SeaD: End-to-end Text-to-SQL Generation with Schema-aware Denoising

Anonymous ACL submission

Abstract

On the WikiSQL benchmark, most methods tackle the challenge of text-to-SQL with pre-defined sketch slots and build sophisticated sub-tasks to fill these slots. Though achieving promising results, these methods suffer from over-complex model structure. In this paper, we present a simple yet effective approach that enables auto-regressive sequence-to-sequence model to robust text-to-SQL generation. Instead of formulating the task of text-to-SQL as slot-filling, we propose to train sequence-to-sequence model with Schema-aware Denoising (SeaD), which consists of two denoising objectives that train model to either recover input or predict output from two novel erosion and shuffle noises. These model-agnostic denoising objectives act as the auxiliary tasks for structural data modeling during sequence-to-sequence generation. In addition, we propose a clause-sensitive execution guided (EG) decoding strategy to overcome the limitation of EG decoding for generative model. The experiments show that the proposed method improves the performance of sequence-to-sequence model in both schema linking and grammar correctness and establishes new state-of-the-art on WikiSQL benchmark. Our work indicates that the capacity of sequence-to-sequence model for text-to-SQL may have been under-estimated and could be enhanced by specialized denoising task.

1 Introduction

Text-to-SQL aims at translating natural language into valid SQL query. It enables layman to explore structural database information with semantic question instead of dealing with the complex grammar required by logical -form query. On the WikiSQL benchmark, most models adopt a sketch-based slot filling approach. It decomposes the task of convert query to SQL into several sub-tasks that are relatively easy to handle, e.g., the ‘SELECT’ column mentioned or the query span corresponding to a condition value. The entire SQL can be recovered from the results of the sub-tasks deterministically.

Though being a typical sequence-to-sequence (seq2seq) task, auto-regressive models (LSTM, Transformer, etc.), however, fail to achieve state-of-the-art results for text-to-SQL task. Previous works attribute the sub-optimal results of seq2seq models to three major limitations. First, SQL queries with different clause order may have exact same semantic meaning and return same results by execution. The token interchangeability may confusion model that based on seq2seq generation. Second, the grammar constraint induced by structural logical form is ignored during auto-regressive decoding, therefore the model may predict SQL with invalid logical form. Third, schema linking, which has been suggested to be the crux of text-to-SQL task, is not specially addressed by vanilla seq2seq model.

In this paper, we present a simple yet effective method to boost the performance of seq2seq model for text-to-SQL task. Instead of building extra sub-
module or putting constraint on model output, we propose two novel schema-aware denoising objectives trained along with the original seq2seq generation task. These denoising objectives deal with the intrinsic attribute of logical form and could facilitate schema linking required for text-to-SQL task. The inductive schema-aware noises can be categorized into two types: erosion and shuffle. Erosion acts on schema input by randomly permute, drop and add columns into the current schema set. The related schema entity in target SQL query will be jointly modified according to the erosion result. Shuffle is applied via randomly re-ordering the mentioned entity and values in NL or SQL with respect to the schema columns. During training procedure, shuffle is performed during monolingual self-supervision that trains model to recover original text given the noised one. Erosion is applied to seq2seq task that trains model to generate corrupted SQL sequence, given NL and eroded schema as input. These proposed denoising objectives are combined along with the origin seq2seq task to train a SeaD model. In addition, to deal with the limitation of execution-guided (EG) decoding, we propose a clause-sensitive EG strategy that decide beam size with respect to the clause token that is predicted. The proposed method establish new state-of-the-art on the WikiSQL benchmark.

The main contribution of this work is the schema-aware denoising objectives that are designed for text-to-SQL task. The denoising objectives are model-agnostic and could apply to any seq2seq model that are trained in auto-regressive manner. In addition, we also propose a clause-sensitive EG decoding strategy, which can improve the searching efficiency of EG during seq2seq generation. The results of the work demonstrate the effectiveness of the schema-aware denoising approach and shed lights on the importance of task-oriented denoising objective.

2 Related Work

Semantic Parsing The problem of mapping natural language to meaningful executable programs has been widely studied in natural language processing research. Logic forms (Zettlemoyer and Collins, 2012; Artzi and Zettlemoyer, 2011, 2013; Cai and Yates, 2013; Reddy et al., 2014; Liang et al., 2013; Quirk et al., 2015; Chen et al., 2016) can be considered as a special instance to the more generic semantic parsing problem. As a sub-task of semantic parsing, the text-to-SQL problem has been studied for decades. (Warren and Pereira, 1982; Popescu et al., 2003; Li et al., 2006; Giordani and Moschitti, 2012; Bodik). Slot-filling model (Hwang et al., 2019; He et al., 2019a; Lyu et al., 2020) translates the clauses of SQL into subtasks, (Ma et al., 2020) treat this task as a two-stage sequence labeling model. However, the convergence rate between subtasks is inconsistent or the interaction between multiple subtasks may lead to the model may not converge well. Like lots of previous work (Dong and Lapata, 2016; Lin et al., 2018; Zhong et al., 2017; Suhr et al., 2020; Raffel et al., 2019), we treat text-to-SQL as a translation problem, and taking both the natural language question and the DB as input.

Hybrid Pointer Networks Proposed by (Vinyals et al., 2015), copying mechanism (CM) uses attention as a pointer to copy several discrete tokens from input sequence as the output and have been successfully used in machine reading comprehension (Wang and Jiang, 2016; Trischler et al., 2016; Kadlec et al., 2016; Xiong et al., 2016), interactive conversation (Gu et al., 2016; Yu and Joty, 2020; He et al., 2019b), geometric problems (Vinyals et al., 2015) and program generation (Zhong et al., 2017; Xu et al., 2017; Dong and Lapata, 2016; Yu et al., 2018; McCann et al., 2018; Hwang et al., 2019). In text-to-SQL, CM can not only facilitate the condition value extraction from source input, but also help to protect the privacy of the database. In this paper, We use a Hybrid Pointer Generator Network which is similar to (Jia and Liang, 2016; Rongali et al., 2020) to generate next step token.

Denoising Self-training Language model pretraining (Devlin et al., 2018; Yang et al., 2019; Liu et al., 2019; Lan et al., 2019) has been shown to improve the downstream performance on many NLP tasks and brought significant gains. (Radford et al., 2018; Peters et al., 2018; Song et al., 2019) are beneficial to seq2seq task, while they are problematic for some tasks. While (Lewis et al., 2019) is a denoising seq2seq pre-training model, which is effective for both generative and discriminative tasks, reduces the mismatch between pre-training and generation tasks. Inspired by this, we propose a denoising self-training architecture in training to learn mapping corrupted documents to the original.
3 Methodology

Given natural language question $Q$ and a schema $S$, our goal is to obtain the corresponding SQL query $Y$. Here the natural question $Q = \{q_1, ..., q_{|Q|}\}$ denotes a word sequence, the schema $S = \{c_1, ..., c_{|S|}\}$ is composed of a set of columns, where each column $c_i = \{c_{i1}, ..., c_{i|c_i|}\}$ is a sequence of words. $Y = y_1, ..., y_{|Y|}$ denotes the token-wise raw SQL sequence. We approach this task with directly auto-regressive generation, i.e., predicting the SQL sequence token by token. We choose Transformer as our base architecture, which is a widely adopted in seq2seq translation and generation tasks. In this section, we first present the sample formulation that transform text-to-SQL into typical seq2seq task, followed by a brief introduce of the Transformer architecture with pointer generator. Then we describe the proposed schema-aware denoising method and clause-sensitive EG decoding strategy.

3.1 Sample Formulation

Given training samples $\{X_i, Y_i\}, i = 1, ..., N$, $X = \{Q, S\}$, where $Q$ denotes the NL sequence and $S$ denotes the schema set, $Y$ is the SQL sequence. Sample formulation is a function

$$\hat{X}, \hat{Y} = \text{format}(X, Y)S$$

that transforms heterogeneous data into pairwise token sequence. It is performed by filling template that acts as a prompt to guide seq2seq model to generate different types of token with respected to various contexts. For schema formulation, each column name is prefixed with a separate special token $<\text{col}i>$, where $i$ denotes the $i$-th column in the schema set. The column type of each column is also append to the name sequence to form the template for a schema column $<\text{col}i> \text{ col name : col type}$. All columns in schema is formulated and concatenated together to compose the input sequence for schema. The schema sequence is further concatenated with the NL sequence for model input. We explicitly introduce schema-mention alignment to NL sequence by surrounding schema names that are mentioned in NL sequence with bracket tokens [], in order to improve the learning of schema linking.

For SQL sequence, we initialize it with raw SQL query and perform several modifications on it: 1) surrounding entities and values in SQL query with a "" token, and dropping other surroundings if exist; 2) replacing col entities with their corresponding separate token in schema; 3) inserting spaces between punctuation and words. The formulated SQL sequence is illustrated in Figure 1. The formatting procedure improves consistency between tokenized sequences of source and target, and contributes to the identification and linking of schema entities.

3.2 Transformer with Pointer

Following the previous works on seq2seq semantic parsing, we use Transformer (Vaswani et al., 2017) as the backbone of our model. The vanilla Transformer generate tokens with a feed-forward layer that computes the unnormalized score over the target vocabulary. In text-to-SQL task, however, most
schema and value mentions can be extracted from the input sequence. Therefore, we adopt a Hybrid Pointer Generator Network (Jia and Liang, 2016) in our architecture to generate tokens from the target vocabulary \( V \) or copy from the input context. During inference, input sequence \( X \) is first encoded into a sequence of hidden states \( H_{\text{enc}} \). Then, the decoder produces the hidden states \( h_{\text{dec}} \) for step \( t \) based on previously generated sequence and encoded output. The unnormalized scores \( \text{scores}_e = \{s_1, ..., s_{|V|}\} \) over \( V \) can be obtained from \( h_{\text{dec}} \) through a feed-forward layer. \( V = \{V_q, V_c, V_s\} \) is the target vocabulary, where \( V_q \) denotes corpora token vocabulary, \( V_c \) denotes column token set and \( V_s \) denotes available SQL keywords, e.g. SELECT, MAX, MIN, etc. The decoder output \( h_{\text{dec}} \) is also used to compute the unnormalized attention scores \( \text{scores}_a = \{i_1, ..., i_{|X|}\} \) over the input sequence tokens, where \( |X| \) is the sequence length.

We concatenate \( \text{scores}_a \) and \( \text{scores}_e \) to get the hybrid score \( \text{score}_{\text{hybrid}} = \{s_1, ..., s_{|V|}, i_1, ..., i_{|X|}\} \), where the first \( |V| \) elements represent the output distribution of the target vocabulary \( V \) and the remained \( |X| \) are pointers tokens referred to corresponding input tokens. The final probability distribution is computed by \( P = \text{softmax}(\text{score}_{\text{hybrid}}) \), to determine the next token during generation.

### 3.3 Schema-aware Denoising

Similar to masked language modeling and other denoising task, we propose two schema-aware objectives, erosion and shuffle, that train model to either reconstruct the origin sequence from noising input or predict corrupted output otherwise. The denoising procedure is illustrated in Figure 2.

#### 3.3.1 Erosion

Given input sample \( \{Q, S, Y\} \), erosion corrupts the schema sequence \( S \) with a serial compositions of three noising operations:

- **Permutation** Re-order the concatenation sequence of schema columns during schema formulation.
- **Removal** For each column, remove it with a dropping probability \( p_{\text{drop}} \).
- **Addition** With a addition probability \( p_{\text{add}} \), extract a column from another schema that exists in the training database and insert it into current schema set.

During all operations above, the order of separating special tokens remains unchanged, therefore the corresponding anonymous entities in SQL query should be updated along with the erosion operations in schema sequence. In particular, if a column entity mentioned in SQL query is removed during erosion, we substitute the corresponding column token in SQL with a masking token \(<\text{unk}>\) to cope with the absence of the schema information. With such joint modification for schema and SQL sequence, the model is required to identify the schema entities that are truly related to the NL question and learns to raise an unknown exception whenever the schema information is insufficient to compose the target SQL.

#### 3.3.2 Shuffle

Given input sequence \( X' = \{Q, S\} \), where \( Q = \{Q, Y\} \), the shuffle noise reorders the mentioning sequence of entities in the source query while the schema sequence \( S \) is fixed. The denoising objective trains model to reconstruct the query sequence \( Q \) with entities in correct order. The objective of recovering shuffled entity orders trains model to capture the inner relation between different entities and therefore contributes to the schema linking performance. It is also notable that, as a self-supervision objective, both \( Q \) and \( Y \) are engaged.

---

**Algorithm 1:** Training procedure for schema-aware denoising

**Input:** training corpus

\[ \mathcal{X} = \{(Q_i, S_i, Y_i)\}, i \in 1, \ldots |\mathcal{X}| \]

S2S Transformer \( \Theta \)

**foreach** \((Q_i, S_i, Y_i) \in \mathcal{X}\) **do**

\[
\begin{align*}
T_{\text{src}}, T_{\text{tgt}} &\leftarrow Q_i, Y_i; \\
T_{\text{tgt}}, S_i &\leftarrow \text{Erosion}(T_{\text{tgt}}, S_i) \\
&\text{with } P_{\text{shuffle}} \text{ do} \\
&\quad \text{with } P_{\text{swap}} \text{ do} \\
&\quad \quad T_{\text{src}}, T_{\text{tgt}} \leftarrow T_{\text{tgt}}, T_{\text{src}}; \\
&\quad \text{end} \\
&\quad T_{\text{src}} \leftarrow \text{Shuffle}(T_{\text{tgt}}) \\
&\text{end} \\
&\text{if } T_{\text{type}} = \text{SQL} \text{ then} \\
&\quad T_{\text{prefix}} \leftarrow <\text{2sql}>; \\
&\text{else} \\
&\quad T_{\text{prefix}} \leftarrow <\text{2nl}>; \\
&\text{end} \\
&\text{TrainOneSample}(T_{\text{src}}, T_{\text{tgt}}, \Theta) \\
\end{align*}
\]
in this denoising task and get trained separately. Though we dependent on the SQL query to identify the value entities in NL query, order shuffling with only column entities is sufficient to obtain promising performance. Since no parallel data is required, additional corpus with monolingual data for both SQL and NL could help with the re-order task and will be one of the further direction of this work.

### 3.3.3 Training Procedure

Inspired by works on denoising self-training (Song et al.; Lewis et al.), we propose to train the schema-aware denoising objectives along with the primary seq2seq task. During training, for each training sample, we apply a nosing pipeline to it before feeding it into the model. The noises with different type are applied to the sample individually. Through the control of activate probability, they could share the same weights in the overall objective. Such continual noising pipeline generates random-wise corrupted samples during training. It prevents the model from fast over-fitting and could yield results with better generalization (Siddhant et al.). The whole procedure is summarized in Algorithm 1.

### 3.4 Clause-sensitive EG Decoding

During the inference of text-to-SQL task, the predicted SQL may contain errors related to inappropriate schema linking or grammar. EG decoding (Wang et al., 2018) is proposed to amend these errors through an executor-in-loop iteration. It is performed by feeding SQL queries in the candidate list into the executor in sequence and discarding those queries that fail to execute or return empty result. Such decoding strategy, while effective, suggests that the major disagreement in the candidate list focuses on schema linking or grammar. Directly perform EG to the candidates generated with beam search leads to trivial improvement, as the candidates consist of redundant variations focuses on selection or schema naming, etc. This problem can be addressed by setting the beam length of most of the predicted tokens to 1 and releasing those tokens related to schema linking (e.g., WHERE). We also notice that there are cases that combine incorrect schema linking with some aggregation in SELECT clause, which return some trivial results such as 0, thus suppress the EG filter. To mitigate the issue, we suggest to drop aggregate operator in SELECT during EG to maximize the effectiveness of it. Note that with such strategy, the condition with inequation in WHERE clause should be dropped together to ensure the validity of the ground-truth SQL results.

### 4 Experiment

To demonstrate the effectiveness of the proposed method, we evaluate the proposed model on WikiSQL benchmark and compare it to other state-of-the-art methods.

<table>
<thead>
<tr>
<th>Model</th>
<th>Dev</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQLNet</td>
<td>63.2</td>
<td>69.8</td>
</tr>
<tr>
<td>SQLova</td>
<td>81.6</td>
<td>87.2</td>
</tr>
<tr>
<td>X-SQL</td>
<td>83.8</td>
<td>89.5</td>
</tr>
<tr>
<td>HydranaNet</td>
<td>83.6</td>
<td>89.1</td>
</tr>
<tr>
<td>SeaD</td>
<td>84.9</td>
<td>90.2</td>
</tr>
<tr>
<td>IESQL ♦</td>
<td>84.6</td>
<td>89.7</td>
</tr>
<tr>
<td>BRIDGE ♦</td>
<td>86.2</td>
<td>91.7</td>
</tr>
<tr>
<td>SDSQL ♦</td>
<td>86.0</td>
<td>91.8</td>
</tr>
</tbody>
</table>

Table 1: Accuracy (%) of logic form ($Acc_lf$) and execution ($Acc_{ex}$) of our model SeaD and other competitors. Best results in bold. EG: execution-guided decoding. EG$_{CS}$: the proposed clause-sensitive EG strategy for S2S generation. ♦ denotes methods that leverage additional annotation of dataset. ♦ denotes methods that utilize database content during training.

### 4.1 Dataset

As the largest human-annotated dataset of text-to-SQL, WikiSQL consists of 56, 355, 8, 421 and 15, 878 NL-SQL pairs for training, validation and inference respectively. All ground-truth SQL queries are guaranteed with at least one query result. Each SQL contains SELECT clause with at most one aggregation operator and WHERE clause with at most 4 conditions that connected by AND. Each SQL is associated with a schema in database.

### 4.2 Implementation details

We implement our method using AllenNLP (Gardner et al.) and Pytorch (Paszke et al.). For the model architecture, we use Transformer with 12 layers in each of the encoder and decoder with a hidden size of 1024. We initialize the model weight with bart-large pretrained model provided by
Huggingface community (Wolf et al.) and fine-tune it on training dataset for 20 epochs. The batch size during training is set to 8 with a gradient accumulation step of 2. We choose Adam (Kingma and Ba) as the optimizer and set the learning rate to $7e^{-5}$ with a warm-up step ratio of 1%. The weight decay for regulation is set to 0.01. We set the activation probability $P_{swap} = 0.5$ and $P_{shuffle} = 0.3$ to balance the weight between self-supervision and seq2seq objective. $P_{drop}$ for column removal in erosion is set to 0.1. The early stop patience is set to 5 with respect to the BLUE metric (Papineni et al.) on validation set. The overall training procedure spend around 3 hours on an Ubuntu server with 8 NVIDIA V100 GPUs.

### 4.3 Competitors

We compare the proposed method to the following models: (1) SQLNet (Xu et al., 2017) is a sketch-based method; (2) SQLova (Hwang et al., 2019) is a sketch-based method which leverage the pre-trained language model for representation; (3) X-SQL (He et al., 2019a) enhances the structural schema representation with contextual embedding; (4) HydraNet (Lyu et al., 2020) transforms schema linking into column-wise matching and ranking; (5) IESQL (Ma et al., 2020) treats text-to-SQL as a sequence labeling task; (6) BRIDGE (Lin et al., 2020) is a sequential architecture for modeling dependencies between natural language question and related schema; (7) SDSQL (Hui et al., 2021) is a multi-task model with explicitly schema dependency guided module.

### 4.4 Comparison with State-of-the-art Models

The comparison results are summarized in Table 1. Models suffixed with ♦ leverage additional annotation of the dataset. Models suffixed with ♣ utilize database content during training procedure. Without using EG, SeaD significantly outperforms all models without the auxiliary of table content or schema linking annotation. When combined with EG decoding, SeaD achieve best performance ever compared to those models that utilize additional training information. It indicates the effectiveness of the proposed denoising objectives on modeling text-to-SQL through vanilla seq2seq. Notably, the annotation noise makes aggregation prediction a major challenge for WikiSQL. Previous works suggested to improve AGG prediction via rule-based annotation amendment. As shown in Table 2, we argue that the proposed aggregation dropping strategy for EG achieves comparable enhancement, while less human effort is involved. Combined with the AGG dropped clause-sensitive EG, the SeaD model establishes new state-of-the-art on WikiSQL benchmark.

To analyze the detailed improvement for SeaD on text-to-SQL task, in Table 3 we report the accuracy on WikiSQL test set with respect to several SQL components with and without EG decoding. SeaD shows promising results on column selection, aggregation, where column and where value prediction. It outperforms all method except SDSQL, which leverages rule-based annotation of schema linking. After applying EG decoding, SeaD achieves best performance on four out of five components among all competitors.

### 4.5 Ablation Study

To evaluate the contribution of each proposed objective, we perform ablation study to SeaD (4) with
WikiSQL dataset. We start from the Bart model and add components to it in sequence. The pointer net contributes to 1.2% absolute improvement of Acc\textsubscript{f} on test set. Combine text infilling, an effective denoising objective utilized by Bart, into training procedure brings 0.3 absolute Acc\textsubscript{f} improvement. On the other hand, erosion and shuffle objectives contribute to 1.5% and 0.6% absolute Acc\textsubscript{f} improvement for SeaD on test set respectively. It demonstrates the effectiveness of the schema-aware denoising objective for improving seq2seq generation in text-to-SQL task.

5 Conclusions

In this paper, we proposed to train model with novel schema-aware denoising objectives, which could improve performance of seq2seq generation for text-to-SQL task. Combined with the proposed clause-sensitive EG decoding strategy, our model achieves state-of-the-art on the WikiSQL benchmark. The success of the SeaD highlights the potential of utilizing task-oriented denoising objective for seq2seq model enhancement.

Table 4: Ablation study for SeaD model on WikiSQL benchmark.

<table>
<thead>
<tr>
<th>Model</th>
<th>Dev</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Acc\textsubscript{f}</td>
<td>Acc\textsubscript{ex}</td>
</tr>
<tr>
<td>Bart</td>
<td>81.4</td>
<td>87.1</td>
</tr>
<tr>
<td>Bart+ptr</td>
<td>82.8</td>
<td>88.6</td>
</tr>
<tr>
<td>Bart+ptr + infilling</td>
<td>82.8</td>
<td>88.7</td>
</tr>
<tr>
<td>SeaD (Shuffle-only)</td>
<td>83.5</td>
<td>89.0</td>
</tr>
<tr>
<td>SeaD (Erosion-only)</td>
<td>84.2</td>
<td>89.6</td>
</tr>
<tr>
<td>SeaD</td>
<td>84.6</td>
<td>90.2</td>
</tr>
</tbody>
</table>

References


Chenglong Wang Alvin Cheung Rastislav Bodik. Synthesizing highly expressive sql queries from input-output examples.


Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. MASS: Masked sequence to sequence pre-training for language generation.


