settings:

- **Sempre** (Pasupat and Liang, 2015) is a state-of-the-art semantic parser and serves as the baseline;
- **N2N**, NEURAL ENQUIRER model trained using *end-to-end* setting (Sec 4.3);
- **SbS**, NEURAL ENQUIRER model trained using *step-by-step* setting (Sec 4.4).

4.3 End-to-End Evaluation

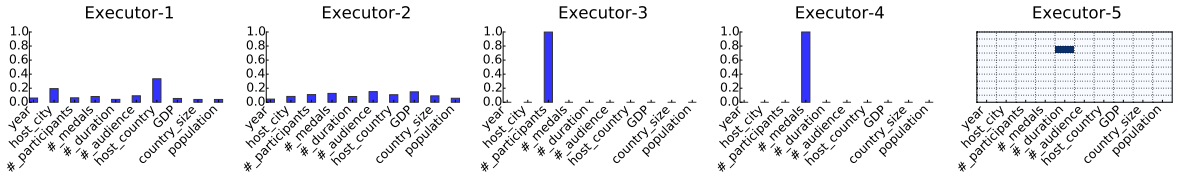
Table 2 summarizes the results of SEMPRE and NEURAL ENQUIRER under different settings. We show both the individual performance for each query type and the overall accuracy. We evaluate SEMPRE only on MIXED-25K because of its long training time even on this small dataset (about 3 days).

In this section we discuss the results under end-to-end (N2N) training setting. On MIXED-25K, the relatively low performance of SEMPRE indicates that our QA task, although synthetic, is highly nontrivial. Surprisingly, our model outperforms SEMPRE on all types of queries, with a marginal gain on *simple* queries (SELECT_WHERE, SUPERLATIVE), and significant improvement on *complex* queries (WHERE_SUPERLATIVE, NEST). On MIXED-100K, our model achieves a decent overall accuracy of 90.6%. These results show that in our QA task, NEURAL ENQUIRER is very effective in answering compositional NL queries, especially those with complex semantic structures compared with the state-of-the-art system.

To further understand why our model is capable of answering compositional queries, we study the attention weights $\tilde{w}(\cdot)$ of Readers (Eq. 1) for executors in intermediate layers, and the answer probability (Eq. 2) the last executor outputs for each entry in the table. Those statistics are obtained on MIXED-100K. We sample two queries (Q_1 and Q_2) in the testing set that our model answers correctly and visualize their corresponding values in Figure 5. To

Q_1: *How long was the game with the most medals that had fewer than 3,000 participants?*

Z_1: `where #_participants < 3,000, argmax(#_duration, #_medals)`



Q_2: *Which country hosted the longest game before the game in Athens?*

Z_2: `where year < (select year,where host_city=Athens), argmax(host_country, #_duration)`

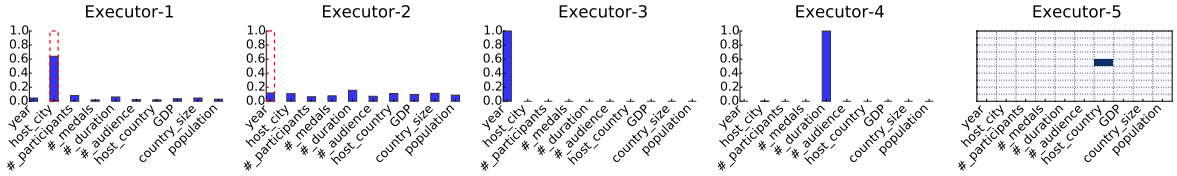


Figure 5: Weights visualization of queries Q_1 and Q_2

better understand the query execution process, we also give the logical forms (Z_1 and Z_2) of the two queries. Note that the logical forms are just for reference purpose and unknown by the model. We find that each executor actually *learns* its execution logic from just the correct answers in N2N training, which is in accordance with our assumption. The model executes Q_1 in three steps, with each of the last three executors performs a specific type of operation. For each row, Executor-3 takes the value of the `#_participants` field as input, while Executor-4 attends to the `#_medals` field. Finally, Executor-5 outputs a high probability for the `#_duration` field in the 3-rd row. The attention weights for Executor-1 and Executor-2 appear to be meaningless because Q_1 requires only three steps of execution, and the model learns to defer the meaningful execution to the last three executors. Compared with the logical form Z_1 of Q_1, we can deduce that Executor-3 "executes" the `where` clause in Z_1 to find row sets R satisfying the condition, and Executor-4 performs the first part of `argmax` to find the row $r \in R$ with the maximum value of `#_medals`, while Executor-5 outputs the value of `#_duration` in row r.

Compared with Q_1, Q_2 is more complicated. According to Z_2, Q_2 involves an additional nest subquery to be solved by two extra executors, and requires a total of five steps of execution. The last three executors function similarly as in answering Q_1, yet the execution logic for the first two executors (devoted to solving the sub-query) is a bit obscure, since their attention weights are scattered instead of being perfectly centered on the ideal fields as highlighted in red dashed rectangles. We posit that this is because during the end-to-end training, the supervision signal propagated from the top layer has decayed along the long path down to the first two executors, which causes vanishing gradients.

4.4 With Additional Step-by-Step Supervision

To alleviate the vanishing gradient problem when training on complex queries like Q_2, we train the model using step-by-step (SbS) setting (Eq. 4), where we encourage each intermediate executor to attend to the field that is known *a priori* to be relevant to its execution logic. Results are shown in Table 2 (column SbS). With stronger supervision signal, the model significantly outperforms the N2N setting, and achieves perfect accuracy on MIXED-100K. This shows that NEURAL ENQUIRER is capable of leveraging the additional supervision signal given to intermediate layers in SbS training. Let us revisit the query Q_2 in SbS setting. In contrast to the result in N2N setting (Figure 5) where the attention weights for the first two executors are obscure, now the weights are perfectly skewed towards each relevant field with a value of 1.0, which corresponds with the highlighted ideal weights.

5 Conclusion

We propose NEURAL ENQUIRER, a fully neural, end-to-end differentiable network that learns to execute compositional natural language queries on knowledge-base tables.

References

[Bahdanau et al.2015] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *ICLR*.

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

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

[Graves et al.2014] Alex Graves, Greg Wayne, and Ivo Danihelka. 2014. Neural turing machines. *CoRR*, abs/1410.5401.

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

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

[Pasupat and Liang2015] Panupong Pasupat and Percy Liang. 2015. Compositional semantic parsing on semi-structured tables. In *ACL (1)*, pages 1470–1480.

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

[Weston et al.2015] Jason Weston, Antoine Bordes, Sumit Chopra, and Tomas Mikolov. 2015. Towards ai-complete question answering: A set of prerequisite toy tasks. *CoRR*, abs/1502.05698.

[Zaremba and Sutskever2014] Wojciech Zaremba and Ilya Sutskever. 2014. Learning to execute. *CoRR*, abs/1410.4615.

[Zeiler2012] Matthew D. Zeiler. 2012. ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701.