Generating SQL queries from natural language

Ikshu Bhalla
Department of Computer Science
Stanford University
Stanford, CA 94305
ikshub@stanford.edu

Archit Gupta
Department of Computer Science
Stanford University
Stanford, CA 94305
rambo@stanford.edu

Abstract

With the increasing amount of world's information stored in relational database systems, it is imperative to have intelligent interfaces for interaction with the data. This should be done so as to enable even non expert users of SQL to make efficient use of the data. Generating executable queries from the question posed by the user has been a long standing problem, which has been gaining much momentum lately. Traditional methods involve using sequence to sequence models but the generalized nature of such models does not exploit the full structure of a SQL query. In this paper we explore and implement the existing state of the art model for generating SQL query from natural language question proposed by Xu et al. (2017) , for the SELECT column and aggregate operator parts of the SQL query. We observe that our approach and implementation improves the model's accuracy by $\mathbf{1.5}\%$ for the aggregate operation and $\sim\mathbf{2.5}\%$ for the SELECT column part of the query, and presents the new state of the art models for each of these.

1 Introduction

Parsing natural language questions into SQL can be viewed as a special form of machine translation. Machine translation is known as one of the fundamental problems in machine learning, attracting extensive research efforts in the last few decades (Waibel et al. (1992); Koncar and Guthrie (1997); Castano et al. (1997)). Traditional machine translation is used to translate text from one language to another. In this age of digitization, relational databases are used in almost every industry to store information and records. While they provide a powerful way to store information, extracting these records requires skill and knowledge of query languages like SQL that retrieve information from databases. Generating SQL queries from natural language has wide applications, and is something that has been researched upon both in academia (Yaghmazadeh et al. (2017)) as well as in the industry (Zhong et al. (2017)). This project tries to explore and implement the techniques introduced by Zhong et al. (2017), solving the problem of retrieving information, so that natural language questions can be used to fetch the required data.

Our approach, described in detail in section 4, involves training a deep neural network that translates natural language questions into their corresponding SQL queries. A SQL query can be broken down into 3 parts: *SELECT, operator (MAX, MIN, COUNT . . .)* and *WHERE*. The idea is to avoid the sequence-to-sequence structure where order does not matter. Thus, the SQL query can be generated based on a sketch. We train these individual components of a SQL query by using various deep learning techniques. In particular, we use Long Short-Term Memory networks (Hochreiter and Schmidhuber (1997)) to encode the input questions and column tokens, and build more deep architecture on top of the encodings to predict the different parts of the SQL query. We also make use of a special kind of attention, called *column attention* while encoding the question tokens such that the column context is preserved while encoding the input question.

The dataset of choice for this task was WikiSQL, introduced by Zhong et al. (2017). It is, to the best of our knowledge, the only large scale dataset used for natural language to SQL tasks, which requires generalization over new table schemas. This is described in detail in section 3. We then compare our results against the current state of the art on this dataset.

We train the select and aggregate components independently using cross entropy loss obtained by comparing the actual SQL query, and the predicted query. Training the individual components separately gives us new **state of the art performance** on the aggregate and the select part of the query. We achieve an aggregate accuracy (Acc_{agg}) of **92.5%**, and SELECT column accuracy (Acc_{sel}) of **94.02%**.

2 Background/Related Work

Natural language interfaces aims at integrating natural language processing and human-computer interactions. It seeks to provide means for humans to interact with computers through the use of natural language. In this paper, we explore a specific aspect of the research which applies to relational databases which can be essentially summarized as translation of natural language questions asked by the user to corresponding SQL queries. Neural machine translation (NMT) is an approach to machine translation that uses an artificial neural network to predict the likelihood of a sequence of words, typically modeling entire sentences in a single integrated model. Deep learning applications appeared first in speech recognition in the 1990s. The research on using neural networks for machine translation started gaining momentum in 2014, when Sutskever et al. (2014) published a paper on sequence to sequence learning with neural networks. Other work using encoder-decoder networks include Cho et al. (2014), Bahdanau et al. (2014), Wu et al. (2016). Variants of seq2seq were used by Vinyals et al. (2015) and Dong and Lapata (2016) to predict next characters in the sequence.

Major improvements in the area of parsing natural language to SQL has been achieved upon recently by Zhong et al. (2017), and Xu et al. (2017). Zhong et al. (2017) introduce Seq2SQL, a deep neural network architecture coupled with policy-based reinforcement learning (RL) for translating natural language questions to corresponding SQL queries. Seq2SQL is trained using a mixed objective, combining cross entropy losses and RL rewards from in-the-loop query execution on a database. Xu et al. (2017) proposed SQLNet, a sketch based architecture to solve this problem. They make use of the structure of a SQL query, and generate it from a dependency sketch which describes the dependency between different parts of the SQL query. A neural network is then used to predict each slot in their sketch. Compared to Zhong et al. (2017), they avoid employing a reinforcement learning approach to solve the order matters problem in the WHERE part of the query by proposing a new approach based on column attention.

3 Dataset

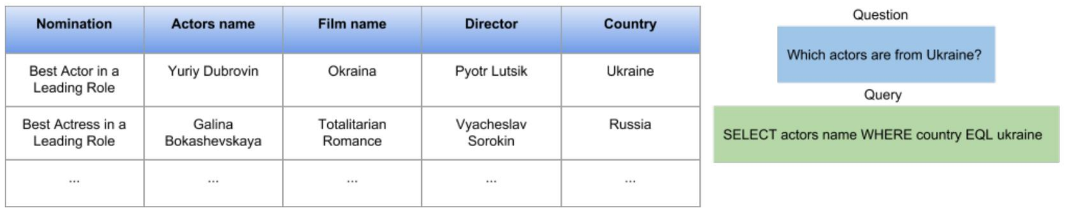
We now describe the dataset used in this paper in detail. Zhong et al. (2017) introduced the WikiSQL database that is used throughout this paper. The the query generation task posed by this dataset is unique when compared to other NL2SQL datasets in a few ways:

- The queries in WikiSQL span across a huge number of tables and thus require a model which is not just generalized over the queries, but also over tables with different schema.
- The dataset was generated by crowd sourcing the natural language questions and thus requires a model which more than just overfits the questions generated using a template.

As shown in figure 1, each example of the dataset, which is the input to the model, consists of two parts:

- natural language question for which the query is to be synthesized
- schema of the corresponding table

Schema of the table consists of both the names of the columns and the type of the data contained in the columns (i.e. numbers v/s string). We also used the resplit dataset, as presented by Xu et al.



Nomination	Actors name	Film name	Director	Country
Best Actor in a Leading Role	Yuriy Dubrovin	Okraina	Pyotr Lutsik	Ukraine
Best Actress in a Leading Role	Galina Bokashevskaya	Totalitarian Romance	Vyacheslav Sorokin	Russia
...	...	...	...	...

Question

Which actors are from Ukraine?

Query

SELECT actors name WHERE country EQL ukraine

Figure 1: An example from the dataset depicting the table, question and the corresponding SQL query

(2017), which is different from the original dataset as presented in SQLNet. In the resplit dataset, tables are not shared between the train, validation and test set which helps furthermore in building a more generalized model. The label for each example is the SQL query to be synthesized, represented as an index of one of the operators (NONE, MAX, MIN, COUNT, SUM, AVG) and the column selected, which is the index of one of the table columns. It is worth noting that we do not need to generate the SQL grammar (keywords like SELECT, FROM, AND ...) and the table-id (for the FROM part) since input question is already assumed to be corresponding to one table only, given as a part of the input.

4 Model

The task explored in this paper is based on the WikiSQL dataset. The problem tackled is the generation of SQL query from the natural language question. Keeping in mind the privacy concern around the actual data contained in the database, the model needs to be agnostic to the contents of the database, since the data might not be available while generating the query from the user supplied question. Thus, the model must be able to construct the SQL query based solely on the table schema and the natural language question. This not only evades the privacy concerns but also makes the model more generic, enough to work with any table and data.

We now present our approach which is based on that proposed in SQLNet but differs in a few aspects which enables us to achieve the new state of the art results. Most of the existing models employed for the task of generation of SQL query from the natural language disregard the inherent structure in the SQL query as depicted in figure 1. The main idea is to separate the SQL grammar (SELECT, FROM, COUNT ...) from the question specific *slots* that need to be combined with the grammar to form the full SQL query.

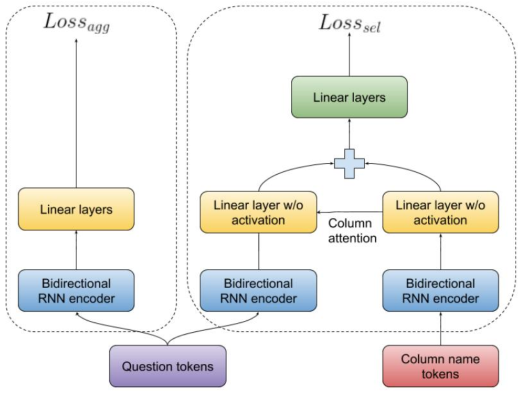

$Loss_{agg}$
$Loss_{sel}$
Linear layers
Linear layers
Linear layer w/o activation
Column attention
Linear layer w/o activation
Bidirectional RNN encoder
Bidirectional RNN encoder
Bidirectional RNN encoder
Question tokens
Column name tokens

Figure 2: Broad overview of the architecture of the model

Figure 2 demonstrates the inputs that are required for prediction of operation and column of the table to which the question pertains. We now describe in detail the approach for the prediction of the operation and the SELECT column from the table headers and the input question.

4.1 Aggregate operation

It is worth mentioning that the set of aggregate operators that are present in the WikiSQL database is: $\{$*NONE, MAX, MIN, COUNT, SUM, AVG*$\}$.

From the figure 2, it can be seen that the aggregate operation depends only on the question. This can be intuitively seen from an example. Consider the question:

> ”How many total runners had jockeys of Olivier Peslier with placings under 2?”.

The correct operator in the SQL query for this question is SUM since the question asks for ”How many”.

Let E_Q denote the encoding of the question. For generation of this encoding we employ a bidirectional RNN with LSTM cells running on top of question tokens $(x_1^Q, x_2^Q \dots x_M^Q)$ where M is the total number of question tokens. We use the final state output of the RNN as E_Q. It is worth noting that the LSTM cells in the forward and backward network of the RNN do not share parameters. We calculate the prediction over all aggregate operators as:

$$\mathbf{P_{agg}} = \mathbf{softmax}(W_2^{agg}\mathbf{tanh}(W_1^{agg}E_Q + b_1^{agg}) + b_2^{agg}) \tag{1}$$

where E_Q is of dimension d, which is the output size of the RNN encoder, and we use a multi layered perception over E_Q to generate the final predictions over aggregate operators. Dimensions of W_1 is $d \times d$, b_1 and b_2 is d each and that of W_2 is $6 \times d$. Finally we take the $\mathbf{max}$ of $\mathbf{softmax}$ normalized predictions $\mathbf{P_{agg}}$ over 6 operators to get the predicted index of the operator. We use the index of operator thus found to calculate the loss $\mathbf{Loss_{agg}}$

Notably, our approach is different from that used in SQLNet(Xu et al. (2017)), and outperforms it. The latter proposes the use of predicted SELECT column, described in section 4.2, for the prediction of aggregate operator. We propose the using of only the question for two reasons. First, the aggregate operation should be infer-able from the question alone since the question contains enough information required for the prediction of the aggregate operation. Second, this avoids forced mis-predictions due to mis-predicted SELECT column. A wrongly predicted column might force the model to wrongly predict the operation.

4.2 SELECT Column

Before we proceed with the methodology used to predict the column from the table for the SELECT part of the query, we would like to introduce the concept of column attention.

Column attention

Let E_Q denote the encoding of the question (same as before), E_{col} denote the column encodings such that E_{col_i} is the encoding of col_i of the given table. Here each col_i is composed of multiple words $(x_1^{col_i}, x_2^{col_i} \dots)$. We employ the same approach to encode each column name as used to encode the question tokens. Dimension of E_{col} is $C \times d$ where C is the total number of columns across all tables in a batch.

Use of E_Q poses the problem that while encoding the question to generate E_Q, the encoder may not remember particular information useful for predicting the column. For example, consider the question ”How many countries got 796.7 **points**?”. It has token ”points” which is relevant to predicting column ”Points” from the list of columns in the table ([”Rank”, ”Member Association”, **”Points”**, ”Group stage”, ”Play-off”, ”AFC Cup”])

To capture this information, we introduce the concept of column attention which enables us to use $E_{Q|col}$ instead of E_Q which incorporates the intuition described above.

Let H_Q denote a $M \times d$ matrix of all hidden state outputs for the question with length M. Let E_{col} denote a $C \times d$ matrix containing encoding of the columns in the given table with C such columns.

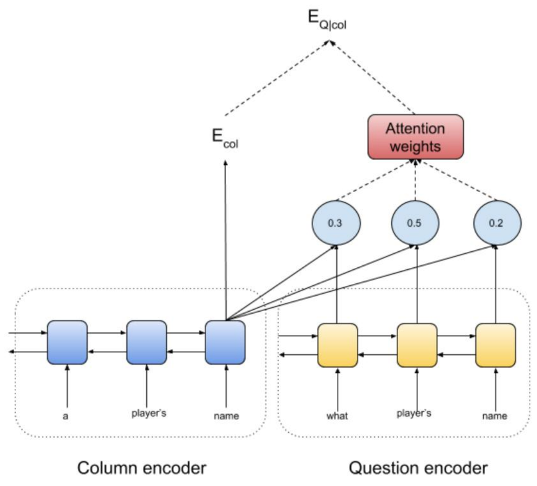


Figure 3: Column attention mechanism for computing the encoding of the question by applying attention w.r.t. to a given column

For a given column col, with encoding E_{col_i}, we define attention weight α, a vector of dimension M as:

$$\alpha = \textbf{softmax}(H_Q W E_{col_i}) \tag{2}$$

where W is a trainable matrix of dimension $d \times d$.

Having computed the attention weights α, we can calculate $E_{Q|col}$ as the weighted sum of each question token's hidden state output based on α:

$$E_{Q|col_i} = \alpha H_Q \tag{3}$$

From this point onward, we can generalize and replace the use of E_Q with $E_{Q|col}$ in our calculations to predict the SELECT column.

Column prediction

Prediction of the column for the SELECT is similar to the process followed for prediction of the aggregate operator. There are notable differences though, that leads to a more complicated architecture of the model for the prediction of the SELECT column that we will now describe in detail.

As can be seen in figure 2, the prediction of the column from the set of columns of a table is dependent not just on the question, but also the columns of the table . For example, consider the question:

”How many significant relationships list Will as a virtue?”

which provides the intuition for the use of table column names as the input for prediction where the model could learn to predict column **”Significant Relationship”** from the set of all columns: ”Approximate Age”, ”Virtues”, ”Psycho Social Crisis”, **”Significant Relationship”**, ”Existential Question”, ”Examples”.

At this point we can conclude that the prediction of the SELECT column is solvable using a pointer over the table columns; we select the column that best matches the input question. For this, we first generate the encoding of the question in the same was as done for the aggregate, as explained in section 4.1.

We then generate the encoding of the columns of the table associate with the query. We employ the same architecture of using a bidirectional RNN using LSTM cells to encode the columns of the table.

We define P_{sel} as the prediction calculated over the column of the tables for the SELECT part of the SQL query.

$$\begin{aligned} L_1 &= W_1^{sel} E_{col} + b_1^{sel} \\ L_2 &= W_2^{sel} E_{Q|col} + b_2^{sel} \\ P_{sel} &= \mathbf{softmax}(W_3^{sel} \mathbf{tanh}(L_1 + L_2) + b_3^{sel}) \end{aligned} \tag{4}$$

Finally we take the max of $softmax$ normalized predictions $\mathbf{P_{sel}}$ over the columns of a table and find the index of the column for the SELECT part of the query. We use the index of operator thus found to calculate the loss $\mathbf{Loss_{sel}}$

4.3 Objective Function

While training, we calculate two losses as mentioned above $Loss_{agg}$ and $Loss_{sel}$ and make use of gradient descent to minimize the value of the two loss functions *independently*. It is notable that we employ a different objective to that used in similar research as described in SQLNet (Xu et al. (2017)) and Seq2Sql (Zhong et al. (2017)) in which a mixed objective function is minimized. The mixed objective function as described in SQLNet is $L = Loss_{agg} + Loss_{sel}$. We deduced that optimizing the mixed objective function is less optimal than optimizing the individual loss functions and is analyzed in detail in section 5.2.

5 Experimental Results and Analysis

We make use of the tokenized re-split dataset as describe in section 3. We implement both our models using TensorFlow (Abadi et al. (2015)) from scratch. We run several experiments that we now describe and analyze in detail.

5.1 Training details

5.1.1 Input tokenization and encoding

We first used the Stanford CoreNLP tokenizer by Manning et al. (2014) to parse the input questions and table columns and tokenize the sequences. We use GloVe (Pennington et al. (2014)) embeddings of size 50 for representation of each word in the input tokens before using them with the bidirectional RNN. It is worth mentioning that the word embeddings thus used are trainable and give a better performance compared to non trainable embeddings. Any word for which the embeddings are not found in the pre-trained GloVe embeddings is assigned the embedding corresponding to *UNK* token which is initialized with random embedding, and as with other word embeddings, is trainable.

5.1.2 Model parameters

Our model employs 3 bidirectional RNNs using LSTM cells (one used to encode the question in each aggregate and select column model and another one used in select column model to encode the table column names). Each LSTM has one layer with **128** units. Each of the single layered neural networks used on top of the encodings to predict the aggregate operators and column index have have 2 layers, **128** units in the hidden layer and use $\tanh$ non linearity wherever applicable. We use Adam optimizer with a learning rate of **0.001**. All layers have a dropout of **0.1** for input and output. We train our model in batches of **64** examples for **20** epochs. The model was trained on two Tesla M60 GPUs.

5.2 Results

Table 1, summarizes the overall results obtained with our approach. We compare our results against the state of the art research done on this dataset for the aggregate and select part of the SQL query. We observe that our approach improves the results on each of these and present a new state of the art. Our accuracy on the aggregation operators is 92.5% which is $\sim 1.5\%$ better than the state of the art and 94.02% for the select column part which is $\sim 2.5\%$ better than the existing state of the art results.

Table 1: Summarized results of our experiments compared to that of existing stae of the art

Model	Acc_{agg}	Acc_{sel}
Seq2SQL(Zhong et al. (2017))	90.0%	89.6%
SQLNet (Seq2set+CA)	90.1%	91.1%
SQLNet (Seq2set+CA+WE)	90.1%	91.5%
Ours	92.5%	94.02%

Using structure of the SQL query

SQL queries can be viewed as composed of 3 parts as shown in Table 2

Table 2: Subparts of a SQL query and the slots to be predicted in those subparts

Part of query	**Slots to predict**
SELECT	aggregate operator, select column
FROM	table-id
WHERE	Triple of the form: (where column - operator - value)

WikiSQL task does not require the prediction of the table-id since every question is asked for a particular table. In this paper, we only explore the SELECT part of the query where we need to predict the aggregate operator and the SELECT column for the table.

An alternative to synthesizing the SQL query structurally is to model the problem as a sequence generation task given the question and column sequence. In this approach, we can use a seq2seq model to generate the entire SQL query (including the grammar). Seq2seq consists of two RNN's called encoder and decoder. We begin by providing the encoder RNN with an input that consists of the word embeddings of the input question, the column tokens, as well as the SQL tokens. The hidden state of the encoder feeds into the decoder, which generates a probability distribution over the whole vocab that is used to predict the next query word. The output of the seq2seq model is also in the form of natural language combined with special sql tokens, corresponding to the grammar of the SQL language (ex. $symselect \rightarrow SELECT, symwhere \rightarrow WHERE \ldots$).

An example from the predictions by our model is (where the words symselect, symagg, symcol etc. are the special SQL tokens):

symselect symagg symcol no. symwhere symcol player symop = symcond patrick o'bryant

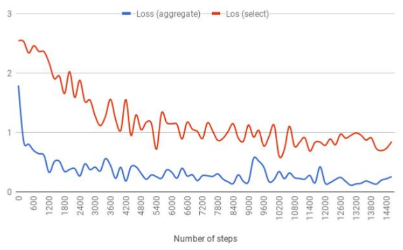
We concluded that the seq2seq model is provably not a good solution for this task. We achieved an overall accuracy of 30% with a seq2seq model. It performs fairly well on shorter queries, but it fails to generate larger SQL queries. Part of the reason why seq2seq does not give good results from SQL generation is because the output in seq2seq consists of the entire vocabulary, whereas we know that the tokens in SQL query depend only on the input question, the columns, and the SQL tokens only. Therefore, generation of output over the whole vocab would lead to a lot of sparsity, noise, as well as is computationally much more expensive to train.

Loss

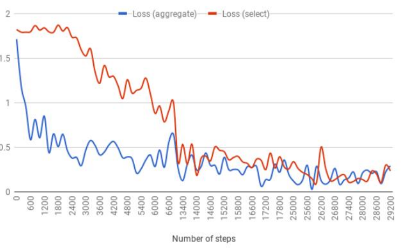
We extensively trained the models with both the variants of the objective function i.e the mixed loss function as described in section 4.3 and independent objective function (different loss for aggregation model and SELECT columns model). As can be seen in figure 4, the model trained by optimizing the two losses independently outperforms the one which minimizes the mixed loss. We attribute this behaviour to the fact that when minimizing the mixed loss, $Loss_{agg}$ decreases, decreasing the overall loss L whereas $Loss_{sel}$ does not decrease much, however when training both the models independently, both $Loss_{agg}$ and $Loss_{sel}$ are optimized yielding a better overall accuracy.

Column attention

We evaluate the performance of the model with and without columns attention as described in section 4.2. We observed that adding column attention improves the performance of the SELECT column



(a) Losses when model is trained with mixed objective function



(b) Losses when objective functions are optimized independently

Figure 4: Plots of losses for the two models

model by $\sim 2.5\%$. This can be attributed to the fact that the model captures the context of the columns when encoding the question which improves the accuracy of prediction of the columns for the SELECT part of the query.

6 Conclusion

In this paper, we worked on generating SQL queries from natural language questions. Initially, we generated SQL queries using a generic seq2seq model as the baseline. We found that while the accuracy for short queries was fairly reasonable, the model started losing context over large queries, and started predicting the same keyword. Also, the training took a huge amount of time to converge, as it generated predictions over the whole vocab. We then exploited the inherent structure of SQL query to break it into parts, that can be predicted independently. Our model architecture is a composition of LSTM (used to encode the question and column tokens), followed by multi layered perception. We evaluated our model by comparing across several aspects involved in the architecture ex. trainable v/s non trainable word embeddings, various hyper parameters, mixed objective function v/s independently optimized models, single layered vs multi layered perception.

In this paper, we explored our approach only on the SELECT part of the SQL query and achieved new state of the art results for these models. We think that this approach will be even more fruitful when applied to the WHERE part of the query. We hope to explore this in the future.

It is also worth noting that the WikiSQL dataset, although contains a large number of tables and human generated questions, is limited in the richness of SQL query. The dataset is limited to SELECT ... FROM table-id WHERE ... kind of queries. This can be expanded to include other SQL functions, like join.

Acknowledgments

We would like to thank our project mentor Kevin, Head TA Abi and Professor Richard Socher for their continuous advice whenever we needed them. We would also like to thank all other TA's for their support and continued availability to address our questions.

References

M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014. URL `http://arxiv.org/abs/1409.0473`.

M. A. Castano, F. Casacuberta, and E. Vidal. Machine translation using neural networks and finite-state models. *Theoretical and Methodological Issues in Machine Translation (TMI)*, pages 160–167, 1997.

K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

L. Dong and M. Lapata. Language to logical form with neural attention. *CoRR*, abs/1601.01280, 2016. URL `http://arxiv.org/abs/1601.01280`.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, Nov. 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL `http://dx.doi.org/10.1162/neco.1997.9.8.1735`.

N. Koncar and G. Guthrie. A natural language translation neural network. In *New Methods in Language Processing*, pages 219–228, 1997.

C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. McClosky. The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, pages 55–60, 2014. URL `http://www.aclweb.org/anthology/P/P14/P14-5010`.

J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014. URL `http://www.aclweb.org/anthology/D14-1162`.

I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215, 2014. URL `http://arxiv.org/abs/1409.3215`.

O. Vinyals, M. Fortunato, and N. Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems*, pages 2692–2700, 2015.

A. Waibel, A. N. Jain, A. E. McNair, J. Tebelskis, L. Osterholtz, H. Saito, O. Schmidbauer, T. Sloboda, and M. Woszczyna. Janus: Speech-to-speech translation using connectionist and non-connectionist techniques. In *Advances in neural information processing systems*, pages 183–190, 1992.

Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, L. Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144, 2016. URL `http://arxiv.org/abs/1609.08144`.

X. Xu, C. Liu, and D. Song. Sqlnet: Generating structured queries from natural language without reinforcement learning. *CoRR*, abs/1711.04436, 2017. URL `http://arxiv.org/abs/1711.04436`.

N. Yaghmazadeh, X. Wang, and I. Dillig. Automated migration of hierarchical data to relational tables using programming-by-example. *CoRR*, abs/1711.04001, 2017. URL `http://arxiv.org/abs/1711.04001`.

V. Zhong, C. Xiong, and R. Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning. *CoRR*, abs/1709.00103, 2017. URL `http://arxiv.org/abs/1709.00103`.