Embedding Individual Table Columns for Resilient SQL Chatbots

Bojan Petrovski†1, **Ignacio Aguado**‡1, **Andreea Hossmann**‡, **Michael Baeriswyl**‡, **Claudiu Musat**‡
† School of Computer and Communication Sciences, EPFL, Switzerland
‡ Artificial Intelligence Group - Swisscom AG
firstname.lastname@{epfl.ch, swisscom.com}

Abstract

Most of the world's data is stored in relational databases. Accessing these requires specialized knowledge of the Structured Query Language (SQL), putting them out of the reach of many people. A recent research thread in Natural Language Processing (NLP) aims to alleviate this problem by automatically translating natural language questions into SQL queries. While the proposed solutions are a great start, they lack robustness and do not easily generalize: the methods require high quality descriptions of the database table columns, and the most widely used training dataset, WikiSQL, is heavily biased towards using those descriptions as part of the questions.

In this work, we propose solutions to both problems: we entirely eliminate the need for column descriptions, by relying solely on their contents, and we augment the WikiSQL dataset by paraphrasing column names to reduce bias. We show that the accuracy of existing methods drops when trained on our augmented, column-agnostic dataset, and that our own method reaches state of the art accuracy, while relying on column contents only.

1 Introduction

Recent developments in Natural Language Understanding (NLU) have led to a big proliferation of text- and speech-based bot interfaces. Home appliances, such as smart speakers and chatbots, rely mostly on a well-structured knowledge base or an external Application Programming Interface (API) to provide the desired response. This limits the usability of such systems in a context where the data is stored in a (local) relational database.

This constraint led to the development of text to Structured Query Language (SQL) systems, also known as SQL bots. Given a question, in natural language, pertaining to a certain database table, these bots will automatically generate the corresponding SQL query and return the requested data. Considering the vast usage of relational databases on the internet and in private companies, SQL bots are a simple new interface that enables non-technical people to access data.

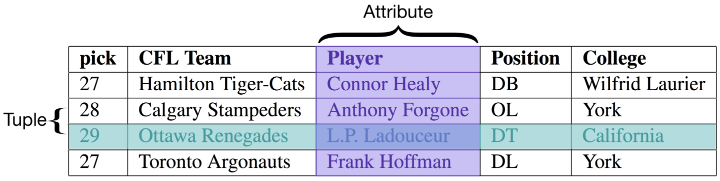
The first approaches in the field relied on parsers and pattern-matching rules to understand the question and produce appropriate answers (Androutsopoulos et al., 1995). Later developments introduced semantic grammar systems and intermediate language systems (Androutsopoulos et al., 1995). More recently, new NLU methods, such as pointer-networks, pushed the state-of-the-art results in several domains, including parsing (Vinyals et al., 2015). Current state-of-the-art models are based on sketches and have primarily two inputs: the question and the descriptions of the table columns (i.e., the column names).

Relying on the column names is limiting, since the whole model is based on several strong premises: (a) the names are high quality and descriptive enough; (b) the names do not change; (c) the names are known to the user of the bot. These are very strong assumptions: often, column names do not even exist (i.e., the generic col1, col2, etc. are used instead). Moreover, if as we observe in Figure 1, a column contains the names of colleges, just changing the column name form "College" to "School" does not make the content any less informative. The expectation from a bot is that their quality is not sensitive to cosmetic changes to the underlying table. Finally, users do not necessarily know the structure of the table, let alone the column names.

In this paper, we build and present ICE (Individual Column Embeddings) – a novel approach of representing the database table columns, by using their contents instead of their names. To do so,

[1] equal contribution

Proceedings of the 2018 EMNLP Workshop SCAI: The 2nd Int'l Workshop on Search-Oriented Conversational AI, pages 67–73
Brussels, Belgium, October 31, 2018. ©2018 Association for Computational Linguistics
ISBN 978-1-948087-75-9


Attribute
Tuple

pick	CFL Team	Player	Position	College
27	Hamilton Tiger-Cats	Connor Healy	DB	Wilfrid Laurier
28	Calgary Stampeders	Anthony Forgone	OL	York
29	Ottawa Renegades	L.P. Ladouceur	DT	California
27	Toronto Argonauts	Frank Hoffman	DL	York

Figure 1: Part of a table from the WikiSQL dataset with the contexts within a relation (table) we can model

we construct a column embedding vector space, where we embed the columns. This embedding is then used as a substitute for the encoding of the column descriptions (headers) in a state of the art sketch-based model.

In addition, to empirically show the value of using ICE, we generate a new, column-agnostic dataset based on the widely used WikiSQL dataset (Zhong et al., 2017). In WikiSQL, a substantial bias towards the inclusion in the question of the column name is built-in. For instance more than 79% of questions contain the name of the column that needs to be selected. Additionally around 59% contain the names of all columns form the SQL `where` clause. With ICE, we are eliminating the strong assumption that the users have access to the table structure. Hence, we also need a less biased dataset to show the value of our method.

We thus create an open source data augmentation tool to paraphrase part of the questions in WikiSQL: where the column names are present, we replace them with similar expressions (e.g., synonyms), removing some of the built-in bias.

We train and test our ICE-based model on both the original WikiSQL dataset and our column-agnostic version of the dataset. We show that we maintain the same accuracy on both datasets with all three tasks: aggregation, column-selection and *where* clause generation. We also train the original SQLNet (Zhong et al., 2017) model on the column-agnostic dataset and find a 7% accuracy drop in the *where* clause generation task.

In a nutshell, the most important contribution of this work is that we **improve the model resilience** by limiting its reliance on arbitrary descriptions of the data within the tables. In addition, we **expand the applicability of SQL bots** to users who do not know the internal structure of the databases they are trying to access. By eliminating the need to encode the column headers, we also **reduce the overall complexity of the model**. This is achieved by removing the LSTM networks used to generate unique column header encodings for the aggregation prediction, selection prediction and *where* clause generation.

The paper is organized as follows: Section 2 presents the related work for translating sentences to SQL and for vector space embeddings. In Section 3, we describe ICE – our method for column content embeddings. In the next section, we introduce our column-agnostic model for translating sentences to SQL. We present the evaluation results in Section 5 and finally conclude in Section 6.

2 Related Work

2.1 Related work in Sentences to SQL

Systems that enable users to use natural language to interact with a database have been researched since the early seventies. As summarized in (Androutsopoulos et al., 1995) these early approaches were mostly rule-based. More successful methods have emerged since the advent of the sequence to sequence (Sutskever et al., 2014) neural network architectures and increased availability of training data in recent years. The first model to leverage this was SEQ2SQL introduced by (Zhong et al., 2017) together with their crowdsourced dataset WikiSQL. SEQ2SQL solves the problem of generating SQL queries in a three-step approach that aligns with the structure of an SQL query. First, it determines the aggregation function for the query i.e. whether to apply count, average, max etc. This is performed by a classifier trained on the encoding of the question and the encodings of the table headers. In the second step, the model determines the column on which to perform the selection, again based on the encoding of the question and the encodings of the table headers. Finally, in the last step, the model generates the where clause of the SQL query. To do so it first determines the number of conditions in the clause and then proceeds to generate tuples of a column, comparison operator and value using a pointer network. Since the order in the where clause is not important when there are multiple conditions the model also implements a reinforcement learning policy to optimize for execution correctness and uses a mixed loss function.

SQLNet (Xu et al., 2017) improved upon SEQ2SQL by eliminating the need for reinforcement learning by using a sketch-based approach. (Bornholt et al., 2016; Solar-Lezama et al., 2006) In the where clause section SQLNet introduces a sequence-to-set model. It first picks a set of columns which will be used in the clause. Subsequently, for each column, it determines a comparison operator using a classifier and picks a comparison value using a printer network. Additionally, this model implements a column attention mechanism which together with sequence-to-set model improves the accuracy over SEQ2SQL by 9% to 13%.

2.2 From Word to Table Embeddings

The most basic form of word embeddings is the bag of words model. It can be augmented by statistics such as TF-IDF, however, such vector space captures very little of the words semantics, morphology, hierarchy and context. Word2vec, introduced by (Mikolov et al., 2013) is one of the first popular neural embedding models. It comes in two general implementations: a continuous bag of words (order in window irrelevant) and a continuous skip gram (weight in window based on distance from current word). The objective function of Word2vec causes words that appear in a similar context to cluster together in the vector space, based on cosine distance. This method was modified by the introduction of global word representation which aims to capture the meaning of the word within the whole corpus (Pennington et al., 2014) and the use of subword information to capture the morphology of the words (Joulin et al., 2016).

With the addition of simple techniques, such as a trained weighted average, word-embedding algorithms were further extended to embed whole sentences (Pagliardini et al., 2018) and whole documents (Le and Mikolov, 2014). Such techniques have also recently been used to get the embedding of whole tables for the purposes of table classification (Ghasemi-Gol and Szekely, 2018).

3 ICE: Individual Column Embeddings

To understand the context and the hierarchy of a table we will use the formal definition of a relation: "a set of tuples (d_1, d_2, ..., d_n), where each element d_j is a member of D_j, the $j-th$ data domain." Tuples, relations and attributes are graphically depicted in Figure 1. We observe that there are two contexts in which an element, or cell, d_j appers either within a tuple (row) or within a data domain (column).

To embed the whole table we need to look at both contexts. This complexity is not necessary in the context of individual column embeddings, where the latter context is sufficient. TabVec uses deviation from the median for table vectors to capture the noise (Ghasemi-Gol and Szekely, 2018), as the final table vector incorporates information from cells that are not conceptually similar. This is not the case for individual column embeddings, as for ICE we assume that the cells within a column are conceptually similar. For instance, if the column is about locations, all the cells are likely to represent location names. This property allows us to simplify the aggregation and use the median vector of all cells as the column representation.

Table cells are not semantic atoms and can contain multiple words, for example in Figure 1 all *Team* names contain at least two words. Thus, given a vector space model for words, we compute the individual cell embedding (ICE) as the average of the word embeddings and the individual column embedding as the median of its cells.

To sum up, let a column D contain cells $c_i \in C(D)$, with each cell consisting of a sequence of n_i words (w_{i1}, ..., w_{ij}, ..., w_{in_i}). Given a function E that computes a word embedding, the ICE of the D is defined as:

$$E(D) = median_{c_i}(\frac{1}{n_i}\sum_{j=0}^{n_i} E(w_{ij})), c_i \in C(D)$$

3.1 Table Word Embeddings.

For the ICE to be meaningful, the word embeddings need to reflect the table semantics. The way words are used in tables differs significantly from the way they appear in normal language. We keep the intuition that a word can be represented as an aggretation over all the contexts in which that word appears. What changes from typical text embeddings (Mikolov et al., 2013; Pennington et al., 2014) is that the context is given by other words that occur in the same table column. We view column tables as synthetic sentences that allow us to learn what the relevant context is. We then use

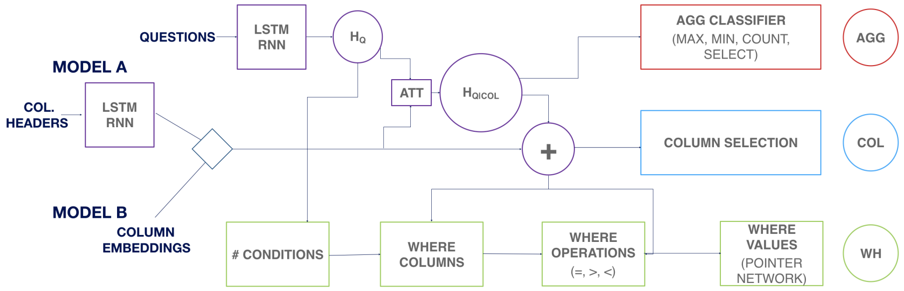


Figure 2: The general network architecture of SQLNet. Model A represents the original model, while Model B represents our model.

SkipGrams with a window of 5 to generate the embedding model.

We first construct a data corpus of synthetic sentences, corresponding to columns. We define a sentence as all the cells in one table column concatenated. Furthermore, we make the assumption that the order of the cells within a column is not important. For the table in Figure 1, a sample sentence would be *Calgary Stampeders Ottawa Renegades Toronto Argonauts Hamilton Tiger-Cats*. We generate 10 random cell shuffles of each column. Using this corpus we train a word2vec model with the Gensim toolkit (Řehůřek and Sojka, 2010).

4 Individual Column Embedding for Bot Resilience

Our work builds upon the SQLNet (Xu et al., 2017) sketch-based approach. To generate a SQL statement, each component of the query is generated individually: *the aggregation*, the *column selection* and the *where clauses*. The task is thus akin to slot filling (Xu et al., 2017). The process is graphically depicted in Figure 2. The input or the SQLNet and previous models (Xu et al., 2017) consists of a representation of the question and a representation of each table column header.

We believe this assumption represents one of the most important drawbacks of the approach, as knowledge about the column headers may not exist in real world conditions. The reason this knowledge was used in previous work is that the dataset itself was biased towards explicitly including the column names in the question formulation. In this section we show how to build a dataset that alleviates this bias. We then use the new dataset to create a model that relies on the column content , not on the column headers.

Column type	Train	Test	Dev
Selection col.	79.0%	79.0%	79.65%
Where col. >= 1	68.0%	67.6%	68.4%
All where col.	58.9%	58.5%	59.2%

Table 1: The percentages in the table show the proportion of questions that contain the specific column header in the different data partitions.

4.1 Column-agnostic WikiSQL

The wikiSQL dataset was crowdsourced using tables from Wikipedia. Workers on Amazon Mechanical Turk[1] were presented with a table and a generated SQL query and were asked to ask a question that matched the query. This method introduces an inherent bias in the dataset as demonstrated in Table 1. Almost 80% of questions contain the column name that is retrieved in the selection step and 68% of questions contain at least one of the column names from the where clause. In total, only 11% of the questions do not contain **exact matches** of the column names, as shown in Figure 1. As the workers were shown the whole table with column names, in a large number of cases they copied the column name in the question.

We paraphrase questions that contain a column name to make the dataset more realistic, as described in Algorithm 1. We create candidate questions by replacing the names with synonyms that share the syntactic and semantic properties of the original names.

The original question and the candidate questions are then embedded in vector space with sent2vec (Pagliardini et al., 2018). Using these vector space representations we compute the cosine similarity between the original question and

[1] https://www.mturk.com/

the potential replacements and choose the most similar candidate. This procedure yields a suitable rephrasing for 20% of the dataset, as we did not find synonyms for all questions containing column names. For instance, the orginal questions *What is the **length (miles)** of endpoints westlake/macarthur park to wilshire/western?*, which contains the column header **length (miles)**, becomes *What is the **distance (miles)** of endpoints westlake/macarthur park to wilshire/western?*.

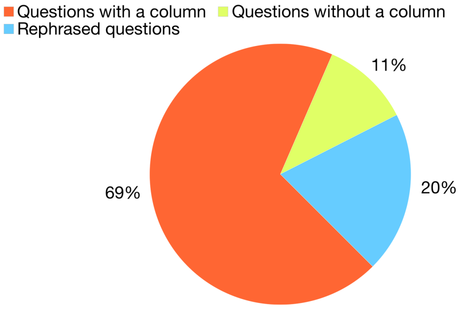


Figure 3: Proportions of the modified dataset

```
Data: Question and column header
Result: Replacement candidate questions
Tokenize and pos tag question;
for word in column header do
    Get word tag in question;
    Get word synonyms using tag;
    if synonyms list > 0 then
        append synonyms to rephrase list;
    end
end
for phrase in rephrase list do
    if length of phrase == length of header
      then
        replace column header in question
          with phrase;
        append new question to candidate list;
    end
end
```

Algorithm 1: Generating replacement candidate questions

4.2 Integrating Individual Column Embeddings

We compute the embeddings for the entire table column corpus as described in chapter 3. This is necessary since the embeddings are required during inference both during training and testing. Due to model size constraints, we keep the individual column embeddings constant during both training and testing. We create a dictionary to link each column to its embedding vector and feed it to the model (Model B) in Figure 2. An attention mechanism has the embeddings as inputs and the result contributes to the aggregation, selection and where clause modules. The column vectors are generated with the same dimensions that we use for the question encoding.

As we replace the column headers with column content embeddings, our model is completely agnostic to the headers. We thus remove the LSTM used to encode the column headers in the three model components: aggregation, selection and where clause generation. This leads to a significant **reduction in the complexity of the model**.

5 Evaluation

5.1 Original WikiSQL Evaluation

The evaluation on the full original WikiSQL dataset determines whether the individual column embeddings are suitable replacements for headers when the column name appears in the question. Table 2 summarizes the results of our model *SQLNet+ICE* and compares them with the results of two baselines: *SQLNet* and *Seq2SQL*. We portray the accuracy values on the development and test sets for the three slots we fill in the sketch: *Aggregation function*, *Column Selection* and *Where clause generation*.

We observe that *SQLNet+ICE* performs similarly to the original *SQLNet* model in both cases and superior to *Seq2SQL*. This result shows that we can build an equally performing model that is resilient to changes to the DB schema or complete absence of knowledge about it.

We note that the accuracy of the aggregation function also changes. This happens because the aggregation classifier has either the column or header embeddings as inputs, as shown in 2. There is a small decrease of accuracy for the Aggregation and Where clauses, while the accuracy on the Column Selection performs slightly better. These results are expected, as the queries strongly rely on the direct column names mentions.

5.2 Column-agnostic WikiSQL Evaluation

The second experiment shows the more realistic results, obtained on the column-agnostic WikiSQL Dataset. The results in Table 3 show that SQLNet struggles to predict correctly the column related

	Dev Set Accuracy			Test Set Accuracy		
	Aggregation	Selection	Where-clause	Aggregation	Selection	Where-clause
Seq2SQL	90.0%	89.6%	62.1%	90.1%	88.9%	60.2%
SQLNet	90.1%	91.5%	74.1%	90.3%	90.9%	71.9%
SQLNet + ICE	89.7 %	92.4	72.2%	89.3 %	91.8	71.1%

Table 2: Model accuracies on the Original WikiSQL Dataset

	Dev Set Accuracy			Test Set Accuracy		
	Aggregation	Selection	Where-clause	Aggregation	Selection	Where-clause
SQLNet	90.1%	87.5%	63.4%	90.3%	87.1%	63.1%
SQLNet + ICE	89.7 %	88.4	70.1%	89.3 %	87.9	69.4%

Table 3: Model accuracies on the Column-agnostic WikiSQL Dataset

	Rephrased Test Set Accuracy		
	Agg.	Sel.	W.-clause
SQLNet	89.5 %	81.3%	43.2%
SQLNet + ICE	88.9 %	83.2	61.3%

Table 4: Model accuracies on the paraphrased questions only on Aggreation, Selection and Where-clause tasks.

parts of the query, especially in the case of the where clause generation. This drop in the accuracy is expected, since the where clause predictor is the most complex part of the model. Without the original dataset bias where the column names were present in the questions, the column names are not descriptive enough.This leads to a drop of 10.7% on the validation and 8.8% on test dataset.

On the other hand, our model is capable of overcoming this situation and find the queries with a much smaller drop of accuracy. Although the performance is also worse than with the original dataset, the accuracy obtained using SQLNet with individual column embeddings in the where clauses is only 2.1% lower in validation and 1.7% in test. Using individual column embeddings makes the SQLNet model more versatile, as it can address the scenario where the user is not aware of the table structure.

Focusing on rephrased questions. To better understand our results on the Column-agnostic WikiSQL dataset we run the evaluation just with questions that have been rephrased, which represent around 20% of the whole data set, as shown in Figure 3. Table 4 summarizes these results, with SQLNet is the original model described in (Xu et al., 2017). The previously seen drop in *SQLNet* accuracy on the column selection and where-clause predictions is exacerbated - showing that indeed the paraphrasing is indeed the root cause. This effect is comparatively mild in *SQLNet + ICE*.

6 Conclusion and Future Work

In this paper, we proposed a new approach to build SQL chatbots without relying on the database table schema. Previous work built around the WikiSQL dataset take advantage of the dataset biases and use the column names to improve performance. This reliance on the schema inhibits their generalization capacity to cases where schema knowledge is absent. Our model, built on SQLNet by adding Individual Column Embeddings *SQLNet + ICE*, does not suffer from these limitations.

We provide a way to create Individual Column Embeddings, different from the Column Embeddings in prior art (Ghasemi-Gol and Szekely, 2018). Furthermore, we publish a method to paraphrase WikiSQL questions to alleviate the dataset bias.

The results of our model on the paraphrased WikiSQL are very similar to the ones obtained on the original dataset, while the SQLNet models struggles to deal with the paraphrasing.

Future Work. Even with these changes, there is still room for improvement in the SQL chatbot area. Large scale operations need the support for multiple tables at the time as well as more operations such as *join*. While WikiSQL is a good starting point and our modified version removes some of the biases present in it, there is a strong need for more data, both in terms of quantity and diversity. This new data needs to include more operations, as well as new ways to collect questions to have more variety in the structure of the user's utterances.

References

Ion Androutsopoulos, Graeme D. Ritchie, and Peter Thanisch. 1995. Natural language interfaces to databases - an introduction. *CoRR*, cmp-lg/9503016.

James Bornholt, Emina Torlak, Dan Grossman, and Luis Ceze. 2016. Optimizing synthesis with metasketches. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '16, pages 775–788, New York, NY, USA. ACM.

Majid Ghasemi-Gol and Pedro A. Szekely. 2018. Tabvec: Table vectors for classification of web tables. *CoRR*, abs/1802.06290.

Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. 2016. Bag of tricks for efficient text classification. *CoRR*, abs/1607.01759.

Quoc V. Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. *CoRR*, abs/1405.4053.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc.

Matteo Pagliardini, Prakhar Gupta, and Martin Jaggi. 2018. Unsupervised Learning of Sentence Embeddings using Compositional n-Gram Features. In *NAACL 2018 - Conference of the North American Chapter of the Association for Computational Linguistics*.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *In EMNLP*.

Radim Řehůřek and Petr Sojka. 2010. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta. ELRA. http://is.muni.cz/publication/884893/en.

Armando Solar-Lezama, Liviu Tancau, Rastislav Bodik, Sanjit Seshia, and Vijay Saraswat. 2006. Combinatorial sketching for finite programs. *SIGPLAN Not.*, 41(11):404–415.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2692–2700. Curran Associates, Inc.

Xiaojun Xu, Chang Liu, and Dawn Song. 2017. Sqlnet: Generating structured queries from natural language without reinforcement learning. *CoRR*, abs/1711.04436.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *CoRR*, abs/1709.00103.