Syntax- and Execution-Aware SQL Generation with Reinforcement Learning

Anonymous EMNLP-IJCNLP submission

Abstract

Text-to-SQL task aims at mapping natural language utterances into structured SQL queries. In this paper, we design a new score to evaluate the generated SQL queries, taking into account the structure of knowledge bases and the syntax of SQL queries. To improve the generation of SQL queries, we utilize this score as a reinforced objective in the sequence-to-sequence models. Experimental results on wikiSQL dataset show that our proposed approaches significantly improve the performance on all metrics. The new score outperforms other rewards which are used with reinforcement learning in previous existing systems, and is competitive with the execution guidance which is used in decoding by interacting with knowledge bases.

1 Introduction

Semantic parsing is concerned with mapping natural language utterances to executable programs (Zelle and Mooney, 1996; Wong and Mooney, 2007; Zettlemoyer and Collins, 2012; Ling et al., 2016; Iyer et al., 2017). It attracts much interests from both academia (Yaghmazadeh et al., 2017) and industry (Zhong et al., 2017), especially in text-to-SQL task (Li et al., 2006; Wang et al., 2017; Guo et al., 2018; Misra et al., 2018). As SQL queries are structured objects fitting SQL syntax and executable on knowledge bases, more studies are made to develop syntax- and execution-aware neural architectures. Syntax-aware examples include decoders with multiple sub-models which restrict SQL queries to the fixed form of "select-aggregator-where" (Xu et al., 2017; Yu et al., 2018a), two decoders orderly used to generate SQL sketch in the first stage and fill in details in the second stage (Dong and Lapata, 2018), modular decoder which consists of sets of modularized models defined based on SQL syntax and trained independently (Yu et al., 2018b). Execution-aware examples include models trained with reinforcement learning using the correctness of the generated queries' execution as the reward (Zhong et al., 2017; Sun et al., 2018), models which utilize the execution of partially generated SQL queries to prune the space in the decoding procedure (Wang et al., 2018b). However, as shown in Figure 1, a subquery with valid syntax may be not executable, e.g., the subquery of "Name=Sweden"; and a subquery with improper syntax can be executed, e.g., the subquery of "Nationality=Name".

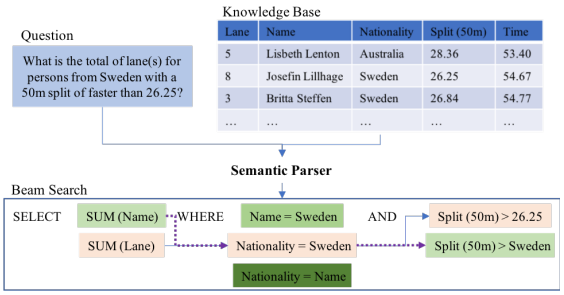

Figure 1: An example of the SQL query with multiple subqueries, the correct parsing results are connected by the solid line. Here, "*Nationality = Name*" doesn't satisfy SQL syntax as subqueries of conditions are in the pattern of "column operator cell", its scores 0.33; "*Name = Sweden*" violates KBs structure as "*Sweden*" doesn't belong to the column of "*Name*", its scores 0.67; other subqueries (e.g. "*SUM(Name)*") don't conform to semantic constraints of KBs as the aggregator need to operate on the number value.

To satisfy SQL syntax and execution, we design a score called SEA score (standing syntax- and execution-aware score) to estimate the probability of the generated SQL query being valid and executable, accounting for the structure of knowledge bases[1] (KBs) and SQL syntax. A SQL query can be represented as a syntax-based parse tree (as shown in Figure 2) whose leaf nodes are elements of the

[1]Tables or relational databases in SQL.

query and branch nodes are generated from SQL syntax. We assign a SEA score for each branch node according to how well its child nodes satisfy the syntax and the KB structure (see Section 3 for details). Then the score for a query is computed from bottom to up and evaluates the validity and executability of the query. For the example in Figure 1, the score of the query connected by the solid line is 1, and of the query connected by the dotted line is 0.77. We utilize this score as the reinforced objective to guide the training of models. The proposed score has at least three advantages. Firstly, this score evaluates executability of a generated query, both the structural and semantic constraints from KBs can be exploited during parsing to ensure the executions of queries. Secondly, this score evaluates SQL syntax at multiple granularities, so syntax satisfaction is ensured for both subqueries and queries during generation. Thirdly, this score is utilized as the reinforced objective during training, and no additional operation is required in prediction.

In our framework, we utilize a Seq2Seq model based on pointer network (Vinyals et al., 2015) as the base generative model, and combine it with reinforcement learning (RL) (Williams, 1992) using SEA score as the reward. The SEA score is employed to evaluate the generated SQL queries and give feedback to guide the learning of the generative model. A new objective function which combines the maximum-likelihood cross-entropy loss and rewards from policy gradient methods is used as the goal of training. In summary, we mainly make the following contributions:

- We design a new score called SEA score to evaluate the generated SQL queries, taking KB structures and SQL syntax into account.
- We utilize the SEA score as the reinforced objective to guide the learning of the base model. Experimental results show that the proposed approach obtains better performance compared with previous systems.

2 Related Work

Semantic Parser. Text-to-SQL task has been studied for decades (Warren and Pereira, 1982; Li et al., 2006; Wang et al., 2017). More recently, neural Seq2Seq models have been applied to semantic parsing with promising results (Zhong et al., 2017; Dong and Lapata, 2018; Sun et al., 2018),

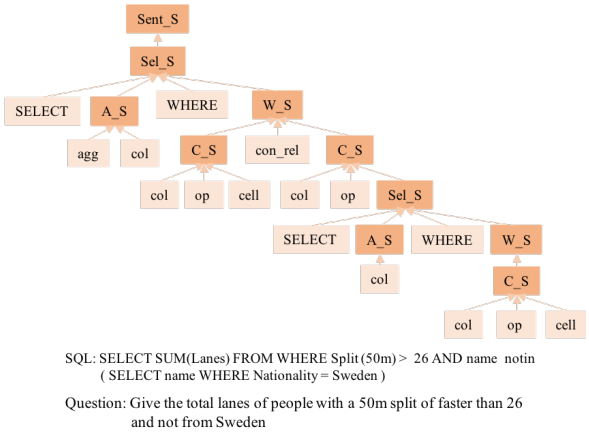

Figure 2: An example of a SQL query's derivations (defined in Table 1) in a recursive manner.

viewing semantic parsing as a natural-language-to-SQL problem. Zhong et al. (2017) develops an augmented pointer network, which is further improved with RL for SQL sequence prediction. Xu et al. (2017) adopts a sequence-to-set model to predict columns of WHERE clause, and uses attentional model to predict the slots in WHERE clause. Dong and Lapata (2018) decomposes the decoding into two stages, the first decoder focuses on predicting a rough sketch of the meaning representation and the second decoder fills in missing details by conditioning on the input and the sketch. Sun et al. (2018) develops a pointer network with multi-channels, and utilizes column-cell relation to improve the generation of WHERE clause.

Reinforcement Learning. The REINFORCE algorithm (Williams, 1992) has been successfully applied to improve the current state-of-the-art Seq2Seq models in a variety of NLP tasks (Li et al., 2016; He et al., 2017; Paulus et al., 2017; Wang et al., 2018c). In supervised text-to-SQL task, due to the orders of subqueries, the correctness of execution output by the generated SQL query is not always consistent with the exact match of the generated query and ground truth. Zhong et al. (2017) and Sun et al. (2018) use RL with the correctness of answer as the reward to improve the generation of SQL queries. In weakly supervised tasks, Misra et al. (2018) uses policy shaping to bias the search procedure for good candidate parses and process the problem of spurious programs.

Execution Guidance. The execution guidance (EG) is used in decoding to avoid generating queries with no results. It is utilized to detect and exclude faulty subqueries during the decoding

Groups	Explanation
tab, *col*, *cell*	Units of KBs, table name, column name, content of a (row, column)
limv	Digital value, limiting the number of results following the keyword of *order by*
agg	Aggregators, {min, max, count, sum, avg}
op	Operators, {>, <, =, !=, >=, <=, like, in, not in, between}
con_rel	Relations between conditions in WHERE clause, {and, or, not}
sub_rel	Relations between SQL queries, {intersect, union, except}

Constituents	corresponding sub-constituents	Conditions of being valid and executable: $\{\text{details}\}_{factor}$
Sent_S	*Sel_S* (*sub_rel Sel_S*$_2$)$\star$	$\{sub\_rel\}_{gtruth}$ $\{(sub\_rel, Sel\_S_2)\}_{syntax}$
Sel_S	select (*A_S*)+ from *F_S* where (*W_S*)? (*G_S*)$\star$ (*R_S*)$\star$	{(select, *A_S*), (from, *F_S*), (where, *W_S*), *num*(*A_S*), *num*(*F_S*), *num*(*W_S*), *num*(*G_S*), *num*(*R_S*)}$_{syntax}$
A_S	(*agg*)? *col*	$\{agg, col\}_{gtruth}$ $\{(agg, col)\}_{dtype}$
F_S	*tab* (join *tab*$_2$ on *col*$_1$ = *col*$_2$)$\star$	$\{tab, tab_2, col_1, col_2\}_{gtruth}$ $\{(col_1, tab), (col_2, tab_2), (col_1, col_2)\}_{kbstructure}$
G_S	group by (*col*)+ (having *C_S*)?	{(having, *C_S*), (group, having), *num*(having)}$_{syntax}$
R_S	order by desc/asc (*col*)+ (limit *limv*)?	$\{col, \text{desc/asc}, limv\}_{gtruth}$ {(order, limit), *num*(limit)}$_{syntax}$
C_S	*col op cell* / *Sel_S*	$\{col, op, cell / Sel\_S\}_{gtruth}$ $\{(col, op), (op, cell))$ / (*op*, select *col* of *Sel_S*)$\}_{dtype}$ $\{(col, cell), (col,$ select *col* of *Sel_S*)$\}_{kbstructure}$
W_S	*C_S* (*con_rel C_S*$_2$)$\star$	$\{con\_rel\}_{gtruth}$ $\{(con\_rel, C\_S_2)\}_{syntax}$

Table 1: The SEA score of SQL constituents. The second row of the first table shows units from KBs and questions, and SQL keywords (in lower case) are in the third row. The second table shows SQL constituents and their corresponding sub-constituents (derivations), where conditions are set according to SQL syntax ($\{\}_{syntax}$), structural ($\{\}_{kbstructure}$) and semantic constraints ($\{\}_{dtype}$) from KBs. $\{c_1, ..., c_{|c|}\}_{gtruth}$ represents each constituent c_i matches with the ground truth and returns a value of true or false. $\{(c_i, c_j)\}_{syntax}$ represents whether the constituents c_i and c_j can construct a valid subquery. $\{num(c_i)\}_{syntax}$ represents whether the number of c_i is valid, where *num*(a) represents the number of a. $\{(c_i, c_j)\}_{kbstructure}$ judges whether c_i and c_j conforms to KB structures. $\{(c_i, c_j)\}_{dtype}$ judges whether the data types of c_i and c_j match. $\star$ means 0 or more times; $+$ means 1 or more times; ? means 0 or 1 times.

procedure by conditioning on the execution of partially generated SQL query (Wang et al., 2018b; Shi et al., 2018). The key insight is that a partially generated query can already be executed, and the results of that execution can be used to guide the generation procedure.

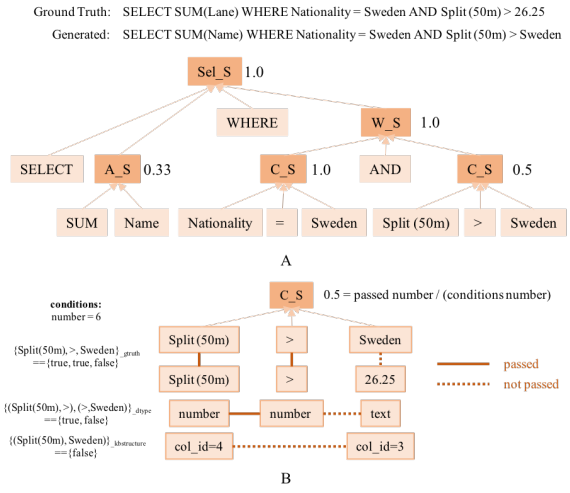

Figure 3: An example of the SEA score computation. The graph A shows the computation process of a query, whose score is $(0.33 + 1 + 0.5 + 1 + 1)/5 \approx 0.77$. B shows the computation of a constituent C_S (other constituents' shown in Figure 7 of Appendix A), conditions ($\{\}_{gtruth}$, $\{\}_{dtype}$, $\{\}_{kbstructure}$) are from Table 1.

To satisfy syntax and improve execution, we design a new score and utilize it as a reward to guide the learning of models during training.

3 SEA Score Definition

The text-to-SQL task takes a question q and a KB b as the input, and outputs a SQL query y corresponding to the question. A SQL query consists of KB units, SQL keywords, and values from the question, as shown by the example in Figure 1.

A SQL query has two characteristics:

- A SQL query is a structural meaning representation based on an underlying formalism or grammar. A valid SQL query can be derived from this grammar (Date, 1994).

- A SQL query could be executed on KBs to output the answer. An executable SQL query should conform to structural and semantic constraints of KBs, and each subquery should be executable (Wang et al., 2018b).

According to these characteristics, we design the SEA score with consideration of three factors: SQL syntax, structures and semantic constraints of KBs, as elaborated below:

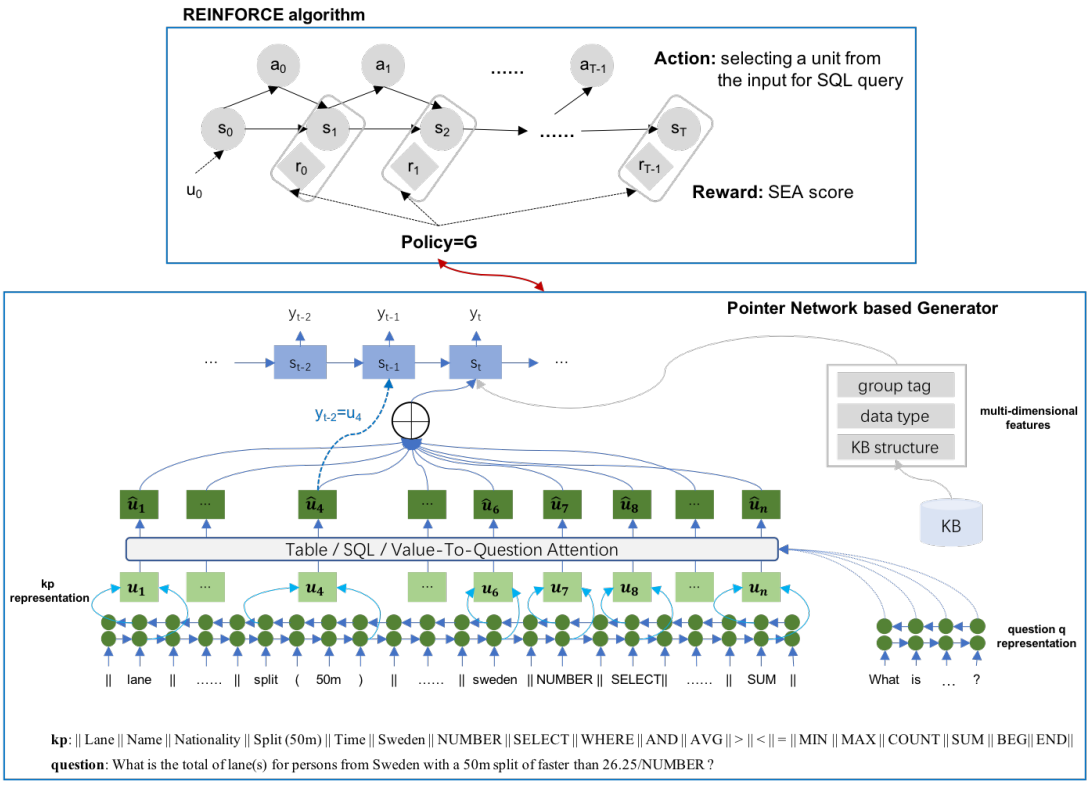

Figure 4: Overview of our method. A Seq2Seq model based on pointer network is used as the base generative model (Section 4.1), which is further improved with RL using the SEA score as the reward (Section 4.2).

- SQL syntax: a SQL query is composed of multiple subqueries or constituents in a recursive manner, each introduced by a SQL keyword or is composed of other subqueries (as shown in Figure 2). We give a summary of SQL syntax from which SQL queries can be derived, as elaborated in the bottom part of Table 1. The first and second columns show derivations of subqueries, and conditions are shown in the third column. In order to simplify derivations, we group SQL keywords and KB units according to their functions, as shown in the top part of Table 1.

- KB structures: there are table-column and column-cell relations in a KB, representing a column belongs to a table and a cell belongs to a column respectively. A SQL query is executable only if it conforms to KB structures.

- KB semantic constraint: we define two data types: *number* and *string* for KB units[2] and SQL keywords[3], denoting the type of value for KB units and the type of argument[4] for SQL keywords. A SQL query is executable only if the data types of its SQL keywords and KB units are matching.

The SEA score of a SQL query is the average of scores of all constituents, calculated as follows:

$$r(\hat{y}, y^g) = \frac{1}{m}\sum_{i=1}^{m} S(c_i) = \frac{1}{m}\sum_{i=1}^{m} \frac{pcon_i}{con_i} \quad (1)$$

where $\hat{y}$ is the generated SQL query with m constituents, y^g is the ground truth. c_i is the i-th constituent. con_i represents the number of conditions of the i-th constituent (defined in Table 1). $pcon_i$ represents the number of passed conditions of the i-th constituent. $S(c_i) = \frac{pcon_i}{con_i}$, representing the SEA score of the i-th constituent, is the proportion of passed conditions, i.e., $\{condition\}_{factor}$ returns a value of true. For example, the SEA score of the incorrect query in Figure 1 is 0.77, its computation is shown in Figure 3.

[2]The data type of a column depends on the data types of its cells, and the data type of a cell depends on itself.

[3]The data type of (min, max, sum, avg) and (>, >=, <, <=) is *number*, and others *string*.

[4]A keyword operates on an argument, for example *SUM(Lane)*, where *Lane* is the argument of *SUM*.

4 Models with Grammar Score

We explore the way of training our base generative model using the SEA score as the reinforced objective, as shown in Figure 4. The whole architecture mainly consists of two sub modules:

- **Generator of SQL queries**: based on the natural language questions and KBs, the generator G[5] aims to generate SQL queries. We use a Seq2Seq model as the generator.
- **Improved with RL**: the standard supervised prediction is combined with RL during training, using the SEA score as the reward. The combined method learns a policy by optimizing the long-term reward from ongoing simulations (Williams, 1992).

4.1 Generator of SQL queries

As shown in the bottom of Figure 4, a Seq2Seq model is used as the backbone of the generator. The model for text-to-SQL task is similar to a pointer network with augmented inputs (Zhong et al., 2017; Sun et al., 2018), as the output space of the generated SQL query is limited to SQL keywords and KB units[6]. Our generator also utilizes a pointer in the decoder.

Table-Aware Input. We employ the special token "||" to concatenate KB units and SQL keywords to get a new input sequence kp (as shown in Figure 4), which is different from Zhong et al. (2017) and Sun et al. (2018). KB units consist of all columns and the related cells obtained by searching over KBs (see Section 5.2 for details). $kp = ||u_{1,1}...u_{1,|u_1|}||...||u_{n,1}...u_{n,|u_n|}||$, where the k-th span $u_k = u_{k,1}...u_{k,|u_k|}$ with $|u_k|$ words is called a unit of kp. So in our case, there are two input sequences, one is the natural language question q, the other is the output space kp.

In order to improve the generation of SQL queries, features are collected according to the SEA score, including group tags, data types and relations from KB structures (see Section 3). They are used as additional information in both the encoder and decoder, and demonstrated to be effective in the next experiments.

[5]We also label it as G_θ, θ represents the parameters of G.

[6]In general, a SQL query includes tokens from questions as the values of conditions in WHERE clauses. As these tokens are in the pattern of number, we assign them into *NUMBER* in both questions and KBs. Then they can be mapped to cells of KBs even though they don't exactly match any cell.

BiLSTM Encoder. We use two BiLSTMs to encode the two input sequences q and kp respectively. For the word w_j of q, the hidden states of both directions are concatenated as the final hidden state $h_j = [\overrightarrow{h_j}; \overleftarrow{h_j}]$. For the unit u_k of kp, the LSTM hidden states at positions $u_{k,1}$ and $u_{k,|u_k|}$ are concatenated as the hidden state $\mathbf{u}_k = [\overrightarrow{u_{k,|u_k|}}; \overleftarrow{u_{k,1}}]$. All concatenated vectors $\{\mathbf{u}_k\}_{k=1}^n$ are used as the encoding vectors for kp. We use an attention mechanism towards questions encoding to obtain the most relevant words for each unit of kp. The attention score from $\mathbf{u}_k$ to h_i is computed via $a_{k,i} \propto exp\{e(\mathbf{u}_k) \cdot e(h_i)\}$, where $e(\cdot)$ is a one-layer neural network, and $\sum_i a_{k,i} = 1$. Then we compute the new vector $\hat{\mathbf{u}}_k = \sum_i a_{k,i} h_i$ to replace $\mathbf{u}_k$ in other model components. The final representation of inputs of the decoder is $\{\hat{\mathbf{u}}_k\}_{k=1}^n$.

LSTM Decoder. We use an attention-based LSTM decoder (Bahdanau et al., 2014), which selects a unit from the input sequence to replicate at each time-step. The probability of selecting the i-th unit u_i at the t-th time-step is calculated as Equation 2, where s_t^{dec} is the decoder hidden state at the t-th time-step, $\hat{\mathbf{u}}_i$ is the encoding representation of the unit u_i, W_a is the model parameter.

$$p(\hat{y}_t = u_i | \hat{y}_{<t}, kp) \propto exp(W_a[s_t^{dec}; \hat{\mathbf{u}}_i]) \quad (2)$$

4.2 Improved with RL

The SEA score is designed to evaluate syntax and execution of a SQL query at multiple granularities. We utilize it to improve SQL query generation by combining the RL methods in decision-making with Seq2Seq models. The policy, action and reward function for our work are shown in the top of Figure 4. We use a baseline strategy (Zaremba and Sutskever, 2015; Williams, 1992) to decrease the learning variance. The expected reward for an instance is calculated as follows:

$$L_{RL} = \frac{1}{N} \sum_{i=1}^{N} log(p(y_i)) * w_i * (r(y_i, y^g) - r_b) \quad (3)$$

where y_i is a generated SQL query, y^g is the ground truth, $p(y_i)$ is the probability of y_i being generated by our model. N is the number of sampled SQL queries (we use the top-N generated sequences in the beam with normalized probabilities). $r(y_i, y^g)$ is the reward function, i.e., SEA score in Equation 1. r_b is the baseline reward whose goal is to force the model to generate queries that yield a reward $r > r_b$ and discourage those that have reward $r <$

r_b. $w_i = \frac{p(y_i)}{\sum_{j=1}^{N} p(y_j)}$ represents the weight of the i-th sample.

We follow the method of Ranzato et al. (2015) to pre-train the generator for a few epochs using the cross-entropy loss and then slowly switch to the REINFORCE loss, as described in Algorithm 1. Then, the parameters of decoder are updated with policy gradient, and the parameters of encoder are fixed during training of RL.

Algorithm 1 REINFORCE algorithm

input: Input sequences X; ground truth output sequences Y; and a pre-trained generator G_θ
while *not converged* **do**
  Select a batch of size B from X and Y
  Sample N full sequences from top-N beam of G_θ
  Calculate the loss according to Eq. 3
  Update the parameters of network $\theta \leftarrow \theta + \alpha \bigtriangledown L_{RL}$
end while

4.3 Training Details

Base generator. The training objective is the minimization of the cross-entropy (CE) loss:

$$L_{CE} = -\sum_{t=1}^{T} log\ p_\theta(\hat{y}_t|\hat{y}_{t-1}, s_t^{dec}, kp) \qquad (4)$$

where $\hat{y}_t$ is output at time t, s_t^{dec} is the hidden state at time t, kp is the representation of the input.

The features are used as additional information, through one-hot encoding, with sizes of $\{6, 3, 28\}$ respectively. Their encodings are concatenated with $\{\hat{\mathbf{u}}_i\}_{i=1}^{n}$ in the encoder, or added into s_t^{dec} through attention mechanism in the decoder.

Improved with RL. We combine standard supervised prediction and RL in two ways:

- We pre-train the model with CE loss until the best performance is achieved on dev set. Then we use the REINFORCE algorithm to retrain the model (as described in Algorithm 1).
- We use η to handle the transition from using CE loss to REINFORCE loss, $L_{mixed} = \eta L_{RL} + (1-\eta) L_{CE}$. η is equal to $epoch \times 0.03$ when $epoch \leq 30$ and η is equal 1 when $epoch \geq 30$.

In our experiments, models obtained in the first way perform better.

In Equation 3, we set N as 5, r_b as 1, α as 1 in the first way and $(1-\eta)$ in the second way. During training, $L_{RL} = max(0, L_{RL})$.

5 Experiments

We conduct experiments on the WikiSQL dataset[7]. Following other models, we use three evaluation metrics: logical form accuracy (Acc_{lf}), the percentage of the examples that have exact string match with the ground truth, query match accuracy (Acc_{qm}), calculated ignoring the false negatives due to only the order-matters of SQL subqueries, and execution accuracy (Acc_{ex}), the proportion of correct answers.

5.1 Experimental Setup

Our parameter setting of generator follows Xu et al. (2017). Adam (Kingma and Ba, 2014) with a learning rate of 10^{-3} is used as the optimizer, and the model with the best performance on the dev set is selected for evaluation on the test set. Dropout with rate 0.3 is used during training. Greedy search is used for decoding. Adam with a learning rate of 10^{-4} is used as the optimizer in RL. For both training and decoding we use a K40 GPU. Hidden state sizes for both encoder and decoder are set to 100. The word embeddings are initialized from Glove word embeddings (Pennington et al., 2014) on Common Crawl, and are fixed during training.

5.2 Preprocess

In order to get related cells from KBs to construct the input sequence kp, we use every word of the question and its stem[8] to search over cells of KBs (as a cell or a part of it typically appears in a question (Sun et al., 2018)). In order to improve the recall of cells, we assign number and date into *NUMBER* both in questions and KBs. The recall of cells is about 97%, and accuracy is about 40% (see Appendix B for details).

5.3 Results and Analysis

We compare our model (Pnt-G) against several previously published systems.

- Seq2SQL (Zhong et al., 2017): this method trains two separate classifiers for SELECT aggregator and SELECT column, and utilizes RL for further model training.
- STAMP (Sun et al., 2018): this method proposes an end-to-end learning pointer network

[7] https://github.com/salesforce/WikiSQL. We had to access content of KBs to obtain related cells. Not all content used in spider dataset is accessible, we don't conduct experiments on spider dataset.

[8] http://www.nltk.org

Mothods	Dev			Test		
	$Acc_{lf}(\%)$	$Acc_{qm}(\%)$	$Acc_{ex}(\%)$	$Acc_{lf}(\%)$	$Acc_{qm}(\%)$	$Acc_{ex}(\%)$
Seq2SQL	48.2	–	58.1	47.4	–	57.1
STAMP	61.5	–	74.8	60.7	–	74.4
COARSE2FINE	–	72.9	79.2	–	71.7	78.4
Seq2SQL + RL	49.5 (+1.3)	–	60.8 (+2.7)	48.3 (+0.9)	–	59.4 (+2.3)
STAMP + RL	61.7 (+0.2)	–	75.1 (+0.3)	61.0 (+0.3)	–	74.6 (+0.2)
(Wang et al., 2018b)						
Pointer-SQL + EG (3)	–	66.6 (+4.8)	77.3 (+4.8)	–	66.7 (+4.4)	76.9 (+5)
Pointer-SQL + EG (5)	–	67.5 (+5.7)	78.4 (+5.9)	–	67.9 (+5.6)	78.3 (+6.4)
COARSE2FINE + EG (3)	–	75.6 (+2.7)	83.4 (+4.2)	–	74.8 (+3.1)	83.0 (+4.6)
COARSE2FINE + EG (5)	–	76.0 (+3.1)	84.0 (+4.8)	–	75.4 (+3.7)	83.8 (+5.4)
Pnt-G	66.4	67.5	74.9	66.0	66.9	74.2
+ RL	67.1	68.2	76.0	66.5	67.5	75.1
+ FeaEn	70.0	70.9	77.4	69.4	70.0	75.9
+ FeaEn + RL	72.9 (+6.5)	73.7 (+6.2)	81.5 (+6.6)	72.1 (+6.1)	73.1 (+6.2)	80.3 (+6.1)
+ FeaDe	69.4	70.6	77.0	69.0	69.4	75.3
+ FeaDe + RL	72.6 (+6.2)	73.5 (+6.0)	81.1(+6.2)	71.9(+5.9)	72.9 (+6.0)	80.0(+5.8)

Table 2: Performances of different approaches on the wikiSQL dataset. Three evaluation metrics are logical form accuracy (Acc_{lf}), query match accuracy (Acc_{qm}) and execution accuracy (Acc_{ex}). Our model is **Pnt-G**. Models of the second row are from Zhong et al. (2017); Sun et al. (2018); Dong and Lapata (2018) respectively. Models of the third row utilize the correctness of answer executed by the generated SQL query as the reward. Models of the forth row utilize the execution guidance in decoding, n of EG (n) represents a beam size of n.

with multiple channels, and utilizes column-cell relation to improve the WHERE clause. Meanwhile, it also utilizes RL for further model training.

- COARSE2FINE (Dong and Lapata, 2018): this method trains two separate classifiers for SELECT aggregator and SELECT column, and uses a sketch encoder-decoder model for the WHERE clause with a classifier model for sketch selection.
- +EG (Wang et al., 2018b): this method utilizes execution guidance to detect and exclude faulty subqueries during decoding procedure by conditioning on the execution of partially generated SQL queries, demonstrated on four SOTA models with different approaches.

Pnt-G as the base generator is based on a pointer network, which is similar to STAMP, Seq2SQL and PointerSQL[9] (Wang et al., 2018a). Both STAMP and PointerSQL utilize multiple channels as input, including questions, column names and SQL keywords. The values of conditions are copied from questions without guaranteeing that a multi-word value could be successively generated. Different from them, we firstly obtain the related cells which could be used as conditions' values and utilize them as units of kp. So Pnt-G guarantees integrities of multi-word values and outperforms STAMP and PointerSQL. COARSE2FINE utilizes a fixed form of "select-aggregator-where" to partially account for SQL syntax, and outperforms Pnt-G which is not restricted to any form.

Our model utilizes the SEA score as the reinforced objective to improve the syntax and execution of a SQL query. As shown in Table 2, our model (+ FeaEn + RL) gets the best performance on the test set, improving all metrics by 6%. It already outperforms models using other rewards (mentioned in the second part) and is comparable to models using execution guidance (mentioned in the third part). We report the results of the following ablation models:

- (+ RL) model: the SEA score is as the reinforced objective of RL and is utilized to guide the training of base generator.
- (+ FeaEn) model: features' encoding is used as additional information in the encoder through concatenating it to the final representation of kp.
- (+ FeaEn + RL) model: (+ RL) model and (+ FeaEn) model are combined.
- (+ FeaDe) model: features' encoding is used as additional information in the decoder through attention mechanism.
- (+ FeaDe + RL) model: (+ RL) model and (+ FeaDe) model are combined.

Results of (+RL) model show that the effect of SEA score without additional information is not obvious, as no useful information is obtained for the learning of RL model. Results of (+ FeaEn)

[9]This model is similar to STAMP, without utilizing column-cell relation.

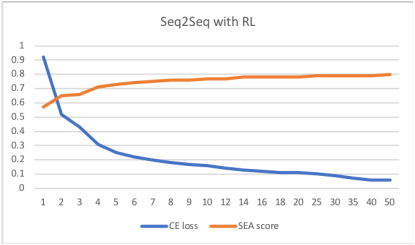

Figure 5: Learning curve during training.

and (+ FeaDe) models show that our proposed features are helpful to improve the generation of SQL queries both in the encoder and decoder. Results of models combining features and RL show that the SEA score effectively learns useful information from features and gives effective feedback to guide the training of the generator. This combination significantly improves the model's performance and maximizes the effects of RL.

5.4 Model Analysis

Following other models, we also report more fine-grained evaluation metrics over constituents of SQL queries (as shown in Table 3). The improvements on the SELECT and WHERE clauses are consistent with our primary intuition on designing syntax- and execution-aware score.

Methods	Acc_{sel}	Acc_{agg}	Acc_{whe}
Seq2SQL (Zhong et al., 2017)	88.9%	90.1%	60.2%
STAMP (Sun et al., 2018)	90.0%	89.9%	76.3%
PntG	92.4%	89.3%	78.4%
PntG + FeaEn + RL	94.6%	90.7%	83.6%

Table 3: Performances of different approaches on WikiSQL test sets. Accuracy (Acc_{lf}) is evaluated on SELECT column (Acc_{sel}), aggregator (Acc_{agg}), and WHERE clause (Acc_{whe}), respectively.

The goal of training with RL is to maximize the expected reward (shown in Equation 3). Details about our RL experiments are shown in Figure 5. We can see that the reward increases during training. The training method of pre-training and retraining (as described in Section 4.3) gets better performance, outperforms models of combine-training by 1.7% on Acc_{qm}.

We conduct a case study to illustrate the results of Pnt-G with the SEA score, with a comparison to Pnt-G (as shown in Figure 6). In the first example, (+ FeaEn + RL) model learns from the KB structures that the cell of "*roman kreuziger*" belongs to the column of "*young rider classification*". In the second example, (+ FeaEn + RL) model satisfies SQL syntax better that it has learned *C_S* with a pattern of "*column operator cell*". In the third example, (+ FeaEn + RL) model exploits KB semantic constraints to exclude the subquery of *max (largest city)* as the data types of *max* and *largest city* don't match.

question	Who led the points classification when Roman Kreuziger led the young rider classification ?
Golden SQL query	SELECT points classification WHERE young rider classification EQL roman kreuziger
SQL query (Pnt-G)	SELECT mountains classification WHERE points classification EQL roman kreuziger
SQL query (+FeaEn +RL)	SELECT points classification WHERE young rider classification EQL roman kreuziger
question	What member of the original Toronto cast played the character played by Constantine Maroulis in the first national tour cast?
Golden SQL query	SELECT original toronto cast WHERE first national tour cast EQL constantine maroulis
SQL query (Pnt-G)	SELECT original toronto cast WHERE first national tour cast EQL second national tour cast
SQL query (+FeaEn +RL)	SELECT original toronto cast WHERE first national tour cast EQL constantine maroulis
question	what is the largest city where the population is 6620100?
Golden SQL query	SELECT largest city WHERE population (2013) EQL 6620100
SQL query (Pnt-G)	SELECT max (largest city) WHERE population (2013) EQL 6620100
SQL query (+FeaEn +RL)	SELECT largest city WHERE population (2013) EQL 6620100

Figure 6: Case study on the test set, Pnt-G (+ FeaEn + RL) model is compared with Pnt-G: each question is followed by the standard SQL query and the generated ones from the two approaches.

5.5 Discussion

Compared with models that restrict SQL queries to fixed forms of "select-aggregator-where", our model is less tailored. Compared with models that utilize the correctness of answers as the reward, our model accounts for SQL syntax and execution at multiple granularities and get better performance. Compared with models that utilizes execution guidance in decoding, our model do not need interact with KBs in the prediction. We believe that it is easy to apply our model to other datasets with the appropriate definition of the SEA score. We need to apply this approach to weakly supervised text-to-SQL tasks to search good candidate semantic parses and process the problem of spurious programs. The SEA score gives full consideration to SQL syntax and KB structures, and ensure the syntax and execution of SQL queries.

6 Conclusion

In this paper, we design a new score for supervised text-to-SQL task, and utilize it as a reinforced objective to guide the generator. The proposed score accounts for SQL syntax, KB structures and semantic constraints. Experimental results show that our model get better results compared with models using other rewards, and get competitive results with models utilizing execution guidance in the prediction. In the future, we would like to apply the framework to weakly supervised text-to-SQL tasks and other logical forms.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Chis J Date. 1994. *A Guide to the SQL Standard: A User's Guide to the Standard Realtional Language SQL*. Addison-Wesley.

Li Dong and Mirella Lapata. 2018. Coarse-to-fine decoding for neural semantic parsing. *arXiv preprint arXiv:1805.04793*.

Daya Guo, Yibo Sun, Duyu Tang, Nan Duan, Ming Zhou, and Jian Yin. 2018. Question generation from sql queries improves neural semantic parsing. *arXiv preprint arXiv:1808.06304*.

Di He, Hanqing Lu, Yingce Xia, Tao Qin, Liwei Wang, and Tie-Yan Liu. 2017. Decoding with value networks for neural machine translation. In *Advances in Neural Information Processing Systems*, pages 178–187.

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer. 2017. Learning a neural semantic parser from user feedback. *arXiv preprint arXiv:1704.08760*.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Jiwei Li, Will Monroe, Alan Ritter, Michel Galley, Jianfeng Gao, and Dan Jurafsky. 2016. Deep reinforcement learning for dialogue generation. *arXiv preprint arXiv:1606.01541*.

Yunyao Li, Huahai Yang, and HV Jagadish. 2006. Constructing a generic natural language interface for an xml database. In *International Conference on Extending Database Technology*, pages 737–754. Springer.

Wang Ling, Edward Grefenstette, Karl Moritz Hermann, Tomáš Kočiskỳ, Andrew Senior, Fumin Wang, and Phil Blunsom. 2016. Latent predictor networks for code generation. *arXiv preprint arXiv:1603.06744*.

Dipendra Misra, Ming-Wei Chang, Xiaodong He, and Wen-tau Yih. 2018. Policy shaping and generalized update equations for semantic parsing from denotations. *arXiv preprint arXiv:1809.01299*.

Romain Paulus, Caiming Xiong, and Richard Socher. 2017. A deep reinforced model for abstractive summarization. *arXiv preprint arXiv:1705.04304*.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. 2015. Sequence level training with recurrent neural networks. *arXiv preprint arXiv:1511.06732*.

Tianze Shi, Kedar Tatwawadi, Kaushik Chakrabarti, Yi Mao, Oleksandr Polozov, and Weizhu Chen. 2018. Incsql: Training incremental text-to-sql parsers with non-deterministic oracles. *arXiv preprint arXiv:1809.05054*.

Yibo Sun, Duyu Tang, Nan Duan, Jianshu Ji, Guihong Cao, Xiaocheng Feng, Bing Qin, Ting Liu, and Ming Zhou. 2018. Semantic parsing with syntax- and table-aware sql generation. *arXiv preprint arXiv:1804.08338*.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer networks. In *Advances in Neural Information Processing Systems*, pages 2692–2700.

Chenglong Wang, Marc Brockschmidt, and Rishabh Singh. 2018a. Pointing out sql queries from text.

Chenglong Wang, Alvin Cheung, and Rastislav Bodik. 2017. Synthesizing highly expressive sql queries from input-output examples. In *ACM SIGPLAN Notices*, volume 52, pages 452–466. ACM.

Chenglong Wang, Kedar Tatwawadi, Marc Brockschmidt, Po-Sen Huang, Yi Mao, Oleksandr Polozov, and Rishabh Singh. 2018b. Robust text-to-sql generation with execution-guided decoding. *arXiv preprint arXiv:1807.03100*.

Li Wang, Junlin Yao, Yunzhe Tao, Li Zhong, Wei Liu, and Qiang Du. 2018c. A reinforced topic-aware convolutional sequence-to-sequence model for abstractive text summarization. *arXiv preprint arXiv:1805.03616*.

David HD Warren and Fernando CN Pereira. 1982. An efficient easily adaptable system for interpreting natural language queries. *Computational Linguistics*, 8(3-4):110–122.

Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256.

Yuk Wah Wong and Raymond Mooney. 2007. Learning synchronous grammars for semantic parsing with lambda calculus. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 960–967.

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

Navid Yaghmazadeh, Yuepeng Wang, Isil Dillig, and Thomas Dillig. 2017. Sqlizer: Query synthesis from natural language. *Proceedings of the ACM on Programming Languages*, 1(OOPSLA):63.

Tao Yu, Zifan Li, Zilin Zhang, Rui Zhang, and Dragomir Radev. 2018a. Typesql: Knowledge-based type-aware neural text-to-sql generation. *arXiv preprint arXiv:1804.09769*.

Tao Yu, Michihiro Yasunaga, Kai Yang, Rui Zhang, Dongxu Wang, Zifan Li, and Dragomir Radev. 2018b. Syntaxsqlnet: Syntax tree networks for complex and cross-domaintext-to-sql task. *arXiv preprint arXiv:1810.05237*.

Wojciech Zaremba and Ilya Sutskever. 2015. Reinforcement learning neural turing machines-revised. *arXiv preprint arXiv:1505.00521*.

John M Zelle and Raymond J Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the national conference on artificial intelligence*, pages 1050–1055.

Luke S Zettlemoyer and Michael Collins. 2012. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. *arXiv preprint arXiv:1207.1420*.

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