NLP2SQL Using Semi-supervised Learning

H. Vathsala[1](✉) and Shashidhar G. Koolagudi[2]

[1] Centre for Development of Advanced Computing, Bengaluru, Karnataka, India
vathsala.h@gmail.com

[2] Department of Computer Science and Engineering, NITK, Mangalore, Karnataka, India

Abstract. Human Computer interaction has been moving towards Natural language in the modern age. SQL (Structured Query Language) is the chief database query language used today. There are many flavors of SQL but all of them have the same basic underlying structure. This paper attempts to use the Natural Language inputs to query the databases, which is achieved by translating the natural language (which in our case is English) input into the SQL (specific to MySQL database) query language. Here we use a semi-supervised learning with Memory augmented policy optimization approach to solve this problem. This method uses the context of the natural language questions through database schema, and hence its not just generation of SQL code. We have used the WikiSQL dataset for all our experiments. The proposed method gives a 2.3% higher accuracy than the state of the art semi-supervised method on an average.

Keywords: NLP (Natural Language Processing) · SQL (Structured Query Language) · MAPO (Memory Augmented Policy Optimization) · NLP2SQL

1 Introduction

Natural Language Processing has been in existence for more than 50 years. As computers continue to become more and more affordable and accessible, the importance of user interfaces that are effective, robust, unobtrusive and user-friendly become more pronounced. Since natural language usually is the preferred mode for human-human interaction, it is only sensible to adopt it in Human Computer interaction also. Hence there has been a major move towards incorporating Natural Language Processing (NLP) into the Human Computer Interaction.

A large swathes of information is stored in the form of relational databases (RDBMS). Relational databases are becoming increasingly important in actual applications and Web sites. They are often used by people who do not have great competence in this domain and who do not know exactly their structure. The only way to efficiently access, manage and manipulate the data in the databases is through the use of the Structure Query Language (SQL). This is why translators from natural language to SQL queries are being developed.

These translators will prove to be intelligent interfaces for interaction with the data. A lot of approaches have been used to tackle the problem including the traditional sequence to sequence model [19] which is pretty generalized and doesn't make use of the full structure of the SQL queries. Hence a new approach which uses sequence to set along with column attention was proposed to address the previous problems. We solve this problem using a reinforcement learning algorithm (vanilla policy gradient) coupled with off-policy experience replay.

The chief contributions of this paper include

– Use of GloVe [13], a pre-trained word embedding.
– Apart from storing the high positive trajectories, poorly performing trajectories have also been used for training.
– Hyper-parameter tuning with respect to the architecture of internal nodes in every LSTM, dimension of GloVe [13].
– Use of Epsilon Greedy strategy for systematic exploration.
– Use of Bidirectional LSTM [6].

The following section provides details of, (i) Literature review done during the research process (Sect. 2) (ii) The proposed algorithm (Sect. 3) (iii) Implementation Experimentation and results (Sect. 4)

2 Literature Review

This section describes the literature review performed during the research process.

2.1 MAPO

Memory Augmented Policy Optimization (MAPO) [7] reduces the variance in the policy gradient estimates and improves the sampling efficiency with the help of a high trajectory memory buffer. Memory clipping alleviates cold start of policy gradients and an efficient search algorithm is used to explore the memory buffers, for which bloom filters are used. For calculation of Expected rewards from the large memory buffers, sampling is employed. Distributed actor-learner (with 30 actors) is used to quicken the training process. This technique was first employed in [12].

2.2 Coarse2Fine

This algorithm is analogous to 3-step machine translation. It uses attention mechanisms [1] in both encoder and decoder. A rough sketch a is first generated, and this is used to guide the final decoding. The rough sketch is devoid of argument-names, variable-names (replaced with it's datatype), and all other lower level details. All the components of input x, rough sketch a and final output y are treated as sequences. Hence, this becomes a seq2seq problem [20]. GloVe vectors are used to encode the input. The training objective maximizes the joint probability of the final meaning representation and that of the intermediate rough sketch also. During preprocessing of the data, 10-dimensional PoS tag embeddings are appended to the embeddings of the words in the natural language question. This novel idea was presented in [5]. Parent feeding technique is employed in the current work. The authors of this paper have not mentioned how this work is better than that of Yin and Neubig [18] which uses a generic system of abstract syntax trees(AST).

2.3 Gated Convolution Neural Network (G-CNN)

This work aims to perform language modeling using Gated CNNs. This is the first non recurrent approach to language modeling, and the paper achieves substantial accuracy results on large-context datasets also. The main unique selling proposition (USP) is that the CNN units can be parallelized and hence this model is more faster than LSTM based models, only during inference, not necessarily during training. It uses stacked convolutions which increases the number of operations per context to O(nk) (per kernel), where n is the context size and k is the kernel width. Causal convolutions are used, i.e., the kernel can only see the previous words and not the future words. This ensures that the model does not cheat.

The initial part of the sentence is padded with $(k - 1)$ tokens, where k is the width of the kernel. Comparison between Gated Linear Unit (GLU) and Gated Tanh Unit has been performed. GLU is better than the latter, because it handles the vanishing gradient problem better. RNNs are parallelized over different sequences while training, and CNNs can be parallelized over different contexts or tokens. The model uses residual activation blocks and the gradients have been clipped to $[-5, 5]$. Kaiming initialization, Momentum and weight normalization are some of the techniques that help in faster convergence even with large learning rates = 1. Adaptive softmax is used, instead of the conventional softmax. This is less expensive but approximately gives the same results. It is empirically found that a context size of 40 tokens is sufficient to obtain high accuracy. Similarly, it is sufficient to limit the backpropagation to 40 time steps (instead of the theoretical infinite limits).

Although this work provided better results theoretically, we found it hard to reproduce similar results in our paper when we replaced LSTMs/RNNs with G-CNNs.

2.4 DeepFix

The model trains a neural network for fixing the syntactic (non-semantic) programming errors in the C language. The authors train a GRU with attention to achieve 60% accuracy. The literals are given a fixed vector representation (as they do not affect the syntax) and all the identifiers are given a similar fixed representation. The initial weights of the GRU is drawn from a uniform distribution within the range $(-0.07, 0.07)$. Both encoder and decoder have 4 layers, 300 cells each. The model performs the error localization upto 80% accuracy.

Certain types of errors might never be suitable for LSTMs. For instance, assignment of an array into another is not allowed in the C language (unlike Python). The solution to this is to use a for loop and index through all the elements of the array. It might be impossible to enable an LSTM to generate such solutions.

It is not entirely clear if the vanilla attention mechanism is adding value to the process. The model is stopped from continuing if one of it's proposed corrections is rejected by the oracle(during iterative repair). If this bound is reduced to 3/4 it might give the model more chances to take corrective actions, because a single line might contain multiple fixes(at different positions) which are required to eliminate a single error message.

Eliminating bias has not been argued comprehensively in the paper because real-world data might also have a similar distribution of the types of errors. By forcing equal representation, we could be losing the information regarding the priority of errors.

The mutations in the data set (seeded) were performed manually by the authors. This might have introduced bias and might be a reason for the poor performance. The authors select only one erroneous program per student for every programming task citing the concern of bias. But since syntactic errors are independent of the semantic nature of the program, this leads to wastage of training data. It is true that there may be some correlation between the programming task and the syntax errors and a specific student, but it will be useful if a comparison had been provided.

2.5 SQLNet

The paper proposes a sketch-based model for SQL generation. The contents of the sketch is filled with the help of neural networks. Column attention is used extensively in three different steps. A sql2set model is used in the paper, as the order of the conditions are to be considered only during evaluation in SQL. This technique is very similar to the approach of generating Abstract Syntax Tree (AST) and filling in the slots. To decide whether or not to include a column name in the WHERE condition, binary classification is performed. The mechanism of predicting the column names for SELECT and WHERE clauses are identical. But the weights of the trainable matrices cannot be shared, because the column in either of SELECT or WHERE need not be present in the other also (in fact it might not be present in most of the cases). In predicting columns for WHERE condition, the authors do not share weights between the bi-LSTMs used for encoding the column names and the question. They propose that this ensures independence of the decisions made. But this is a tradeoff, because the current methodology requires a large number of trainable parameters. Apart from this, the WikiSQL dataset does not contain examples of self-join where a single table is involved.

2.6 Execution Guided Decoding (EGD)

EGD uses the partially generated queries and weeds out the incorrect SQL statements(those that produce a runtime error) and queries that do not match any records. Some of the questions may not have any suitable records in the table. The authors hypothesize that this could be because the decoder generated an overly-restrictive where condition. EGD is performed only during inference, and not during training. Using EGD during training may potentially improve the model performance. EGD is model-agnostic and is useful in a variety of auto-regressive tasks. It is difficult to use EGD in pure seq2seq models, because it's hard to know what stage the partial programs are executable. Execution guidance only tries to reduce the number of execution errors. As a by product, the accuracy increases by a small value (because the number of erroneous programs decreases). This increases the number of semantically meaningful programs, but not necessarily the number of semantically correct programs.

The authors make an impoprtant observation that many queries in WikiSQL are grammatically wrong. This might hinder possible approaches like that of GANs. It is not clear whether the beam-size plays an important role in determining the accuracy.

The paper does not use the Teacher-Forcing method while training of LSTMs. We have incorporated this methodology in our proposed algorithm.

Execution guidance only tries to reduce the number of execution errors. As a by product, the accuracy increases by a small value (because the number of erroneous programs decreases). This increases the number of semantically meaningful programs, but not necessarily the number of semantically correct programs

It's not possible to implement EGD for all auto-regressive tasks. Example, for tasks like music generation, there are no standard set of rules to determine if an audio piece is syntactically and semantically correct (except for systems like Carnatic/Hindustani music).

2.7 SQLova

It is very similar in structure to SQLNet [17], but has three important contributions.

- It uses BERT word representations instead of GloVe [13].
- It uses Sequence to sequence (Seq2Seq) model
- It uses Execution Guided Decoding [15]

NLQ is concatenated with the table headers. Every token consists of token, position and segment embedding(see BERT for more details). It is not entirely clear if this ordering of the input had any effect on the accuracy metric. (This is because this paper largely borrows from SQLNet, and SQLNet does not impose such restrictions on ordering of input). [17] predict the where-value using pointer-networks (Vinyals). But this paper follows [5] and predicts the start and the end token for the where-val instead of going for a seq2seq approach. Also, where-val is conditioned on where-col and where-op. The order of where conditions is ignored in measuring logical form accuracy in the model. The final output of where-clause will be that one which has the highest joint probability with respect to all the four where predictors.

The authors claim that SQLNet conditions the where-val only on the where-col. But this is not true. It conditions the where-val on NLQ, column_name, and the partially generated query. This probability is calculated for every column and softmax is used to choose the one with the highest probability.

2.8 XSQL

This work brings in a fresh set of fine-tuning of the results by leveraging recent strides in natural language processing like the MT-DNN algorithm [11] over BERT [3]

- It Uses MT-DNN instead of BERT to encode the question and to generate a new structural representation for schema
- The [EMPTY] token is appended to every table schema to account for cases where there are no WHERE conditions.
- KL-Divergence is used as an objective only for predicting the WHERE condition column name.
- The ground truth Q in predicting the WHERE column name is calculated as follows:
- If there is no where clause, Q[EMPTY] receives probability mass 1 for special column [EMPTY]
- For n $\geq$ 1 where clauses, each where column receives probability mass of 1/n Although this paper shows improvements in predicting the individual slots of the SQL query, it would behave been fair playing ground if the authors presented the test and dev accuracy that is prevalent in the NL2SQL research community.

3 Proposed Algorithm

The current work proposes to generate the SQL statements by using the policy gradient [14] algorithm. The algorithm has been developed on top of Memory Augmented Policy Optimization [10]. Similar to MAPO, the memory buffers have also been used to store both the high reward trajectories and the low reward trajectories. Sampling and training periodically from these buffers ensure that the agent/model does not forget the high/low-reward trajectories.

The low-reward trajectories are also included in the memory buffer because agent has equal opportunity to learn from both the high-reward and low reward trajectory. The usage of policy gradient makes it all the more important to reinforce both positive and negative reward trajectories.

The authors of MAPO [10] have not made use of pretrained word embeddings, but they have used random word embeddings. This work alleviates those lacunas by using GloVe word embeddings [13]. This ensures that the inductive priors from the unsupervised training of GloVe word embeddings are leveraged in the proposed NLP2SQL model.

Similar to the original MAPO paper, the epsilon greedy strategy to ensure systematic exploration of the search space has been retained. This step uses bloom filters in order to store billions of patterns to ensure that no programs are missed out.

We have also performed hyper parameter tuning with respect to the size of the pretrained word embeddings and we realized that the optimal value is 300 dimensions. Apart from these, bidirectional LSTM in both encoder and decoder has also been used.

4 Implementation, Experiments and Results

The algorithm has been implemented using Python 3.6, Tensorflow 1.7, gensim 3.2.0, nltk 3.3, Babel 2.5.3, bloom-filter 1.3. The model is tested against the WikiSQL dataset [19] in the Linux environment over a cluster of 2 Nvidia K40 GPUs. Tensorflow GPU optimizations have also been used.

The reinforcement learning agent has been allowed to train for a maximum of 15000 training steps in the environment. We have used the standard baselines from OpenAI [4] for comparison with the vanilla policy gradient algorithm.

4.1 Dataset

WikiSQL [19] is a large database built mainly for the development of interfaces for the natural language processing for relational databases. Annotated with SQL queries, the dataset primarily consists of entries from Wikipedia and other common sources of information. In comparison with other question-answer datasets (like WikiTableQuestions), WikiSQL has simple semantics as the SQL queries have simpler structure and fewer operators. While most of the state-of-the-art models are dependent on the question-program pairs as the data for the supervised learning, we will be using the question-answer pairs for semi-supervised learning.

4.2 Model Architecture

As mentioned earlier, the model proposed in this paper is an improvement over the Memory-Augmented Policy Optimisation(MAPO) [10] model. Neural Symbolic Machine (NSM) [9] framework has been used the implementation of the model, with a bi-directional LSTM as the encoder and two-layer LSTM for both encoder and decoder. The ability of LSTM in assigning credit from high-dimensional and/or continuous actions based on backpropagation and its learning from long-term temporal dependencies to infer states in partially observable tasks makes it one of the best models to be used in the current scenario [2].

4.3 Results

The results from the experiment can be visualised in Table 1 and Table 2, and in the Fig. 1 shown below.

Table 1 is the compilation of the measurement of the accuracy of each feature in the model. As more features are added over the vanilla MAPO implementation, the dev accuracy and the test accuracy increases. Finally, after the inclusion of all the features, dev accuracy of **76.8%** and test accuracy of **77.6%** is achieved.

Table 1. Measurement of accuracy of each feature in the model.

Features	Dev accuracy (%)	Test accuracy (%)
MAPO	72.4	74.9
GloVe Word Embeddings	75.1	76.7
Word Embeddings of Dimension 300	76.9	77.7
Bidirectional LSTM	75.3	75.6
Epsilon greedy exploration	77.3	78.0
Training with extremely good bad performances	77.3	77.9
All the above changes	**79.3**	**78.7**

Figure 1 is a line graph of the dev accuracy of the model against the training steps (epochs). The dev accuracy grows slowly over the epochs, from 62.5% for very small epochs up to 76% for large epochs. The accuracy saturates to 76.8%, which is recorded as the performance of the model proposed.

Table 2 is a comparison of the performance of the model with other state-of-the-art supervised and semi-supervised learning models in terms of dev accuracy and test accuracy. The accuracy of the proposed model is significantly larger than the state-of-the-art semi-supervised learning models (models proposed in [15, 17, 19] and [8]), and is close to the accuracy of the supervised learning model (proposed in [5]).

4.4 Discussion

Analysis of the dev accuracy and test accuracy results of the model, and comparison of its performance with various state-of-the-art models can be concluded as follows:

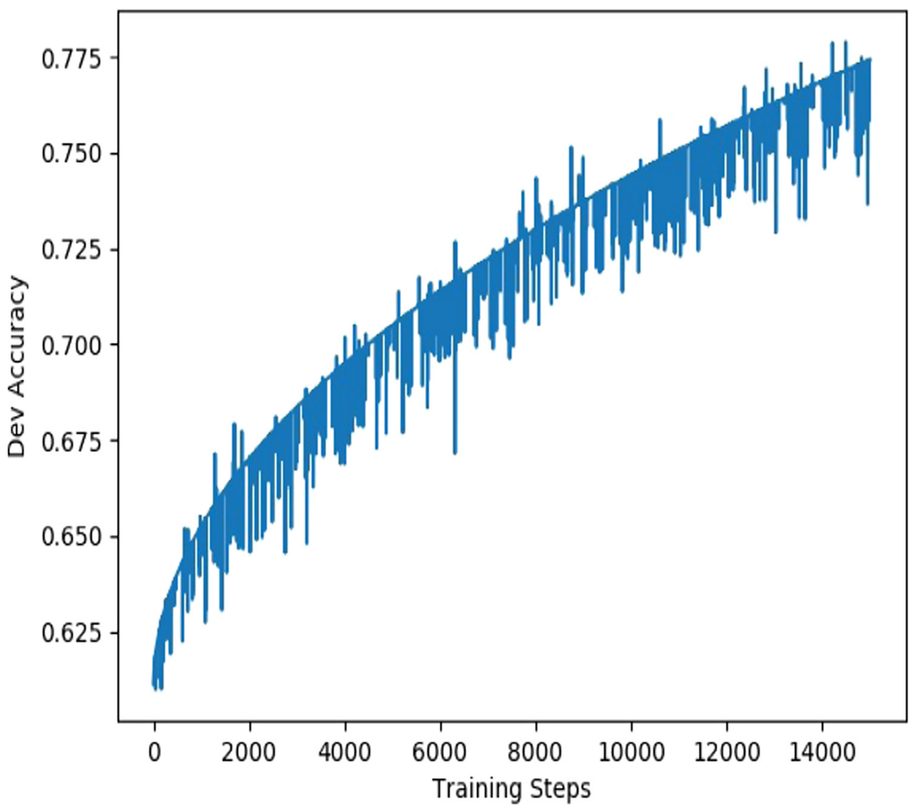

Fig. 1. Dev accuracy vs number of training epochs.

Table 2. Comparison with the previous works in supervised and weakly-supervised setting

Fully supervised	Dev accuracy (%)	Test accuracy (%)
[19]	60.8	59.4
[15]	67.1	66.8
[17]	69.8	68.0
[8]	68.3	68.0
[5]	79.0	78.5
Semi supervised	Dev accuracy (%)	Test accuracy (%)
MAPO	72.4	72.6
MAPO (mean of 5 runs)	72.2	72.1
MAPO (std of 5 runs)	0.2	0.3
MAPO (ensemble of 10)	–	74.9
Current algorithm	**79.3**	**78.7**

- Addition of various features (like GloVe embeddings and epsilon greedy exploration) over vanilla MAPO has resulted in the increase in dev accuracy and test accuracy of the model
- The dev accuracy curve is of positive slope but negative curvature with respect to the training epochs. This implies that the dev accuracy increases with increase in epochs, but there is a drop in the rate of increase of the accuracy. Also, the accuracy peaks to a value of 76.8%
- The performance of the model proposed is better than vanilla MAPO model and the current state-of-the-art semi-supervised models, and is on par with the current state-of-the-art supervised models.

We also explored the option of incorporating recent policy gradient algorithms like Proximal Policy Optimization(PPO) and Trust Region Policy Optimization(TRPO). But these algorithms are substantially different from that considered in the MAPO research paper. The major unique selling point of these algorithms over policy gradient is that they use different objective functions like KL-Divergence and Clipping respectively, to ensure that the newer policy is not significantly different from the old policy. This is done to ensure that the agent is robust and is not brittle.

The baseline paper that we have chosen is MAPO, which makes use of replay buffers to *remember* high trajectory rewards. Although Actor-Critic Experience Replay (ACER) [16] algorithm resembles this setup, the ACER algorithm does not specifically address the usage of high-reward trajectories. More specifically, we could not find theoretic justification with respect to the convergence properties in this modified algorithm.

We also considered the possibility of pure Q-learning algorithms like Deep Q Learning(DQN), Double DQN and Duelling DQN. We faced substantial difficulty in framing this problem in a manner that is amenable to Q-learning in particular. More importantly, historically Q-learning has not been applied to search problems with such a large search space.

5 Conclusion

We have attempted to solve the problem of "NLP2SQL" using policy gradients. We have used cutting edge techniques from program repair machine translation and question-answering to solve this problem. This method gives 2.3% higher accuracy than the state of the art on an average.

As future work, the theoretical underpinnings of this algorithm need to be verified. More specifically, it is necessary to obtain convergence proofs for this algorithm so that it can be applied to other problem areas like robotic control and natural language understanding. Another line of thought worth exploring would be the, although tangential to the topic of this paper, is optimal tuning of hyper-parameters. Such tuning techniques would be extremely useful in generating provably optimal model architectures.

Acknowledgements. We thank Chenna Keshava B.S, Rithesh K and H.L Praveen Raj, Department of Computer Science and Engineering, National Institute of Technology Karnataka, India for programming the said concepts during the current work.

References

1. Bahdanau, D., Chorowski, J., Serdyuk, D., Brakel, P., Bengio, Y.: End-to-end attention-based large vocabulary speech recognition. CoRR abs/1508.04395 (2015), http://arxiv.org/abs/1508.04395
2. Bakker, B.: Reinforcement learning by backpropagation through an LSTM model/critic. In: 2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning, Honolulu, HI, 2007 (2007)
3. Devlin, J., Chang, M., Lee, K., Toutanova, K.: BERT: pre-training of deep bidirectional transformers for language understanding. CoRR abs/1810.04805 (2018), http://arxiv.org/abs/1810.04805
4. Dhariwal, P., et al.: Openai baselines. GitHub, GitHub repository (2017)
5. Dong, L., Lapata, M.: Coarse-to-fine decoding for neural semantic parsing. CoRR abs/1805.04793 (2018), http://arxiv.org/abs/1805.04793
6. Greff, K., Srivastava, R.K., Koutník, J., Steunebrink, B.R., Schmidhuber, J.: LSTM: a search space odyssey. CoRR abs/1503.04069 (2015), http://arxiv.org/abs/1503.04069
7. Gupta, R., Pal, S., Kanade, A., Shevade, S.: Deepfix: fixing common c language errors by deep learning. In: Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, AAAI 2017, pp. 1345–1351. AAAI Press (2017). http://dl.acm.org/citation.cfm?id=3298239.3298436
8. Hwang, W., Yim, J., Park, S., Seo, M.: A comprehensive exploration on wikisql with table-aware word contextualization. CoRR abs/1902.01069 (2019), http://arxiv.org/abs/1902.01069
9. Liang, C., Berant, J., Le, Q.V., Forbus, K.D., Lao, N.: Neural symbolic machines: learning semantic parsers on freebase with weak supervision. CoRR abs/1611.00020 (2016), http://arxiv.org/abs/1611.00020
10. Liang, C., Norouzi, M., Berant, J., Le, Q.V., Lao, N.: Memory augmented policy optimization for program synthesis with generalization. CoRR abs/1807.02322 (2018), http://arxiv.org/abs/1807.02322
11. Liu, X., He, P., Chen, W., Gao, J.: Multi-task deep neural networks for natural language understanding. CoRR abs/1901.11504 (2019), http://arxiv.org/abs/1901.11504
12. Mnih, V., et al.: Playing atari with deep reinforcement learning. CoRR abs/1312.5602 (2013), http://arxiv.org/abs/1312.5602
13. Pennington, J., Socher, R., Manning, C.: Glove: global vectors for word representation. In: Proceedings of the 2014 Conference on Empirical Methods In Natural Language Processing (EMNLP), pp. 1532–1543 (2014)
14. Sutton, R.S., McAllester, D.A., Singh, S.P., Mansour, Y.: Policy gradient methods for reinforcement learning with function approximation. In: Advances in Neural Information Processing Systems, pp. 1057–1063 (2000)
15. Wang, C., Huang, P.S., Polozov, A., Brockschmidt, M., Singh, R.: Execution-guided neural program decoding. arXiv preprint arXiv:1807.03100 (2018)
16. Wang, Z., et al.: Sample efficient actor-critic with experience replay. arXiv preprint arXiv:1611.01224 (2016)

17. Xu, X., Liu, C., Song, D.: Sqlnet: generating structured queries from natural language without reinforcement learning. arXiv preprint arXiv:1711.04436 (2017)
18. Yin, P., Neubig, G.: A syntactic neural model for general-purpose code generation. arXiv preprint arXiv:1704.01696 (2017)
19. Zhong, V., Xiong, C., Socher, R.: Seq2sql: generating structured queries from natural language using reinforcement learning. arXiv preprint arXiv:1709.00103 (2017)
20. Ziqiang, C., Chuwei, L., Wenjie, L., Li., S.: Joint copying and restricted generation for paraphrase. In: AAAI (2017)