Multimedia Tools and Applications (2022) 81:15205–15217
https://doi.org/10.1007/s11042-022-12573-0

Enhance text-to-SQL model performance with information sharing and reweight loss

Chi Wei[1] · Shaobin Huang[1] · Rongsheng Li[1]

Received: 17 March 2021 / Revised: 16 June 2021 / Accepted: 31 January 2022 /
Published online: 28 February 2022

Abstract
The goal of Text-to-SQL task is to map natural language queries into equivalent structured query languages(NL2SQL). On the WikiSQL dataset, the method used by the state-of-the-art models is to decouple the NL2SQL task into subtasks and then build a dedicated decoder for each subtask. There are some problems in this method, such as the model is too complicated, and the ability to learn the dependency between different subtasks is limited. To solve these problems, this paper innovatively introduces the sharing mechanism of multi-task learning into the NL2SQL task and realizes sharing by letting different subtasks share the same decoder. Firstly, sharing decoders for different subtasks can effectively reduce the complexity of the model, and at the same time, allows different subtasks to share knowledge during the training process so that the model can better learn the dependencies between different subtasks. This paper also designed a re-weighted loss to balance the complexity of the SELECT clause and the WHERE clause. We have evaluated the method in this article on the WikiSQL dataset. The experimental results show that the accuracy of the proposed model is better than state-of-the-art on the WikiSQL without execution guided decoding.

Keywords Text-to-SQL · Multi-task learning · Re-weighted loss

✉ Rongsheng Li
dasheng@hrbeu.edu.cn

Chi Wei
2236058667@qq.com

Shaobin Huang
huangshaobin@hrbeu.edu.cn

[1] College of Computer Science and Technology, Harbin Engineering University, Harbin 150001, China

1 Introduction

With the development of digitization, more and more data are stored in the database. These databases contain a lot of knowledge. Therefore, allowing users to interact with the database using natural language is a research goal with notable applicable value [16], and the goal of the Text-to-SQL task is to allow users to interact with the database using natural language. At present, SQL statements that only involve a single table and simple query still play an important role in practical application [9]. Therefore, this paper further researches the NL2SQL model based on the WikiSQL dataset, which only contains single table and simple query.

The WikiSQL dataset released by Zhong et al. in 2017 contains 80,654 natural language questions, which correspond to manually annotated SQL query statements and data tables [20]. The publication of WikiSQL datasets has greatly stimulated people's enthusiasm for the research of NL2SQL. The representative works include the SQLNet [20], TypeSQL [21], Coarse-to-fine [4], SQLova [8], X-SQL [6] and HydraNet [13]. These works decouple the NL2SQL tasks into six subtasks, as shown in Fig. 1. In these systems, dedicated decoder are constructed for each subtask, and the neural network used by these decoders is usually a long short-term memory network(LSTM) [18].

There are two disadvantages to these advanced models. Firstly, six special decoders make the model structure very complicated. The complex model structure makes the model difficult to train and increases training time. Secondly, there are dependencies between different parts of the SQL query. For example, when the element of the column specified by the Select-Column is not a number, the result of Select-Aggregation can only be None or Count. If the element of the column specified by the Select-Column is a number, the result of Select-Aggregation can also be the Max, Min, and Avg. Ma [9] pointed out that these dependencies can be obtained from input natural language queries. Therefore, if each subtask uses a dedicated decoder, the distributed representation of natural language queries received by each subtask is different, which will inevitably make it difficult for such a model to learn the dependencies between different subtasks.

In order to solve the problem of complex model structure and challenging to learn the dependency relationship between different subtasks, this paper proposes a method based on information sharing. Like the current advanced models. This paper uses BERT [10] as the encoder to obtain the distributed representation of the input natural language query and its corresponding table schema. The difference is that this paper no longer constructs a dedicated decoder for each subtask but one decoder shared by all subtasks. This idea comes from multi-tasking [12, 15]. The decoder uses a Bi-LSTM network. The input of the decoder is the output of the pre-training model BERT, and the output of the decoder is the final embedded representation of natural language query and table schema. When predicting different subtasks, a fully connected neural network is used to learn features related to specific subtasks. The

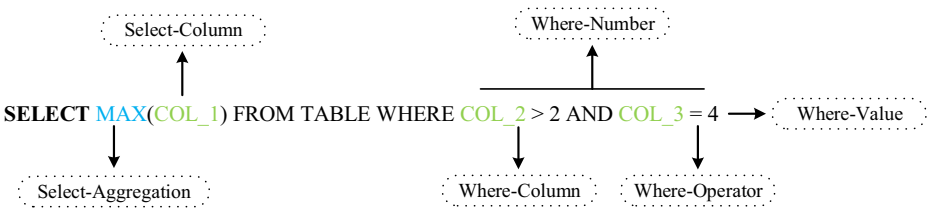

Fig. 1 Divide the SQL into six parts, each part as a subtask

attention mechanism is used to capture information more related to the current task. In addition, on WikiSQL datasets, the average length of the WHERE clause is larger than that of the SELECT clause. To balance the complexity of the SELECT clause and WHERE clause, a weight is reassigned for each subtask loss similar to the method proposed by Qi [17]. Compared with the existing methods, the proposed method reduces the size of the decoder to one-sixth of the original and effectively reduces the complexity of the model. Secondly, all the subtasks use the same decoder in prediction. The natural language input representation is the same, which can make the model better learn the dependency relationship between different subtasks.

We evaluated our model on the WikiSQL and compared it with other advanced models. Compared with state-of-the-art, the evaluation index for execution accuracy has increased by 0.5% in this paper on the testset without execution guided decoding.

2 Related work

The goal of the semantic parsing task is to map natural language to machine-interpretable representation, such as code and SQL [11]. The purpose of this paper is to map the natural language to the equivalent structured query language (NL2SQL), which is a subset of semantic parsing tasks. The early research of NL2SQL mainly used semantic parsing task for reference and used the Seq-to-Seq [19] structure with an attention mechanism to map natural language queries to SQL. However, due to the strong syntax restriction of SQL statements, the Seq-to-Seq structure cannot impose specific syntax restrictions on the output space [3]. Therefore, the Seq-to-Seq structure is not suitable for NL2SQL tasks.

To solve this problem, Zhong et al. Proposed the seq2sql model [4]. The task of generating SQL is decoupled into two parts of developing target SQL query: SELECT clause and WHERE clause, which is equivalent to adding syntax constraints to the output of the model, and successfully improves the execution accuracy of the test set to 59.4%. The Seq2SQL [22] model proves the effectiveness of adding syntax constraints when generating SQL, but the rules are still too simple. Therefore, the SQLNet [20] model proposed by Seq2SQL further decomposes the NL2SQL task into six subtasks: Select-Column, Select-Aggregation, Where-Number, Where-Column, Where-Operator, and Where-Value. This way is equivalent to adding more substantial syntax constraints to the output of the model. The later SQLova [8] and X-SQL [6] models also use the decoding method of the SQLNet and have achieved better performance than human beings on the WikiSQL dataset. These works prove that it is imperative and necessary to consider SQL syntax rules in decoding.

However, this method of introducing syntax rules by decomposing NL2SQL tasks into different subtasks will inevitably lead to the independence of each subtask, which makes it difficult for the model to learn the dependency relationship between different subtasks. In the face of this problem, the existing advanced models will determine the prediction order of each subtask according to the dependency relationship between subtasks. First, predict the dependent subtask, and then use the result as the input of the module that depends on the subtask. Although it can help the model learn the dependencies among tasks to a certain extent, it needs to design the prediction order artificially. The ability to capture the dependencies among subtasks is limited. In addition, the X-SQL model uses a fully connected neural network and attention mechanism to replace LSTM as the decoder of the model, which can effectively reduce the amount of calculation and improve the effectiveness of the model. X-SQL shows

that the WikiSQL task does not require an overly complicated decoder. The NL2SQL task is decomposed into six subtasks, and then a select decoder is designed for each subtask, which will inevitably lead to the complexity of the model structure.

In terms of input, TypeSQL helps the model understand natural language input by adding additional type information to the words in the input natural language questions [21]. Guo automatically labels the natural language input and columns using the string matching method [5]. The way of Guo was also equivalent to adding some prior knowledge to the model. Although it can improve the performance of the model, it needs additional human work. The work of Xu [6] and Hwang [8] shows that the more powerful pre-training word vector [10, 14] can effectively improve the performance of the model. Bogin's work shows that a graph neural network is very suitable for embedding table schema. Nodes in graph neural networks can be used to represent tables and columns, and edges can be used to describe the relationship between tables and columns [1, 2]. However, graph neural network is mainly used in NL2SQL tasks involving multi tables and complex queries.

3 Methodology

The model of this paper refers to the works of Guo [5]. Guo designed a dedicated decoder for each subtask, illustrated in Fig. 2(a). Different from the works of Guo, we design a shared decoder for subtasks and then use different output layers to predict the output of each subtask, illustrated in Fig. 2(b). The shared decoder makes the model easier for the model to capture the association between the subtasks from the natural language. Simultaneously, the model can use the dependence between the various subtasks to improve the generalization ability.

3.1 Encoder layer

This paper uses BERT as the encoder. The input of the encoder consists of three parts:

Natural language embedding: Use the vocabulary provided by BERT to get the one-hot code of each input word. Then do the dot product with a trainable matrix **W** to get the initial representation $\boldsymbol{q}_i$ of each word.

Position embedding: Since BERT is an attention-based model, it cannot capture the positional relationship between words like LSTM. So BERT needs position embedding of each input word so that the model can learn position information between words.

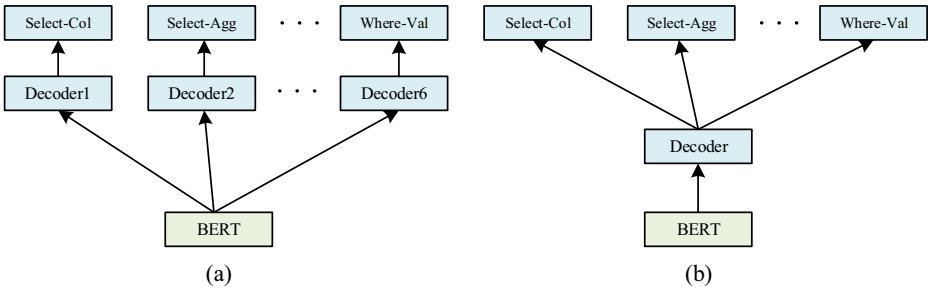

Fig. 2 (**a**) Independent decoder; (**b**) Shared decoder

Segment embedding: The input of the encoder should include the table mode and the natural language question. For the model to distinguish whether the input comes from the question sentence or the table mode, each word adds a segment embedding.

3.2 Decoder layer

As shown in Fig. 3, $\boldsymbol{h}_{[CLS]}$, $\boldsymbol{h}_1, \boldsymbol{h}_2, \boldsymbol{h}_3, \cdots, \boldsymbol{h}_{[SEP]}, \boldsymbol{h}_{C_{11}}, \boldsymbol{h}_{C_{12}}$, and $\mathbf{h}_{C_{21}}$ is the output of the encoder. The representation of the question is $\boldsymbol{h}_i$, the representation of the column is $\boldsymbol{h}_{C_{ij}}$, and the dimension is d. The goal of the decoder layer is to acquire two parts of the input of the NL2SQL task: natural language query and database table schema representation. This paper is using LSTM as the decoder.

LSTM is an improved version of the recurrent neural network (RNN) network, which was first proposed by Hochreiter [7]. It can fully capture the sequence information of sequence data without causing the problem of gradient disappearance, so this article uses it to learn the sequence information between question words. The structure of LSTM can be formalized using **Formulas** (1) through (6):

$$\boldsymbol{f}_t = \sigma\left(\boldsymbol{x}_t \cdot \boldsymbol{\omega}_{xh}^{f} + \boldsymbol{h}_{t-1} \cdot \boldsymbol{\omega}_{hh'}^{f} + \boldsymbol{b}_h^{f}\right) \tag{1}$$

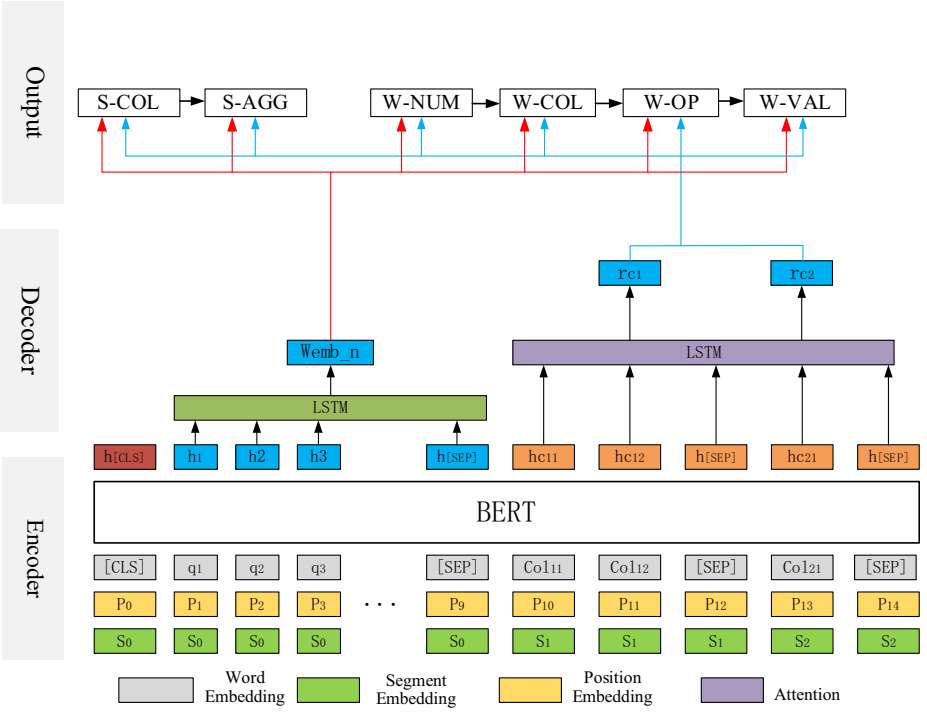

Fig. 3 Overview of our model. Encoder encodes natural language and column into dense vectors. Decoder parses the output of the encoder into two parts of the input of the NL2SQL task: natural language query Wemb_n and database table schema representation r_{c_i}. Output layer generates the final SQL according to Wemb_n and r_{c_i}

$$\boldsymbol{i}_t = \sigma\left(\boldsymbol{x}_t \cdot \boldsymbol{\omega}^{i}_{xh} + \boldsymbol{h}_{t-1} \cdot \boldsymbol{\omega}^{c}_{hh'} + \boldsymbol{b}^{i}_{h}\right) \tag{2}$$

$$\boldsymbol{c}^{'}_t = \tanh\left(\boldsymbol{x}_t \cdot \boldsymbol{\omega}^{c}_{xh} + \boldsymbol{h}_{t-1} \cdot \boldsymbol{\omega}^{c}_{hh'} + \boldsymbol{b}^{c}_{h}\right) \tag{3}$$

$$\boldsymbol{c}_t = \boldsymbol{i}_t \otimes \boldsymbol{c}^{'}_t + \boldsymbol{f}_t \otimes \boldsymbol{c}_{t-1} \tag{4}$$

$$\boldsymbol{o}_t = \sigma\left(\boldsymbol{x}_t \cdot \boldsymbol{\omega}^{o}_{xh} + \boldsymbol{h}_{t-1} \cdot \boldsymbol{\omega}^{o}_{hh'} + \boldsymbol{b}^{o}_{h}\right) \tag{5}$$

$$\boldsymbol{h}_t = \boldsymbol{o}_t \otimes \mathbf{tanh}(\boldsymbol{c}_t) \tag{6}$$

Where σ indicates sigmoid activation function, and tanh indicates hyperbolic tangent activation function. $\boldsymbol{f}_t$, $\boldsymbol{i}_t$, $\boldsymbol{o}_t$ represent the three states of the network at timestep t, which are called forget gate, input gate, and output gate. $\boldsymbol{\omega}$ and $\boldsymbol{b}$ represent the learnable weight parameters and bias parameters of the network. $\boldsymbol{c}^{'}_t$ combines the three types of information $\boldsymbol{f}_t$, $\boldsymbol{i}_t$ and $\boldsymbol{o}_t$, and passes down with timestep t, which represents the information flowing. $\boldsymbol{h}_t$ represents the hidden layer feature at timestep t of the network, and also represents the output value at that moment.

3.3 Output layer

The output layer generates the final SQL according to the Wemb _ n and r_{C_i}. Like Xu [6] and Hwang [8], the NL2SQL task is decoupled into six subtasks, and each subtask predicts a part of the SQL statement. Unlike them, the six subtasks in our model use the same natural language query and database table schema representation, which can better learn the dependencies between subtasks.

Select-Column task (S-COL) predicts the column of the SELECT clause, and Select-Aggregation task (S-AGG) predicts the aggregate function of the column. The probability of S-COL is computed as

$$P^{S-COL}(C_i) = \mathrm{Softmax}\left(\mathbf{W}^{S-COL}([\boldsymbol{r}_{C_i}, \mathbf{Wemb\_n}])\right) \tag{7}$$

Where $\mathbf{W}^{\mathrm{S-COL}} \in R^{1 \times d}$. Before the Select-Aggregate task(S-AGG), the context semantic vector is re-weighted according to the result of S-COL so that the information related to the S-AGG task has more weight. When S-COL selects column i,

$$\widehat{\mathbf{W}}\mathbf{emb\_n}_1 = \sigma\left(\mathbf{W}^{A_i}[\mathbf{Wemb\_n}, \mathbf{r}_{\mathrm{C_i}}]\right) \cdot \mathbf{Wemb\_n} \tag{8}$$

Where $\mathbf{W}^{A_i} \in R^{d \times 2d}$, the probability of aggregator is computed as

$$P^{\mathrm{S-AGG}}(\mathrm{A}_j|\mathrm{C}_i) = Softmax\left(\mathbf{W}^{\mathrm{S-AGG}}[:\mathrm{j}]\left(\left[\mathbf{r}_{\mathrm{C_i}}, \widehat{\mathbf{W}}\mathbf{emb\_n}_1\right]\right)\right) \tag{9}$$

Where $\mathbf{W}^{\mathrm{S-AGG}} \in R^{6 \times d}$ with six being the number of aggregators.

The remaining four tasks W-NUM, W-COL, W-OP, and W-VAL together determine the WHERE clause. The Where-Number task (W-NUM) predicts the number of columns in the WHERE clause. This paper assumes that the WHERE clause contains only four columns at most. So W-NUM is a five-category task.

$$P^{\mathrm{W-NUM}}(n) = Softmax\left(\mathbf{W}^{\mathrm{W-NUM}}\mathbf{Wemb\_n}\right), n = 1, 2, 3, 4 \tag{10}$$

Where $\mathbf{W}^{\mathrm{W-NUM}} \in R^{4 \times d}$. The Where-Column (W-COL) task predicts the columns of the WHERE clause. The probability of W-COL is computed as:

$$P^{\mathrm{W-COL}}(\mathrm{C}_i) = Softmax\left(\mathbf{W}^{\mathrm{W-COL}}([\mathbf{r}_{\mathrm{C_i}}, \mathbf{Wemb\_n}])\right) \tag{11}$$

Where $\mathbf{W}^{\mathrm{W-COL}} \in R^{1 \times d}$. According to the result of W-NUM, the top-n probability columns are selected as the columns of WHERE clause. Before the Where-Operator task(W-OP), the context semantic vector is re-weighted according to the result of W-COL, so that the information related to W-OP has more weight. Assume W-COL selects column i

$$\widehat{\mathbf{W}}\mathbf{emb\_n}_2 = \sigma\left(\mathbf{W}^{O_i}[\mathbf{Wemb\_n}, \mathbf{r}_{\mathrm{C_i}}]\right)\cdot\mathbf{Wemb\_n} \tag{12}$$

Where $\mathbf{W}^{O_i} \in R^{d \times 2d}$. The probability of W-OP is computed as

$$P^{\mathrm{W-OP}}\left(O_j|\mathrm{C}_i\right) = Softmax\left(\mathbf{W}^{\mathrm{W-OP}}[:\mathrm{j}]\left(\left[\mathbf{r}_{\mathrm{C_i}}, \widehat{\mathbf{W}}\mathbf{emb\_n}_2\right]\right)\right) \tag{13}$$

Where $\mathbf{W}^{\mathrm{W-OP}} \in R^{3 \times d}$ with four being the number of operators.

The Where-Value (W-VAL) predicted the value of the WHERE clause. In this paper, the W-VAL is predicting a subset of questions, which is simplified to predict the starting position of the subset:

$$P^{\mathrm{W-VAL}}_{start}\left(q_j|\mathrm{C}_i\right) = Softmax\left(g\left(\mathbf{U}^{start}\boldsymbol{h}_{q_j} + \mathbf{V}^{start}\left(\left[\mathbf{r}_{\mathrm{C_i}}, \widehat{\mathbf{W}}\mathbf{emb\_n}_2\right]\right)\right)\right) \tag{14}$$

And the end position

$$P^{\mathrm{W-VAL}}_{end}\left(q_j|\mathrm{C}_i\right) = Softmax\left(g\left(\mathbf{U}^{end}\boldsymbol{h}_{q_j} + \mathbf{V}^{end}\left(\left[\mathbf{r}_{\mathrm{C_i}}, \widehat{\mathbf{W}}\mathbf{emb\_n}_2\right]\right)\right)\right) \tag{15}$$

Where $g(\boldsymbol{x}) = \mathbf{W}\boldsymbol{x} + \boldsymbol{b}$ and $\mathbf{U}^{start}$、$\mathbf{V}^{start}$、$\mathbf{U}^{end}$、$\mathbf{V}^{end} \in R^{m \times d}$.

3.4 Reweight loss

The loss function of multi-task learning is the weighted sum of the loss function of each task

$$\mathrm{L} = \sum_{\mathrm{i}=1}^{\mathrm{n}} \alpha_{\mathrm{i}}\mathrm{L}_{\mathrm{i}} \tag{16}$$

Where $L_i$represents the loss of the i-th subtask, and α_i represents the weight of the subtask. Generally, the importance of all subtasks is the same, so $\alpha = 1$. This average weighting method has the advantages of simple structure, easy implementation, and low computation. But in this task, because the average length of the SELECT clause and WHERE clause is different, the training difficulty is different. However, the average weighting method always keeps the same weight for various subtasks in the training phase, so the average weighting method cannot balance the training difficulty of the two clauses. Therefore, we hope that the weight of

subtasks is no longer equal, to help the network maintain the balance of contribution of each subtask, to ensure the quality of the final results. In a word, this paper proposes the following weighting method

$$L/6 = 0.1292*(L_{sc} + L_{sa}) + 0.1854*(L_{wn} + L_{wc} + L_{wo} + L_{wv}) \tag{17}$$

The coefficients in this formula are based on the statistics of the WikiSQL dataset. Specifically, in the training set of the WikiSQL, the average length of the WHERE clause is 1.3615, and the average length of the SELECT clause is 1. So the weight vector of subtask is **a** = [1, 1, 1.3615,1.3615,1.3615,1.3615,1.3615], and then normalize it to get the final loss weight of subtask.

4 Experiment

4.1 Dataset

This paper uses the WikiSQL [22] dataset to train and test the model, which is currently the most extensive labeled NL2SQL task dataset. Like the existing method, we use the default split method of the WikiSQL to split the dataset into the train set, validation set, and test set. The training set contains 56,356 pieces of data, the validation set contains 8422 pieces of data, and the test set contains 15,879 pieces of data. In order to ensure the validity of the test set, the test set of the WikiSQL does not include any tables that have appeared in the training set and validation set.

4.2 Evaluation

The logical form accuracy Acc_{lf} is used as an optimization goal during the training period.

$$\mathrm{Acc}_{lf} = \frac{N_{lf}}{N} \tag{18}$$

Where N_{lf} is the number of queries has exact string match with the ground truth query used to collect the para-phrase.

Execution accuracy Acc_{ex} is also a common index to evaluate the performance of the NL2SQL model.

$$\mathrm{Acc}_{ex} = \frac{N_{ex}}{N} \tag{19}$$

Where N_{ex} is the number of queries that, when executed, result in the correct result.

For S-COL, S-AGG, W-NUM, W-OP, and W-VAL, the cross-entropy loss function is used as the loss function. The W-COL task uses KL divergence as the loss function. Because W-COL may contain multiple columns and the order of the columns will not affect the accuracy of the results, but the cross-entropy loss function is sensitive to the order.

4.3 Optimization

The optimization used in this paper is Adaptive Moment Estimation (Adam). This method can be seen as a combination of the momentum method and RMSprop. It not only uses momentum

as the parameter update direction but also adjusts the learning rate adaptively. The parameter update of the Adam is computed as

$$\Delta\theta_t = -\frac{\alpha}{\sqrt{\widehat{G}_t + \varepsilon}}\widehat{M}_t \tag{20}$$

Where $\widehat{M}_t = \frac{M_t}{1-\beta_1^t}$, $\widehat{G}_t = \frac{G_t}{1-\beta_2^t}$, $M_t = \beta_1 M_{t-1} + (1-\beta_1)g_t$, and $G_t = \beta_2 G_{t-1} + (1-\beta_2)g_t \cdot g_t$. g_t is the gradient of parameter θ at the current time step t. β_1 and β_2 are the attenuation rates of the two moving averages, and are usually taken as $\beta_1 = 0.9$, $\beta_2 = 0.99$.

4.4 Experimental setup

This paper uses BERT-base as the encoder. In particular, we use the trained BERT-base model parameters submitted by Guo [5] and others to initialize our BERT-base model. The hidden layer dimension of $LSTM_n$ are 300 dimensions, and the hidden layer dimension of $LSTM_h$ are 100 dimensions. Use the Adam algorithm to train the model. We set the batch size to 32 and the epochs to 200. Using BERT fine-tune, the learning rate is 10^{-5}, and the learning rate of the acquisition layer and the output layer is 0.0005. Use drop-out regularization and set the drop-out value to 0.2.

The experiment in this paper is based on Windows 10 Operation System and RTX3090; the programming language is python3.7, and the deep learning framework used is Pytorch1.4.

4.5 Result

4.5.1 Test accuracy

We compared our method with the advanced method of NL2SQL based on the task decoupling method on the WikiSQL dataset. These works include Seq2SQL [22], SQLNet [20], TypeSQL [21], SQLova [8], X-SQL [6], HydraNet [13] and Guo [5]. On the test set without executive guidance (EG), the execution accuracy of the method based on BERT-base in this paper is better than other methods, including SQLova using BERT-Large and X-SQL model based on MT-DNN. The pre-training models BERT-Large and MT-DNN are better than that of BERT-base, but even though the performance of the pre-training model has disadvantages, our model is very competitive (Table 1).

In order to understand and analyze the performance of the model in more detail, Table 2 shows the test accuracy of each model on the six subtasks. Table 2 shows that the test accuracy of the subtasks of the model in this paper is not outstanding compared with the existing models, which means that having all subtasks share one encoder will weaken the expressive ability of the model.

Since subtasks with incorrect predictions will appear in different SQL query predictions, the accuracy of overall test execution is lower than that of subtasks, and the greater this difference between the two execution accuracy, the smaller correlation between the various subtasks. The experimental results show that the subtasks test execution accuracy of the model in this paper is not outstanding compared with the existing model, but the overall test accuracy is better than the existing models, which proves that our model has

Table 1 The accuracy of models on the verification/test set

Model	Dev		Test	
	Acc_{LF}	Acc_{EX}	Acc_{LF}	Acc_{EX}
Seq2SQL	49.5	60.8	48.3	59.4
SQLNet	63.2	69.8	61.3	68.0
TypeSQL	68.0	74.5	66.7	73.5
SQLova	81.6	87.2	80.7	86.2
X-SQL	83.8	89.5	83.3	88.7
HydraNet	83.6	89.1	83.8	89.2
Guo	84.3	90.1	83.7	89.2
This work	**84.4**	90.1	83.6	89.5
This work-RLC	84.3	**90.4**	**83.9**	**89.7**

better performance in learning the dependence between subtasks. Further, this result shows that the idea of sharing decoders proposed in this paper is effective, and it is helpful for the model to learn the dependencies between different subtasks.

In addition, for the result with re-weight loss, the test accuracy of the four subtasks related to the WHERE clause has been improved. The overall test execution accuracy and logical form accuracy have also been improved. The experimental result shows that the re-weighting loss proposed in this paper is useful and can help the model effectively balance the complexity of the WHERE clause and the SELECT clause, thereby improving the performance of the model.

The most important thing is that the value of this paper is more about the idea of letting different subtasks share the same decoder and the application of re-weighting losses, not just higher accuracy.

4.5.2 Train time

This paper allows six subtasks to share the same decoder based on the sharing mechanism, so the decoder layer's complexity is effectively reduced. In order to measure the influence of the shared decoder on the complexity of the model decoder, we compared the training time of the model with and without the shared decoder. Except for whether to share decoder, the hyperparameters and experimental settings are the same. We take the training time of the first 20 rounds, repeat ten times, and take the average of ten experiments for comparison. The result is shown in Fig.4.

Table 2 Test accuracy of the subtasks

Model	S_{col}	S_{agg}	W_{no}	W_{col}	W_{op}	W_{val}
SQLova	96.8	90.6	98.5	94.3	97.3	95.4
X-SQL	97.2	91.1	98.6	95.4	97.6	96.6
HydraNet	**97.6**	**91.4**	98.4	95.3	97.4	96.1
Guo	97.4	90.0	**99.1**	**97.9**	**98.1**	**97.6**
This work	96.7	90.5	98.1	96.6	96.9	96.3
This work-RLC	96.7	90.5	98.2	96.8	97.2	96.6

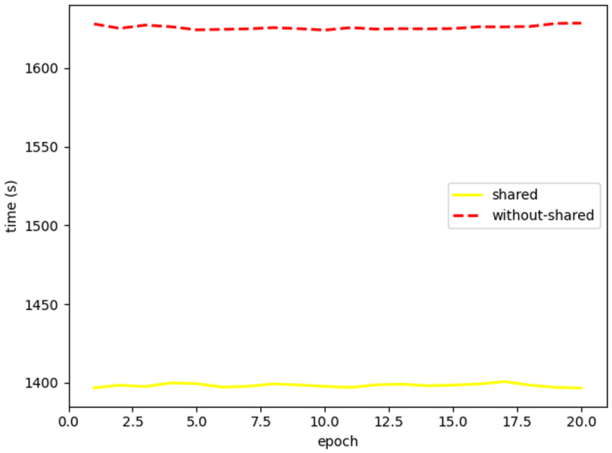

Fig. 4 Comparison of training time between shared decoder model and without-shared decoder model

Average the training time of the first 20 rounds. The training time of the model with the shared decoder is 1398.32 s, the training time of the model without the shared decoder is 1625.69 s, and the training time is shortened by 13.99%. The experimental result concluded that the sharing mechanism effectively reduces the complexity of the model.

4.5.3 Case analyse

In this section, we use the model proposed by Guo [5] as the baseline for case analysis. We found a total of 198 cases in which the baseline prediction was incorrect, and this paper prediction was correct on the WikiSQL testset. The number of mistakes on each subtask is shown in Table 3:

Table 4 shows three NL2SQL cases in which the baseline prediction was incorrect, but this paper prediction was correct on the WikiSQL test set. In the first example, the baseline incorrectly predicted the S-AGG result as Count. The baseline model obviously failed to understand the semantics of the natural language. It is found that in the second example, the W-VAL prediction error of the baseline is caused by the fact that when "something in something" appears in the query, the where clause is usually affected, resulting in overfitting of the baseline. In the Third example, since there is no explicit character in the header "%" in the data table, the model must understand natural language to predict accurate SQL statements. In summary, these examples show that the work of this paper can improve the model's ability to understand natural language, better capture the dependencies between subtasks, and prevent the model from overfitting.

Table 3 The number of mistakes on each subtask

	S_{col}	S_{agg}	W_{no}	W_{col}	W_{op}	W_{val}	Total
Num	8	30	2	41	3	39	198

Table 4 Cases

Type	Description
NL	What is the number of s sikh where 955 is the number of buddhists?
Ground	**Select** s Sikh **Where** Buddhist=995
Baseline	**Select** Count(s Sikh) **Where** Buddhist=995
This paper	**Select** s Sikh **Where** Buddhist=995
NL	If % lunsford is 51.82% what is the % mcconnell in Letcher?
Ground	**Select** % McConnell **Where** % Lunsford=51.82%
Baseline	**Select** % McConnell **Where** % Lunsford=51.82% and County=Letcher
This paper	**Select** % McConnell **Where** % Lunsford=51.82%
NL	The man who received 87,676 votes in Queens won what percentage of the total for the election?
Ground	**Select** % **Where** Queens=87,676
Baseline	**Select** Total **Where** Queens=87,676
This paper	**Select** % **Where** Queens=87,676

5 Conclusion and future work

This paper innovatively regards the Text-to-SQL task on the WikiSQL dataset as a multi-task learning task. Based on the sharing mechanism of multi-task learning, this paper simplifies the decoder layer of the existing method, which improves the efficiency of model training and enhances the model's ability to capture dependencies between different subtasks, thereby improving improved model performance. In order to solve the problem of varying complexity between the SELECT clause and the WHERE clause of the SQL statement involved in the WikiSQL dataset, this paper proposes a method of re-weighting the loss, which further improves the performance of the model.

However, there are still some problems with this paper. The method of sharing the decoder weakens the expressive ability of the model. At the same time, the technique of re-weighting the loss proposed in this paper is relatively simple and static. So in the future, we will try to improve the expressive ability of the model and find a more effective way to weight the loss.

References

1. Bogin B, Berant J, Gardner M (2019) Representing Schema Structure with Graph Neural Networks for Text-to-SQL Parsing[C]//Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics. 4560–4565
2. Bogin B, Gardner M, Berant J (2019) Global Reasoning over Database Structures for Text-to-SQL Parsing[C]. Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP). 3650–3655.
3. Dong L, Lapata M (2016) Language to Logical Form with Neural Attention[C]//Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). 33–43.
4. Dong L, Lapata M (2018) Coarse-to-Fine Decoding for Neural Semantic Parsing[C]//Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). : 731–742.
5. Guo T and Gao H (2019) Content Enhanced BERT-based Text-to-SQL Generation. arXiv preprint arXiv: 1910.07179
6. He P, Mao Y, Chakrabarti K, et al. (2019) X-SQL: reinforce schema representation with context[J]. arXiv e-prints, arXiv: 1908.08113
7. Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Comput 9(8):1735–1780

8. Hwang W et al. (2019) A comprehensive exploration on wikisql with table-aware word contextualization. arXiv preprint arXiv:1902.01069,
9. Jianqiang MA et al. (2020) Mention Extraction and Linking for SQL Query Generation. In: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP). 6936–6942.
10. Kenton JDMWC, Toutanova LK (2019) BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding[C]//Proceedings of NAACL-HLT. 4171–4186.
11. Krishnamurthy J, Dasigi P, and Gardner M (2017) Neural semantic parsing with type constraints for semi-structured tables. in Proceedings of the 2017 Conf Empir Method Nat Language Process .
12. Liu X, He P, Chen W, et al. (2019) Multi-Task Deep Neural Networks for Natural Language Understanding[C]//Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics. : 4487–4496.
13. Lyu Q et al. (2020) Hybrid ranking network for text-to-sql. arXiv preprint arXiv:2008.04759
14. Pennington J, Socher R and Manning CD (2014) Glove: Global vectors for word representation. in Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)
15. Pires T, Schlinger E, Garrette D (2019) How Multilingual is Multilingual BERT?[C]//Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics. : 4996–5001.
16. Popescu A-M, Etzioni O, and Kautz H (2003) Towards a theory of natural language interfaces to databases. in Proceedings of the 8th international conference on Intelligent user interfaces
17. Qi K, et al. (2020) Multi-task MR Imaging with Iterative Teacher Forcing and Re-weighted Deep Learning. arXiv preprint arXiv:2011.13614
18. Sundermeyer M, Schlüter R and Ney H (2012) LSTM neural networks for language modeling. in Thirteenth annual conference of the international speech communication association
19. Sutskever I, Vinyals O, Le QV (2014) Sequence to sequence learning with neural networks. Adv Neural Inf Proces Syst 27:3104–3112
20. Xu X, Liu C, Song D (2018) SQLNet: Generating Structured Queries From Natural Language Without Reinforcement Learning
21. Yu T et al (2018) TypeSQL: Knowledge-based type-aware neural text-to-SQL generation. In: 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL HLT 2018. Assoc Comput Linguistics (ACL):588–594
22. Zhong V, Xiong C, Socher R (2018) Seq2SQL: generating structured queries from natural language using reinforcement learning[J]