A cross-domain natural language interface to databases using adversarial text method

Wenlu Wang[†]
supervised by Wei-Shinn Ku[†] and Haixun Wang[‡]
Auburn University[†] and WeWork Research[‡]
wenluwang@auburn.edu

ABSTRACT

A natural language interface (NLI) to databases is an interface that supports natural language queries to be executed by database management systems (DBMS). However, most NLIs are domain specific due to the complexity of the natural language questions, and an NLI trained on one domain is hard to be transferred another due to the discrepancies between different ontology. Inspired by the idea of stripping domain-specific information out of natural language questions, we propose a cross-domain NLI with a general purpose question tagging strategy and a multi-language neural translation model. Our question tagging strategy is able to extract the "skeleton" of the question that represents its semantic structure for any domain. With question tagging, every domain will be handled equally with a single multi-language neural translation model. Our preliminary experiments show that our multi-domain model has excellent cross-domain transferability.

1. INTRODUCTION

Relational databases are widely adopted in real-world applications [15, 14]. However, it requires a certain knowledge of query languages to operate on DBMSs, which motivated the study of NLI to databases [1] with the purpose of making DBMSs operable by anyone without training.

The challenges of NLI to databases lies in the discrepancies of different ontology, which makes general-purpose NLI hard to achieve. Most existing general purpose NLIs exploit syntax-guided decoding and require the grammar of the structured queries (domain specific grammar) as part of the model. Such a model cannot be shared between different grammars, while we propose a general purpose model where different types of queries and difference domains share the same components. To overcome the obstacles of generalizing one NLI model to different domains or even unseen domains, we perform a pre-processing step inspired by the strategy of separating domain-specific information from the question [22]. By detaching domain-specific data elements, the NLI model is able to focus on the semantic meaning and agnostic of the natural language question, which facilitates the cross-domain generalization.

We first "strip" a natural language question (shown in Figure 1), each type of the query (SQL and Lambda expression in our examples) is treated equally, then translate the tagged question to a structured query. The definition of "strip" is enclosing a phrase that describes a data element (tables, columns, values, keywords, etc.) appearing in the query by inserting a symbol (k_1, v_1, etc.) representing the type and index of the data element in front of the phrase, and an "end of element" symbol (e.g., $\langle eoe \rangle$) at the end of the phrase. In Figure 1, we show two types of queries (Lambda Expression and SQL), k represents a column field, a table name, or a keyword, and v represents a value.

Question	Which cities are located in Virginia ?
Query	city(A), location(A, B), const(B, stateid("Virginia"))
Question	⟨**lambda**⟩ Which ⟨k_1⟩ cities ⟨*eoe*⟩ are ⟨k_2⟩ located in ⟨*eoe*⟩ ⟨v_2⟩ Virginia ⟨*eoe*⟩ ?
Query	k_1(A), k_2(A, B), const(B, stateid(v_2))
	(1)
Question	Which movies were scheduled to release on May 19 2019 ?
Query	SELECT movie WHERE release date = May 19 2019
Question	⟨**SQL**⟩ *Which* ⟨k_1⟩ *movies* ⟨*eoe*⟩ *were scheduled* ⟨k_2⟩ to release on ⟨*eoe*⟩ ⟨v_2⟩ May 19 2019 ⟨*eoe*⟩ ?
Query	SELECT k_1 WHERE k_2 = v_2
	(2)

Figure 1: Two types of queries (SQL and Lambda expression) with corresponding Natural language questions.

Another challenge of our cross-domain task is to handle different query types. The aforementioned symbol insertion strategy is able to handle questions of different types equally but fails to differentiate them. Inspired by Google's multilingual translation model [11] where an artificial token is introduced at the beginning of the input sentence to indicate the target language. We prefix a query type symbol to indicate the target query type the NLI model should covert to (e.g., ⟨*SQL*⟩, ⟨*lambda*⟩). For instance, consider the following question –> SQL pair:

Which is the highest score? –> SELECT MAX(score)

It will be modified to:

⟨SQL⟩ Which is the highest score? –> SELECT MAX(score)

Such an approach only needs to prefix one additional token, we will validate in the preliminary experiments that it is the simplest but effective approach.

The core design of our symbol insertion strategy lies in how to identify the phrase that describes a data element appears in the corresponding query. The phrase might not

Proceedings of the VLDB 2019 PhD Workshop, August 26th, 2019. Los Angeles, California.

be the exact words of the data element. In Figure 1(1), the data element "*release date*" is described as "*to release on*" in the question. Inspired by gradient-based adversarial text method, we propose an adversarial method towards a data element detector. Given a natural language question q and a data element e, the data element detector will predict whether e is mentioned in q.

Figure 1 presents two examples. In example (1), the question (imply natural language question in this paper) is converted to a lambda expression, and example (2) is converted to a SQL query.

2. ADVERSARIAL TEXT METHOD

It has been demonstrated that adding a carefully crafted small noise is able to fool the deep neural network models into wrong predictions, while the small noise makes unnoticeable visual difference [4]. Most of the adversarial attack methods on the text [12, 19, 10] try to perturb the features (e.g., words, characters, and phrases) that are the most influential on the predictions. Inspired by gradient-based adversarial text attacks [5], we propose our own solution to identify the position of a data element in a question.

3. DESIGN

3.1 Overview

Given a (question, query) pair, our core methodology is to insert pre-designed symbols and enclose data elements mentioned in the question to achieve the purpose of handling every sample (of different domain/type) equally. Figure 2 shows the framework of our approach bottom-up.

1. Build a binary classifier BC as a data element detector to predict whether a data element e appears in a question q's corresponding query p by the semantic meaning of the question, which takes q and e as inputs without referencing p.

2. Inspired by [5], we search for the most influential phrase in the question using gradient-based adversarial text methods. We refer "the most influential phrase" as the phrase that describes the data element e theoretically.

3. We insert symbols in q to enclose the phrases that describes the data elements, denoted as q'. Since a query type symbol (e.g., $\langle SQL \rangle$) is prefixed to q', a user is able to select a desired query type.

4. Build a multi-lingual cross-domain sequence-to-sequence (seq2seq) model to translate q' to p', and p' is a query where the data elements are replaced by symbols inserted in q.

5. Inserted symbols are replaced with data elements to form the original query (convert p' back to p).

Figure 2 shows two examples of (q, q', p', p) correspond to Figure 1. In Figure 1(1), data elements "*city*" and "*Virginia*" are able to be detected by comparing against the database using string match directly. So are "*movie*" and "*May 19 2019*" in Figure 1(2). However, detecting data elements "*location*" and "*release date*" is problematic, so we use the pre-trained binary classifier BC. If BC is well trained, it will produce positive predictions for data element $e = [$"*location*","*release date*"$]$. As the true label, p is not involved in the process. After translating q' to p', where data elements are represented as symbols, we perform the final step of converting p' back to p.

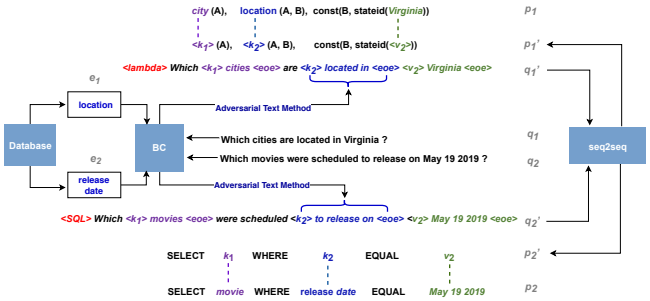

Figure 2: An example of cross-domain framework corresponds to Figure 1, different query types are treated equally.

3.2 Data Element Search

Data elements include table names, column fields, and column values in databases, and keywords in query grammar. For example, "*movies*", "*release date*", and "*May 19 2019*" are data elements in SQL query "`SELECT movie WHERE release date EQUAL May 19 2019`", and the other elements SELECT WHERE EQUAL belong to the template of SQL sketch. In "`city(A),location(A,B),const(B, stateid(Virginia))`", "*city*" is a table name, "*stateid*" is a column field, "*Virginia*" is a column value, and "*location*" is a keyword in lambda expression grammar. The challenge is to discover all the data elements from the question , and for symbol insertion purpose, we need to discover which phrase describes each data element. With such symbols insertion, we relieve the burden of identify the data elements from seq2seq model, and makes it focus on learning semantic structure of the question and logic of the data elements.

We have two challenges to tackle for data element search (use Figure 2 as an example):

1. Identify whether a data element is described in the question. Ultimately, we are trying to detect all the data elements that constitute the query, and we have to infer those data elements from the natural language question based on its semantics. In q_1, we need to identify all the data elements that are described in the question (e.g., "*city*", "*location*", and "*Virginia*"). In q_2, we need to identify "*release data*", "*movie*", and "*May 19 2019*".

2. The phrase that describes a data element needs to be identified by its semantic meaning and contextual comprehension. In q_1 of Figure 2, "*located in*" is identified as the most influential phrase while describing the key word "*location*". In q_2, "*to release on*" is identified as the phrase that describes "*release date*".

To address these two challenges, we propose our general purpose data element search strategy with two steps:

- We propose a **Data Element Detector** (Sec 3.2.1) for the first challenge, which is a binary classifier with

a question q and data element e as an input. The detector is able to detect whether the data element e is mentioned in question q. As presented in Figure 2, a Data Element Detector is shared among all the domains.

- In the case of positive prediction in the previous step, we propose an **Adversarial Text Method** (Sec 3.2.2) for the second challenge, which relies on the information learned by the binary classifier from the first step.

3.2.1 Data Element Detector

We use a bi-directional attentive recurrent neural network to achieve question understanding. For a question q composed of n tokens $[q_1, ..., q_n]$ and a data element e composed of m tokens $[e_1, ..., e_m]$, we use a pre-trained Glove embedding to initiate a word embedding layer. On top of the embedding layer, we use LSTM cells to produce hidden states for each time step (each word in q). We build a similar structure for e. We denote the top layer hidden states as

$$h^q = [h_1^q, \cdots, h_n^q] \qquad h^e = [h_1^e, \cdots, h_m^e]$$

We build a bi-directional LSTM layer on h^q with attention over h^e.

$$\begin{aligned}
&\overrightarrow{d_0} = \mathbf{0} \\
&\overrightarrow{\gamma_t} = \begin{bmatrix} h_t^e \\ \overrightarrow{\beta_t} \end{bmatrix} \\
&\overrightarrow{d_t} = \text{LSTM}_{\rightarrow}\left(\overrightarrow{\gamma_t}, \overrightarrow{d_{t-1}}\right) \\
&\overrightarrow{e_{tj}} = v^T \text{Tanh}(W_0 h_j^q + W_1 h^e + W_2 \overrightarrow{d_{t-1}}) \\
&\overrightarrow{\alpha_{tj}} = \overrightarrow{e_{tj}} / \sum_{j'} \overrightarrow{e_{tj'}} \\
&\overrightarrow{\beta_t} = \sum_{j=1}^{n} \overrightarrow{\alpha_{tj}} h_j^q
\end{aligned}$$

where W_0, W_1, W_2 ,and v are model parameters. Here t enumerates each time step for e, and j enumerates each token in q. We compute bi-directional output $d_t = [\overrightarrow{d_t}, \overleftarrow{d_t}]$, and feed it to a multi-layer perception for binary prediction.

3.2.2 Adversarial Text Method

With the adversarial text method, given a data element e that has a positive prediction from the binary classifier, we propose to search a phrase of the question that describes e. We describe our adversarial text method as follows.

1. We have trained a Data Element Detector that takes a question q and a data element e as inputs and predicts whether e is described in q.

2. We search for the most influential phrase in q using gradient-based adversarial text methods [5]. There are three possible directions (the loss gradient of the Data Element Detector with q and e as inputs is $\nabla L(q, e)$):

   - DeepFool [13]. We iteratively search the optimal direction in which only a minimum perturbation is needed to affect the prediction. Theoretically, the optimal direction is $-\frac{f(q,e)}{||\nabla L(q,e)||_2^2} \nabla L(q, e)$ where $f(\cdot)$ denotes the Data Element Detector.

   - Fast Gradient Method FGM [6]. We add a noise that is proportional to either $\nabla L(q, e)$ or $sign(\nabla L(q, e))$ to the original sample to change the prediction of the Data Element Detector. In particular, the noise for each token q_i is proportional to $\frac{\partial L(q,e)}{\partial q_i}$.

   - JSMA [16]. We calculate the Jacobian-based saliency map based on $\nabla L(q, e)$, and perturb one token at a time. The chosen token has the highest saliency value.

   Since all the methods are trying to add minimum noise that influences the prediction the most. The locations where the noise is added will be the positions of tokens that constitute the most influential phrase – i,e., the phrase that describes the data element e.

3. We search for a phrase in the question where adding a small perturbation will affect the prediction dramatically.

The challenge of our adversarial text method is the discreteness of the text domain. Words or characters are discrete variables thus indifferentiable. To overcome such problem, we propose to calculate the loss gradient (∇L) of the target model w.r.t. the embedding of each word, where the embedding space is smooth.

3.3 Neural Machine Translation

We denote an question post symbol insertion as q' and the corresponding query post symbol replacement as p'. We train a seq2seq model to translate q' to p':

$$p' = \text{SEQ2SEQ}(q')$$

Encoder is a stacked bi-directional multi-layer recurrent neural network (RNN). Decoder is a one-layer *attentive* RNN.

We use a single multilingual neural translation model for our cross-domain NLI task. We believe with a prefixed query type symbol (e.g., $\langle SQL \rangle$), a multi-domain model is able to handle different query types, and each query type is treated equally.

4. RELATED WORK

NLI to databases was first formally introduced in [1]. Semantic parsing [17, 23, 9] and cross-domain semantic parsing [7, 20] are applied in NLI to databases. However, due to the incompatibility among different domains, cross-domain task remains unsolved. The sketch-based solutions are also extensively studied, which is first proposed in [24]. A deep model is trained to fill the slots in the sketch. An extension of sketch-based solution [26] relies on a knowledge base to identify the column values. Such a strategy is confined in pre-defined sketch and existing knowledge base. seq2seq model are also exploited to serve as a translator [8, 28], which has no limitations on query sketch.

5. PRELIMINARY EXPERIMENTS

Domain	Dataset	Method	Test Acc_{qm}	Test Acc_{ex}
Single	WikiSQL	Seq2SQL [28]	51.6%	60.4%
		SQLNet [24]	61.3%	68.0%
		TypeSQL [26]	75.4%	82.6%
Multi	WikiSQL	*Ours-multi*	74.5%	**82.7**%
	OVERNIGHT		76.8%	-
	Geo880		84.1%	-

Table 1: Comparison of models.

5.1 Evaluation

We conduct our preliminary experiments using a seq2seq model with stacked GRU. We use query-match accuracy Acc_{qm} for evaluation, we match synthesized queries against the ground truth p. We also compare the execution results as [28], denoted as Acc_{ex}, if applicable.

We jointly train our multi-domain model on WikiSQL [28], Geo880 [27] , and OVERNIGHT [23], their query types are SQL, Lambda expression, and SQL (we use the dataset that manually converted to SQL in [22]). Some of the domains are over sampled to balance the number of training samples among all the domains. Our method is shown in Table 1 as *Ours-multi*. Since all the domains are trained with in a single model, the accuracy of multi-domain model does not exhibit a large improvement. However, we believe it is a model capacity issue since the accuries of all the domains are very close or better than state-of-the-art performance. We observe that the seq2seq model is able to infer both SQL and Lambda expression as long as a tag (e.g., $\langle SQL \rangle, \langle lambda \rangle$) indicating the query type is provided.

5.2 Spatial Domain

Domain	Geo880		Restaurant	
	Method	Acc_{dm}	Method	Acc_{dm}
Single	SEQ2TREE [2]	87.1%	PEK03 [18]	97.0%
	TRANX [25]	88.2%	TM00 [21]	99.6%
	JL16 [9]	89.3%	FKZ18 [3]	100%
Ours-multi		**90.7**%		**100**%

Table 2: Comparison of models in Spatial Domain.

We conduct preliminary evaluations on the spatial domain (Geo880 [27] and Restaurant [21]), which is a more difficult task since the dataset in the spatial domain is sparse. We adopt the data element detector as a spatial comprehension model, and inject spatial semantics using symbol insertions. In this setting, we jointly train a multi-domain model with both Geo880 and Restaurant training sets (Restaurant is oversampled to balance all the domains), and evaluate on the test set of each separately (since both are for lambda expression queries, a prefix symbol is not inserted). As shown in Table 2 (we use denotation match accuracy Acc_{dm} for evaluation), our method (*Ours-multi*) outperforms previous methods.

6. REFERENCES

[1] I. Androutsopoulos, G. D. Ritchie, and P. Thanisch. Natural language interfaces to databases–an introduction. *Natural language engineering*, 1(01):29–81, 1995.

[2] L. Dong and M. Lapata. Language to logical form with neural attention. *arXiv preprint arXiv:1601.01280*, 2016.

[3] C. Finegan-Dollak, J. K. Kummerfeld, L. Zhang, K. Ramanathan, S. Sadasivam, R. Zhang, and D. Radev. Improving text-to-sql evaluation methodology. *arXiv preprint arXiv:1806.09029*, 2018.

[4] Z. Gong, W. Wang, and W.-S. Ku. Adversarial and clean data are not twins. *arXiv preprint arXiv:1704.04960*, 2017.

[5] Z. Gong, W. Wang, B. Li, D. Song, and W.-S. Ku. Adversarial texts with gradient methods. *arXiv preprint arXiv:1801.07175*, 2018.

[6] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[7] J. Herzig and J. Berant. Neural semantic parsing over multiple knowledge-bases. *arXiv preprint arXiv:1702.01569*, 2017.

[8] S. Iyer, I. Konstas, A. Cheung, J. Krishnamurthy, and L. Zettlemoyer. Learning a neural semantic parser from user feedback. In *ACL*, volume 1, pages 963–973, 2017.

[9] R. Jia and P. Liang. Data recombination for neural semantic parsing. 2016.

[10] R. Jia and P. Liang. Adversarial examples for evaluating reading comprehension systems. *arXiv preprint arXiv:1707.07328*, 2017.

[11] M. Johnson, M. Schuster, Q. V. Le, M. Krikun, Y. Wu, Z. Chen, N. Thorat, F. Viégas, M. Wattenberg, G. Corrado, et al. Googles multilingual neural machine translation system: Enabling zero-shot translation. *ACL*, 5:339–351, 2017.

[12] B. Liang, H. Li, M. Su, P. Bian, X. Li, and W. Shi. Deep text classification can be fooled. *arXiv preprint arXiv:1704.08006*, 2017.

[13] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *CVPR*, pages 2574–2582, 2016.

[14] E. Ngai, Y. Hu, Y. Wong, Y. Chen, and X. Sun. The application of data mining techniques in financial fraud detection: A classification framework and an academic review of literature. *Decision Support Systems*, 50(3):559–569, 2011.

[15] E. W. Ngai, L. Xiu, and D. C. Chau. Application of data mining techniques in customer relationship management: A literature review and classification. *Expert systems with applications*, 36(2):2592–2602, 2009.

[16] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami. The limitations of deep learning in adversarial settings. In *2016 IEEE EuroS&P*, pages 372–387. IEEE, 2016.

[17] P. Pasupat and P. Liang. Compositional semantic parsing on semi-structured tables. In *ACL*, 2015.

[18] A. Popescu, O. Etzioni, and H. A. Kautz. Towards a theory of natural language interfaces to databases. In *Proceedings of the 8th International Conference on Intelligent User Interfaces*, 2003.

[19] S. Samanta and S. Mehta. Towards crafting text adversarial samples. *arXiv preprint arXiv:1707.02812*, 2017.

[20] Y. Su and X. Yan. Cross-domain semantic parsing via paraphrasing. *arXiv preprint arXiv:1704.05974*, 2017.

[21] L. R. Tang and R. J. Mooney. Automated construction of database interfaces: Integrating statistical and relational learning for semantic parsing. In *ACL*, pages 133–141, 2000.

[22] W. Wang, Y. Tian, H. Xiong, H. Wang, and W.-S. Ku. A transfer-learnable natural language interface for databases. *arXiv preprint arXiv:1809.02649*, 2018.

[23] Y. Wang, J. Berant, and P. Liang. Building a semantic parser overnight. In *ACL*, volume 1, pages 1332–1342, 2015.

[24] X. Xu, C. Liu, and D. Song. Sqlnet: Generating structured queries from natural language without reinforcement learning. *arXiv preprint arXiv:1711.04436*, 2017.

[25] P. Yin and G. Neubig. Tranx: A transition-based neural abstract syntax parser for semantic parsing and code generation. *arXiv preprint arXiv:1810.02720*, 2018.

[26] T. Yu, Z. Li, Z. Zhang, R. Zhang, and D. Radev. Typesql: Knowledge-based type-aware neural text-to-sql generation. *arXiv preprint arXiv:1804.09769*, 2018.

[27] J. M. Zelle and R. J. Mooney. Learning to parse database queries using inductive logic programming. In *AAAI*, pages 1050–1055, 1996.

[28] V. Zhong, C. Xiong, and R. Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*, 2017.