CFGNN: Cross Flow Graph Neural Networks for Question Answering on Complex Tables

Xuanyu Zhang
School of Artificial Intelligence
Beijing Normal University, Beijing, China
xyz@mail.bnu.edu.cn

Abstract

Question answering on complex tables is a challenging task for machines. In the Spider, a large-scale complex table dataset, relationships between tables and columns can be easily modeled as graph. But most of graph neural networks (GNNs) ignore the relationship of sibling nodes and use summation as aggregation function to model the relationship of parent-child nodes. It may cause nodes with less degrees, like column nodes in schema graph, to obtain little information. And the context information is important for natural language. To leverage more context information flow comprehensively, we propose novel cross flow graph neural networks in this paper. The information flows of parent-child and sibling nodes cross with history states between different layers. Besides, we use hierarchical encoding layer to obtain contextualized representation in tables. Experiments on the Spider show that our approach achieves substantial performance improvement comparing with previous GNN models and their variants.

1 Introduction

Table based question answering (TB-QA) is an important branch of question answering on structured data. It is a challenging task for machines to comprehend given tables and answer corresponding questions. It can help people obtain information easily without understanding complex logic forms. Different from machine reading comprehension (MRC), the answer in TB-QA may be some statistical figures, which can not be obtained directly from the origin text. Based on this, semantic parsing (Zelle and Mooney 1996) is used for this task. Once a natural language question is mapped to a SQL query by semantic parsing, the answer can be obtained indirectly by executing SQL queries on a database engine.

Recently, with the rapid development of deep learning, sequence-to-sequence (Cho et al. 2014) architectures are used by many models (Yu et al. 2018a; Dong and Lapata 2018; Shi et al. 2018) for neural semantic parsing and achieve state-of-the-art results on some datasets. However, previous datasets, such as WikiSQL (Zhong, Xiong, and Socher 2018), usually contain one table in one database. It is still difficult for models to comprehend complex relations between tables. Fortunately, Yu et al. (2018c) have developed the Spider, a large-scale cross-domain text-to-SQL dataset with complex tables. Here is an example from the Spider in Figure 1. Considering the structure of the database schema, graph neural network (GNN) (Scarselli et al. 2009) is used by Bogin, Berant, and Gardner (2019) to obtain the representation of components of tables. The names of tables and columns in the database are nodes. And the relationships of table-column and primary-foreign key are edges in the schema graph.

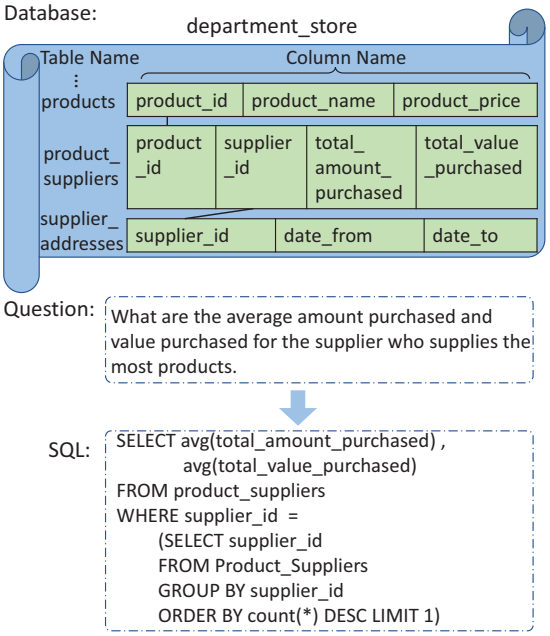

Figure 1: An example from the Spider dataset.

However, most of traditional GNNs (Li et al. 2016; Kipf and Welling 2017; Hamilton, Ying, and Leskovec 2017; Velickovic et al. 2018) usually use summation as the function of neighborhood aggregation. Although it can reflect the relationship between parent and child nodes, the relation-

ship of sibling (child-child) nodes is ignored. Especially, the trade-off between parent-child and child-child relationship received little attention. Moreover, when the nodes have less degrees but more sibling nodes, it can only get very little information by previous GNN. For example, the node of column in the database schema graph may not obtain enough updates in traditional GNNs, because it usually has one parent node and no children nodes. And in the task of natural language processing (NLP), these relationships are all important context information. To understand the complex structure of graph like the schema of databases, humans usually consider the indirect relationship besides the relationship of adjacency and affiliation, then think and inference it again and again.

Considering the issues of previous GNNs, we propose novel **C**ross **F**low **G**raph **N**eural **N**etworks (CFGNN) for question answering on complex tables inspired by the cognitive process of human beings. *Cross Flow* means one node in the graph may obtain different information flows, i.e, the flow of sibling nodes and parent-child nodes in the same layer, and the reasoning flow between different layers of GNNs. For one thing, we first focus on the relationship of sibling nodes besides traditional parent-child relationship in the graph. It can help the nodes with less degrees but more siblings obtain more information flow. And in this task, our approach can also effectively model the node of column in and across tables and view them as context. For another thing, we use two different recurrent neural networks (RNNs) with attention mechanism to integrate these cross flows. One of RNNs is used as the aggregation function among children nodes of the same parent node rather than parent-child nodes. The other is used for aggregating flows and reasoning between layers. And attention mechanism is also used for supplement of the relationship between parent and child nodes. In addition, to integrate the pre-trained contextualized model, BERT (Devlin et al. 2019), with scattered phrase in multiple tables of databases, we use hierarchical encoding layer to obtain the representation of questions, databases, tables and columns from coarse to fine. Experiments on the Spider show that our approach achieves competitive results and outperforms most kinds of GNNs.

2 Related Work

2.1 Graph Neural Networks

Recently, graph neural networks (GNNs) (Scarselli et al. 2009) attract more research interests. Given the representation of the nodes and their adjacencies, GNNs can generate new representation for these nodes after reasoning. Many variants of GNNs are proposed to improve different components. Li et al. (2016) proposed gated graph neural networks (GGNN) to use RNNs for updating the states of nodes. Referring to convolutional neural network, graph convolutional neural networks (GCN) is proposed by Kipf and Welling (2017). And to consider importance by attention mechanism, graph attention networks (GAT) (Velickovic et al. 2018) is proposed. Though effective, the aggregation functions in these GNNs are usually used to aggregate children nodes to parent nodes by summation, which ignores the relationship among sibling nodes. And our proposed CFGNN can obtain the information flow of child-child nodes by RNN besides that of parent-child nodes by attention mechanism. Different from previous RNNs in GNN models (Li et al. 2016; Hamilton, Ying, and Leskovec 2017), we use padding technique for non-existent nodes to learn the difference between the original BERT sequence and each sibling RNN sequence, which uses the identical order and the same length for nodes. And CFGNN still can integrate and leverage these flows between layers by another RNN.

2.2 Semantic Parsing

Semantic parsing (Zelle and Mooney 1996) aims to map natural language to another intermediate representation, such as logic forms and code of program. By executing these scripts, we can obtain the final answer. Traditional methods tend to use lexicalized grammar formalisms, such as combinatory categorial grammars (CCG) (Berant et al. 2013). Recently, natural semantic parsing is used to generate logic forms. Based on sequence-to-sequence (Sutskever, Vinyals, and Le 2014) model, sequence-to-tree (Dong and Lapata 2016) and sequence-to-action (Chen, Sun, and Han 2018) are proposed for semantic parsing. And viewed from the decoding process of sequence-to-sequence, one is token-level decoding (Dong and Lapata 2018), the other is grammar-level decoding (Yin and Neubig 2017). We use the grammar-level decoder in our model.

Semantic parsing is also a universal method for TB-QA and KB-QA. TB-QA also can be viewed as text-to-SQL task or natural language interface for databases. Besides models mentioned above, lots of approaches (Yu et al. 2018a; 2018b; Shi et al. 2018; Hwang et al. 2019) are proposed according to the characteristics of SQL queries. However, these models pay more attention on how to parse natural languages or generate logic forms. And for KB-QA or TB-QA, reasoning on databases or knowledge graphs is also important. So we mainly focus on the core reasoning component, which also can be viewed as the enhancement for the encoder, rather than decoder for semantic parsing.

2.3 Pre-trained Contextualized Model

Recently, pre-trained contextualized models has shown to be effective for representation of words, such as ELMo (Peters et al. 2018), GPT (Radford et al. 2019), BERT (Devlin et al. 2019), MT-DNN (Liu et al. 2019) and so on. Take BERT for example. It is designed to pre-train deep bidirectional representations by jointly conditioning on both left and right context in all layers. It brings a great improvement in natural language tasks, such as language inference and question answering. Different from the previous work on integrating BERT for encoding text (Zhu, Zeng, and Huang 2018; Zhang 2019) or short terms (Hwang et al. 2019; He et al. 2019; Kitaev, Cao, and Klein 2019), such as table name and column name, we design hierarchical encoding layer to obtain different levels of information in databases from coarse to fine, which consists of RNNs at different levels.

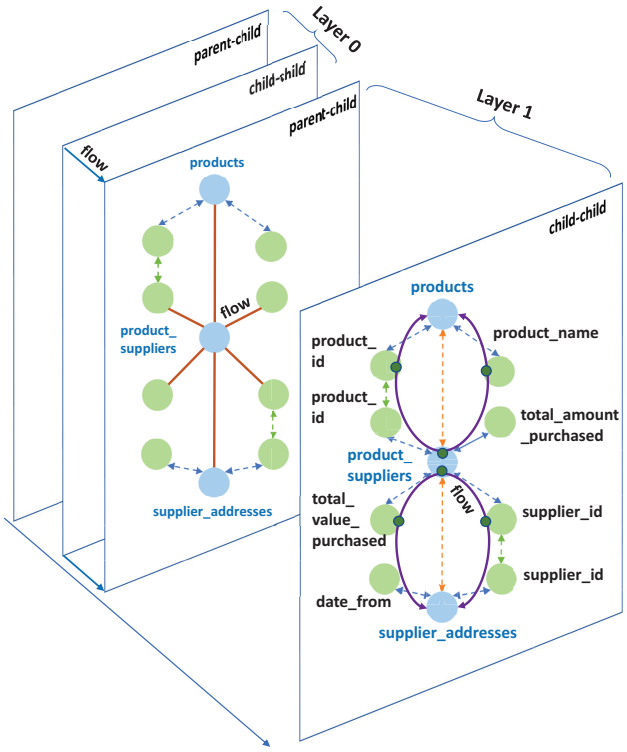

Figure 2: The architecture of our proposed CFGNN.

3 Methodology

In this section, we will first introduce our proposed CFGNN in detail. And then we formulate the task of question answering on tables. At last, we will illustrate the overview structure of our end-to-end model.

3.1 Cross Flow Graph Neural Network

CFGNN is a novel graph neural network proposed by us. Although the Spider is a unique complex and cross-domain dataset with multiple tables in TB-QA, which can be modeled as graph, we still introduce our GNN by general representation. It can also be applied to other areas if the data can be represented as graph.

Assuming we have a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. Nodes are $v \in \mathcal{V}$ and edges are $e = (v, v') \in \mathcal{V} \times \mathcal{V}$. The D-dimensional vector of node v can be represented as $\mathbf{h}_v \in \mathbb{R}^D$. For generalization, we focus on directed graphs, so (v, v') represents a directed edge $v \rightarrow v'$. v' is the neighbour node or child node of v. For undirected graphs, just add another edge in the opposite direction.

Inspired by recent research on GNNs (Li et al. 2016; Kipf and Welling 2017; Hamilton, Ying, and Leskovec 2017; Velickovic et al. 2018), we first use bi-directional RNN as the aggregation function for the flow of child-child nodes. That is to say, RNNs consider nothing about parent nodes. The RNN across different nodes share the same parameters. The process is shown in Eq. 1. To learn the difference rather than the order between different child-to-child sequences in the same graph, each sibling RNN has the same number and order of nodes. The nodes that do not appear in the sibling sequence are replaced by padding tokens.

$$\begin{aligned} \overrightarrow{\mathbf{h}}^k_{u,i} &= \overrightarrow{\mathrm{RNN}}(\overrightarrow{\mathbf{h}}^k_{u,i-1}, \mathbf{W}^k \mathbf{h}^{k-1}_{u,i}) \\ \overleftarrow{\mathbf{h}}^k_{u,|N(v)|-i-1} &= \overleftarrow{\mathrm{RNN}}(\overleftarrow{\mathbf{h}}^k_{u,|N(v)|-i}, \mathbf{W}^k \mathbf{h}^{k-1}_{u,|N(v)|-i-1}) \end{aligned} \tag{1}$$

where $i \in [1, |N(v)|), u \in N(v)$ and $\mathbf{h}^k_{u,i}$ denotes the embedding of the i-th neighbour node u of node v at layer k. And $N(v)$ means the neighbours of node v. $\mathbf{W}^k$ is trainable parameter at layer k. Arrows represent RNN in different directions.

Then we concatenate them together to $\bar{\mathbf{h}}^k_v$ in Eq. 2.

$$\bar{\mathbf{h}}^k_v = [\overrightarrow{\mathbf{h}}^k_{u,|N(v)|-1}, \overleftarrow{\mathbf{h}}^k_{u,0}] \tag{2}$$

Next we model the flow between parent nodes and child nodes. We use an attention mechanism to obtain the neighbour information for parent node v.

$$\begin{aligned} \hat{\mathbf{h}}^k_v &= \gamma\Big(\sum_{u \in N(v)} \alpha_{v,u} \mathbf{W}^k \mathbf{h}^{k-1}_u\Big) \\ \alpha_{v,u} &= \frac{e^{\beta(\mathbf{g}^T[\mathbf{W}^k \mathbf{h}_v, \mathbf{W}^k \mathbf{h}_u]}}{\sum_{u' \in N(v)} e^{\beta(\mathbf{g}^T[\mathbf{W}^k \mathbf{h}_v, \mathbf{W}^k \mathbf{h}_{u'}])}} \end{aligned} \tag{3}$$

where γ and β represent non-linear activation function. And $\mathbf{g}$ is a trainable vector for all layers.

Finally, as shown in Eq. 4, we concatenate the representation of child-child flow and parent-child flow and feed them to another RNN for reasoning flow between different layers. The states of all node in the graph are updated with history reasoning flow, current child-child flow and parent-child flow. This is why we call it *Cross Flow*.

$$\mathbf{h}^k_v = \overrightarrow{\mathrm{RNN}}(\mathbf{h}^{k-1}_v, [\bar{\mathbf{h}}^k_v; \hat{\mathbf{h}}^k_v]) \tag{4}$$

In this paper, we use Gated Recurrent Unit (GRU) (Cho et al. 2014) as RNN in our whole model. The general representation is as follows. z_t and r_t are gates and $\mathbf{h}_t$ represents the hidden states.

$$\begin{aligned} z_t &= \sigma(\mathbf{W}_{hz}\mathbf{h}_{t-1} + \mathbf{W}_{xz}\mathbf{x}_t + b_z) \\ r_t &= \sigma(\mathbf{W}_{hr}\mathbf{h}_{t-1} + \mathbf{W}_{xr}\mathbf{x}_t + b_r) \\ \hat{\mathbf{h}}_t &= \Phi(\mathbf{W}_h(r_t \odot \mathbf{h}_{t-1}) + \mathbf{W}_x\mathbf{x}_t + b) \\ \mathbf{h}_t &= (1 - z_t) \odot \mathbf{h}_{t-1} + z_t \odot \hat{\mathbf{h}}_t \end{aligned} \tag{5}$$

Figure 2 is an example for this task. At layer 1, the center blue node, which represents the table term *product suppliers*, is in the progress of cross flow. The purple curve in the child-child sublayer at layer 1 denotes the bi-directional RNN among the neighbour nodes of the center. The orange lines in the parent-child sublayer at layer 1 denotes the attention of the center node. These two flows cross at the same layer. And the flows of current and history cross between different layers.

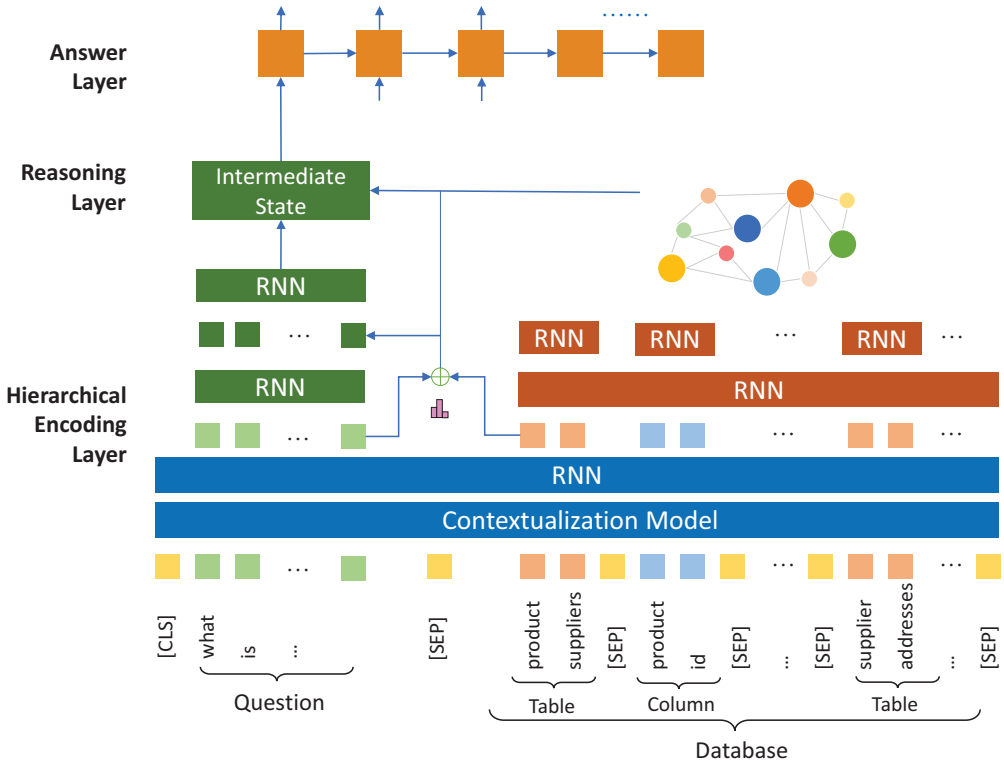

Figure 3: The overview of our model.

3.2 Formulation

Given a question with m words $Q = \{w_i\}_{i=1}^{m}$ and a database with l tables $D = \{T_k\}_{k=1}^{l}$, machines need to find the corresponding answer A. As the answer can be obtained from executed SQL queries S, we can formula the task to $p(S|Q, D)$. And C_j^k is the j-th term of the k-th table T_k in database. $h_{j,i}^k$ denotes the i-th token of C_j^k. In one table, the first term is table name, followed by other columns name. Besides, c_j^k, the encoding of C_j^k, can also be represented as $\{c_j\}_{j=1}^{n}$ ignoring the tables they belong to, where n is the number of all table names and column names. Tables in the same database are connected with each other by the relationship of primary and foreign key.

3.3 Model Structure

Although we use sequence-to-sequence structure for neural semantic parsing, the encoder part is more complicated than previous models. Thus we view the decoder as the answer layer and split the encoder to hierarchical encoding layer and reasoning layer. The overview of our model is shown in Figure 3. We will elaborate them from bottom to up.

Hierarchical Encoding Layer This layer is designed to provide representation of each component in tables. Because the structure of tables in databases is complex, it is very important for the whole model to obtain contextualized representation. Considering the schema of databases, we use coarse-to-fine hierarchical encoding to integrate the latest contextualized model BERT (Devlin et al. 2019) inspired by related work (Hwang et al. 2019; He et al. 2019). As shown in Figure 3, we first concatenate the question with all the names of tables and columns table by table, because the title and columns of the same table are adjacent and people tend to put related columns together (e.g. "date_from" and "date_to" in Figure 1). And `[SEP]` is used as the special token to separate the question, table names and column names. The final sentence fed to the BERT model is: `[CLS]`, w_1, $w_2, \cdots, w_m$, `[SEP]`, $h_{0,0}^0, h_{0,1}^0$, `[SEP]`, $h_{1,0}^0, h_{1,1}^0$, `[SEP]`, $\cdots$, `[SEP]`, $h_{j,i}^k, \cdots$, `[SEP]`, where $h_{0,0}^0$ and $h_{0,1}^0$ are tokens of the first table name, $h_{1,0}^0$ and $h_{1,1}^0$ are tokens of the first column name of the first table.

Comparing with the original way of appending a thin layer after BERT structure, it is too deep to fine-tune our proposed architecture with BERT. Inspired by recent research on whether to fine-tune (Peters, Ruder, and Smith 2019) and related methods (Zhu, Zeng, and Huang 2018; Zhang 2019), we feed this long sentence to the weight-fixed BERT model directly for contextualized embedding. Similar to ELMo (Peters et al. 2018), we use function $emb_i^{\text{BERT}} = \gamma \sum_{d=0}^{D} \alpha_d f(h_i^d)$ to utilize hidden states of all layers, where γ is designed to scale the vector according to the task and α_d is softmax-normalized weight for the d-th layer. These two weights are trainable. And h_i^d comes from the d-th layer of the i-th token in BERT. For long sentences, which exceeds the maximum length of pre-trained BERT, we split the sen-

tence with some coverage and combine them. To solve the index problem of WordPiece (Wu et al. 2016), we choose the first token as f function. Generally, the first token is the root of the word and can represent main meaning.

Then the output of pre-trained BERT model is refined by RNN, which is the coarsest encoding. After that, we use another RNN to encode the question and terms in tables, separately. Then we can obtain the embedding e_i for each token w_i in the question and $e^k_{j,i}$ for each token $h^k_{j,i}$ in the databases. For databases, the finest encoding is conducted among tokens in one table or column name by RNN. The representation of each term C^k_j in databases is the average of embedding of all tokens: $c^k_j = \frac{1}{I+1}\sum_{i=0}^{I} e^k_{j,i}$.

Reasoning Layer This layer aims to refine the representation of questions and terms in databases for answer layer. We first enhance the representation by some features and interaction between questions and databases. Then we use the multi-layer RNN and GNN for reasoning, separately.

For the representation of column name, column type is a useful feature, such as number, text and so on. Following Krishnamurthy, Dasigi, and Gardner (2017) and Bogin, Berant, and Gardner (2019), we assign trainable embeddings for column type. Then we concatenate type embeddings with the representation of table terms c_j and obtain new representation $\hat{c}_j$ for the database. And For the question, we can obtain the database-aware representation for each token of the question (shown in Eq. 6).

$$\begin{aligned} s^i_j &= {e_i}^{\mathrm{T}} c_j \\ a^i_j &= exp(s^i_j) / \sum_{k=1}^{n} exp(s^i_k) \\ h^{attn}_i &= \sum_{j=1}^{n} a^i_j c_j \end{aligned} \tag{6}$$

Besides, we also use other matching features, which can also be viewed as the process of entity linking on terms of databases. Then we use a bi-directional RNN to refine states to obtain the new representation of the question $\hat{h}_i$ in Eq. 7.

$$\hat{h}_i = \mathrm{BiRNN}(\hat{h}_{i-1}, [e_i, h^{attn}_i]) \tag{7}$$

Next our proposed CFGNN is used for reasoning on the database. As shown in Figure 2, the blue node represents tables and the green nodes represents the columns. Following Bogin, Berant, and Gardner (2019), we use three types for edges between nodes: table-column and column-table for columns in the table; foreign-primary key and primary-foreign key for columns; foreign-primary key and primary-foreign key for tables. It means if two columns are connected by foreign-primary key, we will add bi-directional edges both for two columns and for two tables they belong to. And different types have different weights for mapping the representation of nodes.

We can obtain the last layer of GNN $\bar{h}^k$ for each term in Eq. 8. For the question, we refine it again by another RNN in Eq. 9. Lastly, we concatenate it with the representation of the database by attention and obtain $\tilde{h}_i$ for answer layer in Eq. 10. And a^i_j is the same as that in Eq. 6.

$$\bar{c}_j = \mathrm{CFGNN}(\hat{c}_j) \tag{8}$$

$$\bar{h}_i = \mathrm{BiRNN}(\bar{h}_{i-1}, \hat{h}_i) \tag{9}$$

$$\begin{aligned} h^{gnn}_i &= \sum_{j=1}^{n} a^i_j \bar{c}_j \\ \tilde{h}_i &= [\bar{h}_i; h^{gnn}_i] \end{aligned} \tag{10}$$

Please note that tables and columns are fed to both BERT of hierarchical encoding layer and the sibling RNN of reasoning layer in the identical order for the same sample. Because not all nodes appear in each sibling sequence, the whole model needs to learn the difference between the original sequence fed to BERT and different sibling sequences by padding on missing nodes.

Answer Layer In this layer, we use intermediate state after reasoning to generate logic forms, which are expressed by λ-DCS (Liang, Jordan, and Klein 2011) language. Following (Pasupat and Liang 2016; Krishnamurthy, Dasigi, and Gardner 2017; Yin and Neubig 2017; Bogin, Berant, and Gardner 2019), we use grammar-based LSTM decoder and similar types and production rules of the logic form language. The production rules can be divided into table-dependent rules and table-independent rules. Table-dependent rules are designed to generate entities contained in tables, such as column. And Table-independent rules are for generating nonterminals or keywords of SQL queries. The embedding of them are all trainable. And the decoder generates the logic form from top to down. Giving the first root action, the decoder generate others according to production rules. Like generating natural languages, at each step of decoding, the LSTM cell takes the embedding of the previous rule and output new one by softmax.

4 Experiments

4.1 Dataset and Metrics

We use the Spider (Yu et al. 2018c), a large-scale, complex and cross-domain text-to-SQL dataset annotated by human, for our evaluation. To best of our knowledge, it is also the unique TB-QA dataset on multiple tables, the schema in which can be modeled as graph. The source of data comes from six datasets. There are 11,840 questions, 6,445 unique complex SQL queries, and 206 databases with multiple tables in this dataset. We follow division of origin dataset and use development set for our experiments, which ensures the database schema used in training not appear in evaluation.

We follow the official evaluation metric proposed by Yu et al. (2018c): component matching and exact matching. component matching compares the different parts of SQL queries, such as `SELECT`, `WHERE` and so on. According to the complexity of SQL queries, four levels are divided for evaluation: easy, medium, hard, extra hard.

4.2 Implementation Details

We use $\mathrm{BERT}_{\mathrm{LARGE}}$ model to obtain the contextualized before the hierarchical encoding layer. The size of the contextualized embedding is 1024. The dimension of type and

Model	Test					Dev
	Easy	Medium	Hard	Extra Hard	All	All
Seq2Seq[1]	22.0%	7.8%	5.5%	1.3%	9.4%	1.9%
Seq2Seq+Attention	32.3%	15.6%	10.3%	2.3%	15.9%	1.8%
Seq2Seq+Copying	29.3%	13.1%	8.8%	3.0%	14.1%	4.1%
SQLNet[2]	34.1%	19.6%	11.7%	3.3%	18.3%	10.9%
TypeSQL[3]	47.5%	38.4%	24.1%	14.4%	33.0%	8.0%
SyntaxSQLNet[4]	48.0%	27.0%	24.3%	4.6%	27.2%	24.8%
RCSQL[5]	-	-	-	-	24.3%	28.8%
Schema GNN[6]	61.8%	44.7%	26.5%	14.6%	39.4%	40.7%
CFGNN (ours)	**67.0%**	**47.8%**	**33.8%**	**19.1%**	**44.1%**	**48.7%**

Table 1: Accuracy of Exact Matching on SQL with different hardness levels. These models are: Dong and Lapata (2016) [1], Yu et al. (2018b) [2], Yu et al. (2018a) [3], Yu et al. (2018b) [4], Lee (2019) [5], Bogin, Berant, and Gardner (2019) [6]

Method	SELECT	WHERE	GROUP BY	ORDER BY	KEYWORDS
Seq2Seq	13.0%	1.5%	3.3%	5.3%	8.7%
Seq2Seq+Attention	13.6%	3.1%	3.6%	9.9%	9.9%
Seq2Seq+Copying	12.0%	3.1%	5.3%	5.8%	7.3%
SQLNet	44.5%	19.8%	29.5%	48.8%	64.0%
TypeSQL	36.4%	16.0%	17.2%	47.7%	66.2%
SyntaxSQLNet	62.5%	34.8%	55.6%	60.9%	69.6%
RCSQL	68.7%	39.0%	63.1%	63.5%	76.5%
Schema GNN	80.9%	40.7%	67.4%	70.1%	77.0%
CFGNN (ours)	**81.1%**	**42.9%**	**73.0%**	**73.7%**	**80.6%**

Table 2: F1 scores (test set) of Component Matching on SQL.

Question:
What is the id and type code for the template used by the most documents?
Return the id and type code of the template that is used for the greatest number of documents.

Schema:
templates: template_id, template_type_code ...
documents: document_id, template_id, document_name ...
paragraphs: paragraph_id, document_id, paragraph_text ...

Gold Answer / CFGNN:
SELECT documents.template_id, templates.template_type_code FROM documents JOIN templates ON documents.template_id = templates.template_id GROUP BY documents.template_id ORDER BY count (*) DESC LIMIT 1

Schema GNN:
SELECT templates.template_id, templates.template_type_code FROM templates GROUP BY templates.template_id ORDER BY count (*) DESC LIMIT 1

Figure 4: Case study for CFGNN.

rule embedding is 200. We maximize the log-likelihood of the gold sequence for training. The hidden size of RNN is 200 throughout our model. The output dimensions of forward and backward RNN are set to 100. We use ELU (Clevert, Unterthiner, and Hochreiter 2016) as non-linear activation function for GNN. The number of reasoning layers for questions and databases are set to 2. To prevent overfitting, we use dropout technique, and set it to 0.4. The Adamax (Kingma and Ba 2015) is used as our optimizer with 0.004 learning rate.

4.3 Results

We submit our code and model anonymously for the score of the hidden test set on Spider. As shown in Table 1, our full model achieves 44.1% (test set) and 48.7% (dev set) in exact matching on all SQL queries, which outperforms the previous Schema GNN model (Bogin, Berant, and Gardner 2019) by 4.7% (test set) and 8.0% (dev set). As shown in Table 2, our model also have outstanding performance on different SQL components in F1 score. [1] And we find SELECT and KEYWORDS perform best compared to other parts in components matching. Besides, for a fair comparison, we also reimplement the Schema GNN model proposed by Bogin, Berant, and Gardner (2019) and integrate it with our designed hierarchical encoding layer. [2] As shown in Table 3 and 4, the performance of our model is over Schema GNN

[1]We only list some related models, which are published before our model is submitted on the Spider leaderboard. There are still some models perform better than ours recently.

[2]The results of Schema GNN we reimplemented are shown on Table 3 and 4, which is slightly different from origin version.

Method	Easy	Medium	Hard	Extra Hard	All
RCSQL	53.2%	27.0%	20.1%	6.5%	28.8%
Schema GNN	64.8%	41.6%	29.3%	15.9%	40.9%
CFGNN (ours)	**69.2%**	**50.9%**	**34.5%**	**27.6%**	**48.7%**
w/o parent-child flow	68.8%	46.8%	26.4%	21.2%	44.5%
w/o child-child flow	62.8%	44.5%	27.0%	22.9%	42.5%
w/o reasoning flow	66.8%	48.4%	27.0%	21.2%	44.8%
w/o graph structure	68.0%	47.0%	21.8%	15.9%	42.7%
replace CFGNN with GNN	58.8%	39.5%	27.0%	14.1%	37.9%
replace CFGNN with GAT	67.2%	46.8%	28.7%	20.6%	44.4%
replace CFGNN with GGNN	62.8%	39.8%	29.9%	19.4%	40.3%

Table 3: Ablation studies on accuracy of Exact Matching (dev set).

Method	SELECT	WHERE	GROUP BY	ORDER BY	KEYWORDS
RCSQL	68.7%	39.0%	63.1%	63.5%	76.5%
Schema GNN	75.0%	42.8%	71.6%	67.9%	83.1%
CFGNN (ours)	81.9%	**52.5%**	71.5%	**76.6%**	**82.9%**
w/o parent-child flow	81.1%	44.2%	70.9%	73.1%	82.6%
w/o child-child flow	78.3%	46.5%	72.8%	69.1%	82.5%
w/o reasoning flow	**82.3%**	48.6%	73.2%	65.8%	80.9%
w/o graph structure	79.9%	45.8%	72.1%	65.0%	81.0%
replace CFGNN with GNN	73.9%	42.6%	66.9%	65.8%	82.8%
replace CFGNN with GAT	80.1%	43.5%	**73.2%**	70.0%	81.9%
replace CFGNN with GGNN	71.9%	45.1%	68.8%	68.3%	82.7%

Table 4: Ablation studies on F1 scores of Component Matching (dev set).

model with BERT on both accuracy of exact matching and F1 scores of component matching.

4.4 Ablation Study and Case Study

To study how each component contributes to the performance, we conduct an ablation analysis in Table 3. We can observe that all kinds of flows in the CFGNN can help our full model gain more improvement on accuracy. Especially, we can find that child-child flow, i.e., RNN aggregation among sibling nodes rather than parent-child nodes, makes important contributions to the performance of our model. Without it ($\bar{\mathbf{h}}_v^k$ in Eq. 4), the accuracy of exact matching drops 6.2% on all SQL queries and drops 6.4% and 7.5% on easy and hard queries, separately. Besides, without parent-child and reasoning flow (flow between layers), the accuracy also drops 4.2% and 3.9%, separately. And from Table 4, we also can observe that F1 scores on some components are improved greatly with different flows, such as `WHERE` and `ORDER BY`.

Because we mainly focus on the GNN, we also compare our proposed CFGNN with other related GNNs, i.e., GNN (Scarselli et al. 2009), GAT (Velickovic et al. 2018), GGNN (Li et al. 2016) with 3 layers. Experiments show that our model outperforms these GNNs in different dimensions. Although GAT performs better on `GROUP BY`, the overall accuracy is still lower than our proposed model.

We also study some cases between different configurations. The most common situation is that our proposed CFGNN model can comprehend columns and semantic relations across different tables more profoundly with cross flow. Here is an example in Figure 4. In the graph, *templates* and *documents.template_id* are sibling nodes (children of *documents*). Meanwhile, *documents* and *templates.template_type_code* are sibling nodes (children of *templates*). Our cross flow strengthens their connection and representation.

5 Conclusion

In this paper, we propose a novel GNN, i.e., cross flow graph neural networks (CFGNN), for question answering on complex tables. Unlike other GNNs which only consider the relationship of parent-child nodes, CFGNN can utilize the context of child-child nodes and aggregate them with reasoning flow between different layers. Two different RNN flows and attention flows in CFGNN interact with each other. In addition, we use the hierarchical encoding layer to obtain contextualized representation at different levels in tables. We believe that cross flow in GNN is a general approach. We will study further the capability of our approaches on other tasks in the future.

6 Acknowledgments

We thank Zhichun Wang, Ben Bogin and anonymous reviewers for their help and comments. We also thank the authors of the Spider for evaluating our model on the test set. This work is supported by the National Key Research and Development Program of China (No.2017YFB1402105).

References

Berant, J.; Chou, A.; Frostig, R.; and Liang, P. 2013. Semantic parsing on freebase from question-answer pairs. In *EMNLP*, 1533–1544.

Bogin, B.; Berant, J.; and Gardner, M. 2019. Representing schema structure with graph neural networks for text-to-SQL parsing. In *ACL*, 4560–4565.

Chen, B.; Sun, L.; and Han, X. 2018. Sequence-to-action: End-to-end semantic graph generation for semantic parsing. In *ACL*, 766–777.

Cho, K.; van Merriënboer, B.; Gulcehre, C.; Bahdanau, D.; Bougares, F.; Schwenk, H.; and Bengio, Y. 2014. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *EMNLP*, 1724–1734.

Clevert, D.-A.; Unterthiner, T.; and Hochreiter, S. 2016. Fast and accurate deep network learning by exponential linear units (elus). In *ICLR*.

Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*, 4171–4186.

Dong, L., and Lapata, M. 2016. Language to logical form with neural attention. In *ACL*, 33–43.

Dong, L., and Lapata, M. 2018. Coarse-to-fine decoding for neural semantic parsing. In *ACL*, 731–742.

Hamilton, W.; Ying, Z.; and Leskovec, J. 2017. Inductive representation learning on large graphs. In *NIPS*.

He, P.; Mao, Y.; Chakrabarti, K.; and Chen, W. 2019. X-sql: reinforce schema representation with context. *arXiv preprint arXiv:1908.08113*.

Hwang, W.; Yim, J.; Park, S.; and Seo, M. 2019. A comprehensive exploration on wikisql with table-aware word contextualization. In *KR2ML Workshop at NeurIPS*.

Kingma, D. P., and Ba, J. 2015. Adam: A method for stochastic optimization. In *ICLR*.

Kipf, T. N., and Welling, M. 2017. Semi-supervised classification with graph convolutional networks. In *ICLR*.

Kitaev, N.; Cao, S.; and Klein, D. 2019. Multilingual constituency parsing with self-attention and pre-training. In *ACL*, 3499–3505.

Krishnamurthy, J.; Dasigi, P.; and Gardner, M. 2017. Neural semantic parsing with type constraints for semi-structured tables. In *EMNLP*, 1516–1526.

Lee, D. 2019. Clause-wise and recursive decoding for complex and cross-domain text-to-SQL generation. In *EMNLP-IJCNLP*, 6047–6053.

Li, Y.; Tarlow, D.; Brockschmidt, M.; and Zemel, R. 2016. Gated graph sequence neural networks. In *ICLR*.

Liang, P.; Jordan, M.; and Klein, D. 2011. Learning dependency-based compositional semantics. In *ACL*, 590–599.

Liu, X.; He, P.; Chen, W.; and Gao, J. 2019. Improving multi-task deep neural networks via knowledge distillation for natural language understanding. *arXiv preprint arXiv:1904.09482*.

Pasupat, P., and Liang, P. 2016. Inferring logical forms from denotations. In *ACL*, 23–32.

Peters, M. E.; Neumann, M.; Iyyer, M.; Gardner, M.; Clark, C.; Lee, K.; and Zettlemoyer, L. 2018. Deep contextualized word representations. In *NAACL*.

Peters, M. E.; Ruder, S.; and Smith, N. A. 2019. To tune or not to tune? adapting pretrained representations to diverse tasks. In *RepL4NLP-2019*, 7–14.

Radford, A.; Wu, J.; Child, R.; Luan, D.; Amodei, D.; and Sutskever, I. 2019. Language models are unsupervised multitask learners. *OpenAI Blog* 1:8.

Scarselli, F.; Gori, M.; Tsoi, A. C.; Hagenbuchner, M.; and Monfardini, G. 2009. The graph neural network model. *IEEE Transactions on Neural Networks* 20(1):61–80.

Shi, T.; Tatwawadi, K.; Chakrabarti, K.; Mao, Y.; Polozov, O.; and Chen, W. 2018. Incsql: Training incremental text-to-sql parsers with non-deterministic oracles. *arXiv preprint arXiv:1809.05054*.

Sutskever, I.; Vinyals, O.; and Le, Q. V. 2014. Sequence to sequence learning with neural networks. In *NIPS*, 3104–3112.

Velickovic, P.; Cucurull, G.; Casanova, A.; Romero, A.; Lio, P.; and Bengio, Y. 2018. Graph attention networks. In *ICLR*.

Wu, Y.; Schuster, M.; Chen, Z.; et al. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

Yin, P., and Neubig, G. 2017. A syntactic neural model for general-purpose code generation. In *ACL*, 440–450.

Yu, T.; Li, Z.; Zhang, Z.; Zhang, R.; and Radev, D. 2018a. TypeSQL: Knowledge-based type-aware neural text-to-SQL generation. In *ACL*, 588–594.

Yu, T.; Yasunaga, M.; Yang, K.; Zhang, R.; Wang, D.; Li, Z.; and Radev, D. 2018b. SyntaxSQLNet: Syntax tree networks for complex and cross-domain text-to-SQL task. In *EMNLP*, 1653–1663.

Yu, T.; Zhang, R.; Yang, K.; Yasunaga, M.; Wang, D.; Li, Z.; Ma, J.; Li, I.; Yao, Q.; Roman, S.; Zhang, Z.; and Radev, D. 2018c. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *EMNLP*, 3911–3921.

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

Zhang, X. 2019. MCˆ2: Multi-perspective convolutional cube for conversational machine reading comprehension. In *ACL*, 6185–6190.

Zhong, V.; Xiong, C.; and Socher, R. 2018. Seq2sql: Generating structured queries from natural language using reinforcement learning. In *ICLR*.

Zhu, C.; Zeng, M.; and Huang, X. 2018. Sdnet: Contextualized attention-based deep network for conversational question answering. *arXiv preprint arXiv:1812.03593*.