Encoding Database Schemas with Relation-Aware Self-Attention for Text-to-SQL Parsers

Anonymous Author(s)
Affiliation
Address
email

Abstract

When translating natural language questions into SQL queries to answer questions from a database, we would like our methods to generalize to domains and database schemas outside of the training set. To handle complex questions and database schemas with a neural encoder-decoder paradigm, it is critical to properly encode the schema as part of the input with the question. In this paper, we use relation-aware self-attention within the encoder so that it can reason about how the tables and columns in the provided schema relate to each other and use this information in interpreting the question. We achieve significant gains on the recently-released Spider dataset with 42.94% exact match accuracy, compared to the 18.96% reported in published work.

1 Introduction

The ability to effectively query databases with natural language has the potential to unlock the power of large datasets to the vast majority of users who are not proficient in the use of languages such as SQL. As such, a large body of existing work has focused on the task of translating natural language questions into queries that existing database software can execute.

The release of large annotated datasets containing questions and the corresponding database queries has catalyzed significant progress in the field, by enabling the training of supervised learning models for the task [24, 4]. This progress has arrived not only in the form of improved accuracy on the test sets provided with the datasets, but also through an evolution of the problem formulation towards greater complexity more closely resembling real-world applications.

The recently-released Spider dataset [22] exemplifies greater realism in the task specification: the queries are written using SQL syntax, the dataset contains a large number of domains and schemas with no overlap between the train and test sets, and each schema contains multiple tables with many complicated questions being expressed in the queries. Due to the extra difficulty caused by these factors, the best publicly-reported result on this dataset as of this writing achieves about 19% exact matching accuracy on the development set [14], which is significantly worse compared to $> 80\%$ exact matching accuracy reported for past datasets such as ATIS, GeoQuery, and WikiSQL [22, 1].

We posit that a central challenge of the multi-schema problem setting is generalization to new database schemas different from what was seen during training. when the model needs to generate queries for arbitrary new schemas, it needs to take the relevant schema as an input and process it together with the question in order to generate the correct query.

Previous methods on the WikiSQL dataset [24] have also contended with the challenge of generalizing to arbitrary new schemas. However, all schemas in this dataset are quite simple, as they only contain one table. The model has no need to reason about the relationships between multiple tables in order to generate the correct query. As such, models developed for this dataset have largely focused on

Under review.

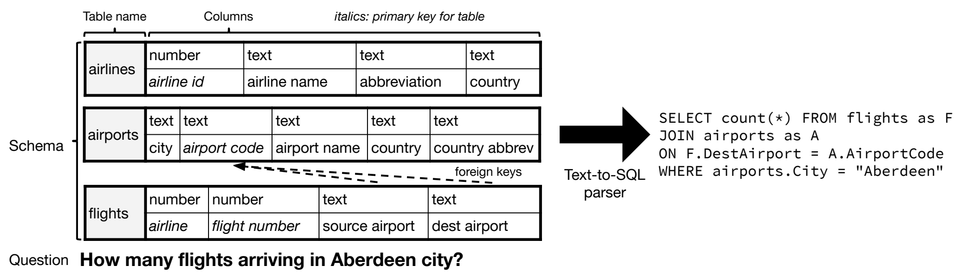

Figure 1: Overview of text-to-SQL task. This paper proposes and evaluates the use of relation-aware self-attention to encode the question and schema, including elements such as the "foreign key" relationship shown.

innovations to the decoder for generating the query, rather than the encoder for the question and the schema. In contrast, most real databases (including those in Spider) contain multiple tables with features such as foreign keys that link rows in one table to another. We hypothesize that to generate correct queries for such databases, a model needs the ability to reason about how the tables and columns in the provided schema relate to each other and use this information in interpreting the question.

In this paper, we develop a method to test this hypothesis. First, we construct a directed graph (with labels on nodes and edges) over all of the elements of the schema. This graph contains a node for each column or table, and an edge exists from one node to another if the two have an interesting relationship (e.g., the two nodes are columns which belong to the same table) with a label encoding that relationship. Each node has an initial vector representation based on the words in the column or table's name. We also obtain a vector representation for each word in the question. For a fixed number of times, we then update each node and word representation based on all other node and word representations, taking the labels of edges between nodes into account. We use these updated representations with a SQL decoder, which uses attention over them at each decoding step, and also points to the column and table representations when it needs to output a column or table reference in the query.

We empirically evaluate our method on the Spider dataset [22], using a decoder based on Yin and Neubig [19]. We achieve 42.94% exact set match accuracy on the development set, significantly higher than the published result of 18.9% [14, 21]. We further verify the utility of directly encoding the relationships within the schema with an ablation study.

2 Problem Formulation

Provided with a natural language question and a schema for a relational database, our goal is to generate the SQL query corresponding to the question. The schema contains the following information, as depicted in Figure 1:

- A list of *tables* in the database, each with a meaningful name (e.g., AIRLINES, AIRPORTS, and FLIGHTS for an aviation database).
- For each table, a list of *columns*. Each column represents an attribute of the entities stored in the table. Each column has a type such as `number` or `text`.
- Each table can designate some of its columns as *primary keys*, which uniquely identify each row in the table.
- A column can have another column in a different table as its *foreign key*, which is used to link together rows across multiple tables.

As mentioned in the introduction, we would like our method to generalize to not only new questions, but also new schemas it has never seen during training time.

3 Motivation for Our Approach

Using natural language to query databases has been a long-standing problem studied for many decades in the research community [2, 11]. We identify several limitations of past work and problem settings:

(a) Some datasets only concern themselves with one domain (e.g., US geography [23]).

(b) Most datasets about one domain also contain only one database schema for the domain, so the system only needs to know how to generate queries for that single schema.

(c) While WikiSQL [24] contains a large number of domains and schemas, each schema only contains one table in it.

(d) Datasets containing only one domain and database necessarily have them overlap across the train and test sets. Furthermore, as discussed by Finegan-Dollak et al. [4], many existing datasets exhibit overlap in queries between the train and test sets, which limits their ability to test how models generalize to generating new queries.

The neural methods common in recent work follow an encoder-decoder paradigm, and past work has largely focused on improvements to the decoder part. As such, the question of how best to encode the question and the schema has remained relatively under-studied. Models developed using datasets which contain only one domain and schema ((a) and (b) above) typically internalize the schema within the learned parameters. The popular WikiSQL dataset necessitates generalizing to new schemas at test time, so models developed for it also encode the schema together with the question; however, as all these schemas only contain one table, the demands placed on the schema encoder are relatively light.

It is most useful if we can train a single model that can generalize to new domains and new database schemas, where both the queries and the schemas have complicated structure that better reflect potential real-world applications. The Spider dataset [22] provides an environment for evaluating this problem setting. In this work, we study how to better encode the question and schema under these more demanding conditions.

4 Existing Encoding Schemes

In this section, we review how some existing works (mostly for the WikiSQL dataset) addressed the challenge of encoding the input question and schema.

Encoding each element independently In SQLNet [17] (for the WikiSQL dataset), the name of each column, and the question, are separately processed using a bidirectional LSTM. The LSTM outputs for the question tokens are utilized in the decoder using attention, and the final LSTM states of the columns with a pointer network. Note that the encoding of each column is uninfluenced by which other column are present; furthermore, the question is encoded entirely separately from the schema.

In SyntaxSQLNet [21] (for the Spider dataset), the question is encoded identically as SQLNet, using a bidirectional LSTM. Each column is encoded similarly, by using a bidirectional LSTM over the concatenation of the words in the column name, words in the table name, and column type (e.g., `number`, `string`).

Encoding the columns jointly TypeSQL [20] computes the encoding of each column by an elementwise averaging of the embeddings of the words in the name, and using a bidirectional LSTM over these averages; therefore, the encoding for each column depends on which other columns are present (and also their order, although that can be arbitrary).

Using the schema while encoding the question Using the information in the schema while encoding the question can help the decoder generate the correct query. In TypeSQL, the word embeddings for each question token are concatenated with a *type* embedding; in particular, question tokens appearing in a column name are specially marked.

Coarse2Fine [3] goes further by using attention to gather information from the schema while encoding the question. First, the input question is encoded using a bidirectional LSTM, then an attention

Type of x	Type of y	Edge label	Description
Column	Column	SAME-TABLE FOREIGN-KEY-COL-F FOREIGN-KEY-COL-R	x and y belong to the same table. x is a foreign key for y. y is a foreign key for x.
Column	Table	PRIMARY-KEY-F BELONGS-TO-F	x is the primary key of y. x is a column of y (but not the primary key).
Table	Column	PRIMARY-KEY-R BELONGS-TO-R	y is the primary key of x. y is a column of x (but not the primary key).
Table	Table	FOREIGN-KEY-TAB-F FOREIGN-KEY-TAB-R FOREIGN-KEY-TAB-B	Table x has a foreign key column in y. Same as above, but x and y are reversed. x and y have foreign keys in both directions.

Table 1: Description of edge types present in the directed graph created to represent the schema. An edge exists from node x to node y if the pair fulfills one of the descriptions listed in the table, with the corresponding label. Otherwise, no edge exists from x to y.

mechanism retrieves a weighted sum of the column embeddings for the LSTM state of each token. These two are concatenated together and processed together in another bidirectional LSTM, to obtain the final embeddings for each question token.

IncSQL [13] uses "cross-serial attention", also updating the column embeddings using the question token embeddings, in addition to the other direction used in Coarse2Fine.

5 Our Approach

In the previous section, we reviewed how previous neural methods developed for the text-to-SQL problem encode the input (the question and the database schema) for use in the decoder. Several of these methods encode the question and the columns entirely independently (e.g., the embedding of a column is uninfluenced by other columns in the schema).

In contrast, we specifically seek interactions between schema elements within our encoder, as explained in Sections 1 and 3. In this section, we describe how we encode the schema as a directed graph and use relation-aware self-attention to interpret it. We will use the following notation:

- c_i for each column in the schema. Each column contains words $c_{i,1}, \cdots, c_{i,|c_i|}$.
- t_i for each table in the schema. Each table contains words $t_{i,1}, \cdots, t_{i,|t_i|}$.
- q for the input question. The question contains words $q_1, \cdots, q_{|q|}$.

5.1 Encoding the Schema as a Graph

To support reasoning about relationships between schema elements in the encoder, we begin by representing the database schema using a directed graph $\mathcal{G}$, where each node and edge has a label. We represent each table and column in the schema as a node in this graph, labeled with the words in the name; for columns, we prepend the type of the column to the label. For each pair of nodes x and y in the graph, Table 1 describes when there exists an edge from x to y and the label it should have.

5.2 Initial Encoding of the Input

We now obtain an initial representation for each of the nodes in the graph, as well as for the words in the input question. For the graph nodes, we use a bidirectional LSTM over the words contained in the label. We concatenate the output of the initial and final time steps of this LSTM to form the embedding for the node. For the question, we also use a bidirectional LSTM over the words. Formally, we perform the following:

$$(\mathbf{c}_{i,0}^{\text{fwd}}, \mathbf{c}_{i,0}^{\text{rev}}), \cdots, (\mathbf{c}_{i,|c_i|}^{\text{fwd}}, \mathbf{c}_{i,|c_i|}^{\text{rev}}) = \text{BiLSTM}_{\text{Column}}(c_i^{\text{type}}, c_{i,1}, \cdots, c_{i,|c_i|}); \quad \mathbf{c}_i^{\text{init}} = \text{Concat}(\mathbf{c}_{i,|c_i|}^{\text{fwd}}, \mathbf{c}_{i,0}^{\text{rev}})$$
$$(\mathbf{t}_{i,1}^{\text{fwd}}, \mathbf{t}_{i,1}^{\text{rev}}), \cdots, (\mathbf{t}_{i,|t_i|}^{\text{fwd}}, \mathbf{t}_{i,|t_i|}^{\text{rev}}) = \text{BiLSTM}_{\text{Table}}(t_{i,1}, \cdots, t_{i,|t_i|}); \quad \mathbf{t}_i^{\text{init}} = \text{Concat}(\mathbf{t}_{i,|c_i|}^{\text{fwd}}, \mathbf{t}_{i,1}^{\text{rev}})$$

$$(\mathbf{q}_1^{\text{fwd}}, \mathbf{q}_1^{\text{rev}}), \cdots, (\mathbf{q}_{|q|}^{\text{fwd}}, \mathbf{q}_{|q|}^{\text{rev}}) = \text{BiLSTM}_{\text{Question}}(q_1, \cdots, q_{|q|}); \quad \mathbf{q}_i^{\text{init}} = \text{Concat}(\mathbf{q}_i^{\text{fwd}}, \mathbf{q}_i^{\text{rev}})$$

where each of the BiLSTM functions first lookup word embeddings for each of the input tokens. The LSTMs do not share any parameters with each other.

5.3 Relation-Aware Self-Attention

At this point, we have representations $\mathbf{c}_i^{\text{init}}$, $\mathbf{t}_i^{\text{init}}$, and $\mathbf{q}_i^{\text{init}}$. Similar to encoders used in some previous papers, these initial representations are independent of each other (uninfluenced by which other columns or tables are present). Now, we would like to imbue these representations with the information in the schema graph. We use a form of self-attention [16] that is relation-aware [12] to achieve this goal.

In one step of relation-aware self-attention, we begin with an input $\mathbf{x}$ of n elements (where $x_i \in \mathbb{R}^{d_x}$) and transform each x_i into $y_i \in \mathbb{R}^{d_z}$. We follow the formulation described in Shaw et al. [12]:

$$e_{ij}^{(h)} = \frac{x_i W_Q^{(h)} (x_j W_K^{(h)} + \mathbf{r_{ij}^K})^T}{\sqrt{d_z/H}}; \quad \alpha_{ij}^{(h)} = \frac{\exp(e_{ij}^{(h)})}{\sum_{l=1}^{n} \exp(e_{il}^{(h)})}$$

$$z_i^{(h)} = \sum_{j=1}^{n} \alpha_{ij}^{(h)} (x_j W_V^{(h)} + \mathbf{r_{ij}^V}); \quad z_i = \text{Concat}(z_i^{(0)}, \cdots, z_i^{(H)})$$

$$\tilde{y}_i = \text{LayerNorm}(x_i + z_i); \quad y_i = \text{LayerNorm}(\tilde{y}_i + \text{FC}(\text{ReLU}(\text{FC}(\tilde{y}_i))))$$

The r_{ij} terms encode the relationship between the two elements x_i and x_j in the input. We explain how we obtain r_{ij} in the next part.

Application Within Our Encoder At the start, we construct the input x of $|c| + |t| + |q|$ elements using $\mathbf{c}_i^{\text{init}}$, $\mathbf{t}_i^{\text{init}}$, and $\mathbf{q}_i^{\text{init}}$:

$$x = (\mathbf{c}_1^{\text{init}}, \cdots, \mathbf{c}_{|c|}^{\text{init}}, \mathbf{t}_1^{\text{init}}, \cdots, \mathbf{t}_{|t|}^{\text{init}}, \mathbf{q}_1^{\text{init}}, \cdots, \mathbf{q}_{|q|}^{\text{init}}).$$

We then apply a stack of N relation-aware self-attention layers, where N is a hyperparameter. We set $d_z = d_x$ to facilitate this stacking. The weights of the encoder layers are not tied; each layer has its own set of weights.

We define a discrete set of possible relation types, and map each type to an embedding to obtain r_{ij}^V and r_{ij}^K. We need a value of r_{ij} for every pair of elements in x. If x_i and x_j both correspond to nodes in $\mathcal{G}$ (i.e. each is either a column or table) with an edge from x_i to x_j, then we use the label on that edge (possibilities listed in Table 1).

However, this is not sufficient to obtain r_{ij} for every pair of i and j. In the graph we created for the schema, we have no nodes corresponding to the question words; not every pair of nodes in the graph has an edge between them (the graph is not complete); and we have no self-edges (for when $i = j$). As such, we add more types beyond what is defined in Table 1:

- If $i = j$, then COLUMN-IDENTITY or TABLE-IDENTITY.
- $x_i \in$ question, $x_j \in$ question:
  QUESTION-DIST-d, where $d = \text{clip}(j - i, D)$. $\text{clip}(a, D) = \max(-D, \min(D, a))$. We use $D = 2$.
- $x_i \in$ question, $x_j \in$ column $\cup$ table; or $x_i \in$ column $\cup$ table, $x_j \in$ question:
  QUESTION-COLUMN, QUESTION-TABLE, COLUMN-QUESTION or TABLE-QUESTION depending on the type of x_i and x_j.
- Otherwise, one of COLUMN-COLUMN, COLUMN-TABLE, TABLE-COLUMN, or TABLE-TABLE.

In the end, we add $2 + 5 + 4 + 4$ types beyond the 10 in Table 1, for a total of 25 types.

After processing through the stack of N encoder layers, we obtain

$$(\mathbf{c}_1^{\text{final}}, \cdots, \mathbf{c}_{|c|}^{\text{final}}, \mathbf{t}_1^{\text{final}}, \cdots, \mathbf{t}_{|t|}^{\text{final}}, \mathbf{q}_1^{\text{final}}, \cdots, \mathbf{q}_{|q|}^{\text{final}}) = y.$$

We use $\mathbf{c}_i^{\text{final}}$, $\mathbf{t}_i^{\text{final}}$, and $\mathbf{q}_i^{\text{final}}$ in our decoder.

Comparison to Past Work We use the same formulation of relation-aware self-attention as Shaw et al. [12]. However, that work only applied it to sequences of words in the context of machine translation, and as such, their r_{ij} only encoded the relative distance between two words. We go beyond by also showing that relation-aware self-attention can be effectively used for encoding more complex relationships that exist within an unordered sets of elements (in this case, columns and tables within a database schema).

Compared to the encoders used in past work such as Coarse2Fine [3] and IncSQL [13], our novel use of relation-aware self-attention frees our encoder from spurious consideration of the order in which the columns and tables are presented in the schema (as the relations we have defined are not impacted by this order).

In their implementation, Shaw et al. [12] shares r_{ij}^K across the H heads and the b examples in a batch, which meant they could use n parallel multiplications of $bH \times (d_Z/H)$ and $(d_z/H) \times n$ matrices. This is possible as r_{ij}^K does not change across the batch when only encoding the relative distances between words. However, due to the more varied relations between x_i in our work, we instead use bn parallel multiplications of $H \times (d_z/H)$ and $(d_z/H) \times n$ matrices, exploiting the fact that we share r_{ij}^K across the H heads.

5.4 Decoder

Once we have obtained an encoding of the input, we used the decoder from Yin and Neubig [19] to generate the SQL query. The decoder generates the SQL query as an abstract syntax tree in depth-first traversal order, by outputting a sequence of *production rules* that expand the last generated node in the tree. However, following SyntaxSQLNet [21], the decoder does not generate the FROM clause; rather, it is recovered afterwards with hand-written rules using the columns referred to in the query. The decoder is restricted to choosing only syntactically valid production rules, and therefore it always produces syntactically valid outputs. To save space, we refer readers to Yin and Neubig [19], although we made the following modifications:

- When the decoder needs to output a column or table, we use a pointer network based on scaled dot-product attention [16] which points to $\mathbf{c}_i^{\text{final}}$ and $\mathbf{t}_i^{\text{final}}$. For choosing a table, we allow the decoder to point to either the correct $\mathbf{t}_i^{\text{final}}$, or any of the $\mathbf{c}_i^{\text{final}}$ for the columns which make up that table.
- At each step, the decoder accesses the encoder outputs $\mathbf{c}_i^{\text{final}}$, $\mathbf{t}_i^{\text{final}}$, and $\mathbf{q}_i^{\text{final}}$ using multi-head attention. The original decoder in Yin and Neubig [19] uses a simpler form of attention.

6 Experiments

In this section, we describe the experiments we conducted to empirically validate our schema encoding approach.

6.1 Experimental Setup

We implemented our model using PyTorch [9]. Within the encoder, we use GloVe word embeddings and hold them fixed during training. All word embeddings have dimension 300. The bidirectional LSTMs have hidden size 128 per direction, and use the recurrent dropout method of Gal and Ghahramani [5] with rate 0.2. Within the relation-aware self-attention layers, we set $d_x = d_z = 256$, $H = 8$, and use dropout with rate 0.1. The position-wise feed-forward network has inner layer dimension 1024. Inside the decoder, we use rule embeddings of size 128, node type embeddings of size 64, and a hidden size of 256 inside the LSTM with dropout rate 0.2 .

We used the Adam optimizer [7] with $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-9}$, which are defaults in PyTorch. During the first $warmup\_steps = max\_steps/20$ steps of training, we linearly increase the learning rate from 0 to 10^{-3}. Afterwards, the learning rate is annealed to 0, with formula $10^{-3}(1 - \frac{step - warmup\_steps}{max\_steps - warmup\_steps})^{-0.5}$. For all parameters, we used the default initialization method in PyTorch. We use a batch size of 50 and train for up to 40,000 steps.

Table 2: Exact match accuracy of different models on the development set of Spider. The first row is the SyntaxSQLNet [21] baseline; the second row is our method; the remainder are ablations on our method.

Model	Easy	Medium	Hard	Extra Hard	All
SyntaxSQLNet	38.40%	15.00%	16.09%	3.53%	18.96%
Our method	57.20%	44.55%	39.66%	21.18%	**42.94**%
No self-attention layers	42.40%	24.77%	22.41%	5.88%	25.53%
2 self-attention layers	53.60%	42.50%	40.80%	17.65%	40.81%
Fewer relation types	44.80%	30.45%	25.86%	7.65%	29.40%
Minimal relation types	42.40%	28.86%	28.74%	7.65%	28.63%
No pretrained word embeddings	40.80%	29.09%	27.01%	5.88%	27.76%

6.2 Dataset and Metrics

We use the Spider dataset [22] for all our experiments. As described by Yu et al. [22], the training data contains questions, queries, and schemas from the Restaurants [10, 15], GeoQuery [23], Scholar [6], Academic [8], Yelp and IMDB [18] datasets. We do **not** use the data augmentation scheme of Yu et al. [21].

As Yu et al. [22] have kept the test set secret, we perform all evaluations using the publicly available development set. We report results using the same metrics as Yu et al. [21]: exact match accuracy on all development set examples, as well as after division into four levels of difficulty. We also measure *component matching* scores, as defined in Yu et al. [22]. As in previous work, these metrics do not measure the model's performance on generating values within the queries. We report results from the snapshot that obtained the best exact match accuracy across 3 repetitions of each configuration.

6.3 Variants Tested

Our main result uses the encoder and decoder described previously, with the number N of relation-aware self-attention layers in the encoder set to 4. To further study the utility of our scheme, we also tried the following variations.

Reduce number of self-attention layers. Set $N = 0$ and $N = 2$. With $N = 0$, there are no relation-aware self-attention layers; we set $\mathbf{c}_i^{\text{final}} = \mathbf{c}_i^{\text{init}}$, $\mathbf{t}_i^{\text{final}} = \mathbf{t}_i^{\text{init}}$, and $\mathbf{q}_i^{\text{final}} = \mathbf{q}_i^{\text{init}}$. As such, the question words, the words in each column's name, and the words in each table's name are encoded separately using bidirectional LSTMs.

Remove relation information from the encoder. We would like to measure the impact of providing to the encoder the 25 relation types we defined earlier. In particular, we want to see whether the self-attention mechanism is sufficient within the encoder to obtain a representation for each schema element that is aware of all of the other schema elements, even if we don't explicitly provide information about how the elements are related.

For "fewer relation types", we exclude all of the types in Table 1, resulting in 15 rather than 25 possible types. For "minimal relation types", we further merge all of {QUESTION,COLUMN,TABLE}-{QUESTION,COLUMN,TABLE} relations into one, as well as {COLUMN,TABLE}-IDENTITY with QUESTION-DIST-0, and so we only have 5 types.

Not using pretrained word embeddings. The Spider dataset only contains 8,659 training examples, which is significantly smaller than many other datasets used in natural language processing. However, there is also reduced overlap in the vocabulary between the training and validation/test sets, as they contain different database schemas and domains. Therefore, we measure the impact of using word embeddings learned from only this dataset.

7 Results and Discussion

Table 2 presents our exact match accuracy results on the development set of Spider, and Table **??** the component-matching F1 scores. For the SyntaxSQLNet row, we obtained the results by running the pretrained model without data augmentation from `https://github.com/taoyds/syntaxSQL`. As expected, our method exceeds the performance of all other configurations tried. In particular, we can see that our method strongly outperforms SyntaxSQLNet [21], the best published baseline, achieving 42.94% exact match accuracy over the 18.96% of the previous work.

Reducing the number of self-attention layers. We can see that the process of relation-aware self-attention is critical for the performance of this encoder, as the accuracy drops precipitiously when the self-attention layers are removed. We observe fairly marginal gains by using 4 such layers (in "Our method") as opposed to 2 ("2 self-attention layers").

Removing relation information from the encoder. Comparing against the rows of "No self-attention layers" and "Our method", we see that while having self-attention layers helps increase performance, it is the relation information provided to the encoder that is responsible for most of the gains.

Not using pretrained word embeddings. Given the small size of the training data, using pretrained word embeddings helps significantly with our result; in fact, our encoder used without pretrained word embeddings performs only slightly better than when we remove all of the self-attention layers but keep the GloVe word embeddings. However, when we evaluate the model without pretrained word embeddings on the subset of the development set where all question words have a learned embedding (i.e. no UNKs in the question; 239 out of 1034 examples), then the exact match accuracy recovers to 40.17%, indicating that UNKs can seriously hurt the performance of the method.

8 Conclusion

This paper proposes the use of relation-aware self-attention [12] when encoding a database schema and a natural language question for the purposes of synthesizing a SQL query. We achieve significantly better results on the Spider dataset than the best published result of Yu et al. [21]. Our ablation study confirms the importance of encoding relations directly in the self-attention mechanism.

References

[1] A large annotated semantic parsing corpus for developing natural language interfaces.: Salesforce/WikiSQL. Salesforce, March 2019. URL `https://github.com/salesforce/WikiSQL`.

[2] I. Androutsopoulos, G. D. Ritchie, and P. Thanisch. Natural Language Interfaces to Databases - An Introduction. *arXiv:cmp-lg/9503016*, March 1995. URL `http://arxiv.org/abs/cmp-lg/9503016`.

[3] Li Dong and Mirella Lapata. Coarse-to-Fine Decoding for Neural Semantic Parsing. *arXiv:1805.04793 [cs]*, May 2018. URL `http://arxiv.org/abs/1805.04793`.

[4] Catherine Finegan-Dollak, Jonathan K. Kummerfeld, Li Zhang, Karthik Ramanathan, Sesh Sadasivam, Rui Zhang, and Dragomir Radev. Improving Text-to-SQL Evaluation Methodology. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 351–360. Association for Computational Linguistics, 2018. URL `http://aclweb.org/anthology/P18-1033`.

[5] Yarin Gal and Zoubin Ghahramani. A Theoretically Grounded Application of Dropout in Recurrent Neural Networks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 1019–1027. Curran Associates, Inc., 2016. URL `http://papers.nips.cc/paper/6241-a-theoretically-grounded-application-of-dropout-in-recurrent-neural-networks.pdf`.

[6] Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer. Learning a neural semantic parser from user feedback. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 963–973, 2017. URL `http://www.aclweb.org/anthology/P17-1089`.

[7] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]*, December 2014. URL `http://arxiv.org/abs/1412.6980`.

[8] Fei Li and H. V. Jagadish. Constructing an interactive natural language interface for relational databases. *Proceedings of the VLDB Endowment*, 8(1):73–84, September 2014. URL `http://dx.doi.org/10.14778/2735461.2735468`.

[9] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. October 2017. URL `https://openreview.net/forum?id=BJJsrmfCZ`.

[10] Ana-Maria Popescu, Oren Etzioni, , and Henry Kautz. Towards a theory of natural language interfaces to databases. In *Proceedings of the 8th International Conference on Intelligent User Interfaces*, pages 149–157, 2003. URL `http://doi.acm.org/10.1145/604045.604070`.

[11] Ana-Maria Popescu, Alex Armanasu, Oren Etzioni, David Ko, and Alexander Yates. Modern Natural Language Interfaces to Databases: Composing Statistical Parsing with Semantic Tractability. In *COLING 2004: Proceedings of the 20th International Conference on Computational Linguistics*, 2004. URL `http://aclweb.org/anthology/C04-1021`.

[12] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-Attention with Relative Position Representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 464–468. Association for Computational Linguistics, 2018. doi: 10.18653/v1/N18-2074. URL `http://aclweb.org/anthology/N18-2074`.

[13] Tianze Shi, Kedar Tatwawadi, Kaushik Chakrabarti, Yi Mao, Oleksandr Polozov, and Weizhu Chen. IncSQL: Training Incremental Text-to-SQL Parsers with Non-Deterministic Oracles. *arXiv:1809.05054 [cs]*, September 2018. URL `http://arxiv.org/abs/1809.05054`.

[14] Spider Leaderboard. Spider: Yale Semantic Parsing and Text-to-SQL Challenge. URL `https://yale-lily.github.io/spider`. Accessed 2019-03-02.

[15] Lappoon R. Tang and Raymond J. Mooney. Automated construction of database interfaces: Intergrating statistical and relational learning for semantic parsing. In *2000 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, pages 133–141, 2000. URL `http://www.aclweb.org/anthology/W00-1317`.

[16] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc., 2017. URL `http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf`.

[17] Xiaojun Xu, Chang Liu, and Dawn Song. SQLNet: Generating Structured Queries From Natural Language Without Reinforcement Learning. *arXiv:1711.04436 [cs]*, November 2017. URL `http://arxiv.org/abs/1711.04436`.

[18] Navid Yaghmazadeh, Yuepeng Wang, Isil Dillig, , and Thomas Dillig. Sqlizer: Query synthesis from natural language. In *International Conference on Object-Oriented Programming, Systems, Languages, and Applications, ACM*, pages 63:1–63:26, October 2017. URL `http://doi.org/10.1145/3133887`.

[19] Pengcheng Yin and Graham Neubig. A Syntactic Neural Model for General-Purpose Code Generation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 440–450. Association for Computational Linguistics, 2017. doi: 10.18653/v1/P17-1041. URL `http://aclweb.org/anthology/P17-1041`.

[20] Tao Yu, Zifan Li, Zilin Zhang, Rui Zhang, and Dragomir Radev. TypeSQL: Knowledge-Based Type-Aware Neural Text-to-SQL Generation. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 588–594. Association for Computational Linguistics, 2018. doi: 10.18653/v1/N18-2093. URL `http://aclweb.org/anthology/N18-2093`.

[21] Tao Yu, Michihiro Yasunaga, Kai Yang, Rui Zhang, Dongxu Wang, Zifan Li, and Dragomir Radev. SyntaxSQLNet: Syntax Tree Networks for Complex and Cross-Domain Text-to-SQL Task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1653–1663. Association for Computational Linguistics, 2018. URL `http://aclweb.org/anthology/D18-1193`.

[22] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921, 2018. URL `http://aclweb.org/anthology/D18-1425`.

[23] John M. Zelle and Raymond J. Mooney. Learning to parse database queries using inductive logic programming. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence - Volume 2*, pages 1050–1055, 1996. URL `http://dl.acm.org/citation.cfm?id=1864519.1864543`.

[24] Victor Zhong, Caiming Xiong, and Richard Socher. Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning. *arXiv:1709.00103 [cs]*, August 2017. URL `http://arxiv.org/abs/1709.00103`.