An Improved Coarse-to-Fine Method for Solving Generation Tasks

Wenyu Guan[1,2], **Qianying Liu**[3], **Guangzhi Han**[2], **Bin Wang**[4] **and Sujian Li**[1]
[1] Key Laboratory of Computational Linguistics, MOE, Peking University
[2] School of Software and Microelectronics, Peking University
[3] Graduate School of Informatics, Kyoto University
[4] Xiaomi AI Lab, Xiaomi Inc.
{guanwy, hanguangzhi10, lisujian}@pku.edu.cn
ying@nlp.ist.i.kyoto-u.ac.jp, wangbin11@xiaomi.com

Abstract

The coarse-to-fine (coarse2fine) methods have recently been widely used in the generation tasks. The methods first generate a rough sketch in the coarse stage and then use the sketch to get the final result in the fine stage. However, they usually lack the correction ability when getting a wrong sketch. To solve this problem, in this paper, we propose an improved coarse2fine model with a control mechanism, with which our method can control the influence of the sketch on the final results in the fine stage. Even if the sketch is wrong, our model still has the opportunity to get a correct result. We have experimented our model on the tasks of semantic parsing and math word problem solving. The results have shown the effectiveness of our proposed model.

1 Introduction

The coarse-to-fine (coarse2fine) methods have been applied in many generation tasks such as machine translation (Xia et al., 2017) , abstract writing (Wang et al., 2018b) and semantic parsing (Dong and Lapata, 2018). They have shown excellent performances but still have many disadvantages. Traditional coarse2fine models usually tackle one task in two stages. In the first stage (coarse stage), a low-level seq2seq model is used to generate a rough sketch, which makes the data more compact and alleviates the problem of data sparsity. Some examples of sketches are shown in Table 1. Besides, Sketches in this stage are also easier to generate. Then, in the fine stage, both text and previous sketches will be input to another high-level seq2seq model to predict the final result so that the high-level model can produce a precise output.

In the coarse2fine models, the concept of template sketch provides a new view of compiling a rough template, but how to guarantee its quality is still a problem. Meanwhile, details from the fine stage will be filled into sketches to produce the final result, so the quality of sketches serves an essential influence on the result. If the generated sketches are in high quality, the coarse2fine model performs well. Otherwise, we fail to get an excellent output. The main reason is that the sketch is misleading and has no possibility to be corrected once it is wrong.

In this paper, we propose an improved coarse2fine model to solve this problem. It has a similar framework which consists of two levels of seq2seq models. First, the model predicts a rough sketch in the coarse stage. In the fine stage, compared with traditional coarse2fine model, we use the generated sketches in the coarse stage as assistant information to help the decoder. Besides, We set a weight to control the degree of how the sketch affects the fine stage. Higher weight means that the fine stage is strictly guided by the sketch. Lower weight will decrease the impact of the sketch on the final output and give the model more flexibility to generate the result which does not rely on the sketch. For different tasks, we will tune the weights by experience and make a balance between the sketch guidance and the model's correction ability.

Our model is a universal framework which can apply on many generation tasks. In this work, we apply it on two semantic parsing tasks (text2logic and text2code) and math word problem (MWP) solving task. Experimental results show that our model achieves a better performance than some baseline models in these tasks.

2 Related Work

In this section, we briefly introduce the tasks where we experimented our model and also the

Tasks	Type	Example
Text2logic	logic	(argmin $0 (and (place:t $0) (loc:t $0 s0)) (elevation:i $0))
	sketch	(argmin #1 (and place:t @1 loc:t @2) elevation:i @1)
	text	what is the lowest point in s0 ?
Text2code	code	decode = curry(_proxy_method, method=bytes.decode)
	sketch	NAME = NAME(NAME, NAME=NAME.NAME)
	text	call the function curry with 2 arguments: _proxy_method and method set to bytes.decode[bytes.decode], substitute the result for decode.
MWP	equation	$x = 150 + 2 - 50$
	sketch	$x = \langle num \rangle + \langle num \rangle - \langle num \rangle$
	text	There are 150 science books, and the storybooks are 50 books less than the science books. How many books are there in the storybooks?

Table 1: Examples of text, sketches and generating goals in different datasets.

method we applied.

2.1 Semantic Parsing

Semantic parsing is a task of translating natural language into computer executable language such as logic form, code in computer language and SQL query. Traditional semantic parsing usually adopts rule based method Tang and Mooney (2000); Wong and Mooney (2007); Andreas et al. (2013). Recently, with the development of neural network techniques, there are many new semantic parsing models with neural methods. Of them, Seq2seq models have been widely applied in semantic parsing tasks. The encoder encodes the text and the decoder predicts the logic symbols (Dong and Lapata, 2016). The seq2tree model encodes inputs by LSTM and generates the logic form by conditioning the output sequences or trees on the encoding vectors.(Dong and Lapata, 2016). Abstract syntax networks (ASN) represent the output as the abstract syntax trees (ASTs) (Rabinovich et al., 2017). Its decoder uses a dynamically-determined modular structure paralleling the structure of the output tree.

2.2 Math Word Problem

Math word problem (MWP) aims to teach computers to read the questions in natural language and generate the corresponding math equations. The methods of solving math word problems can be mainly classified into two categories. The first category is the template-based models which summarize some templates through locating similar questions from a given dataset and then fill the concrete numbers into the templates to solve problems (Huang et al., 2017; Wang et al., 2017). Some cases of math word problems, math equations, templates and sketches are shown in Table 2 These methods are intuitive, but it is difficult to obtain high-quality templates due to data sparsity and transfer them to other datasets. The second category of methods mainly exploits the seq2seq framework to generate the solution equations (Wang et al., 2018a). Recently this kind of methods have shown outstanding performance without manual feature engineering, but they are prone to generate wrong numbers due to its generation flexibility. Some researches have applied reinforcement learning (Huang et al., 2018) or a stack (Chiang and Chen, 2018) to improve the decoding process.

2.3 Coarse-to-fine method

Generalized coarse-to-fine method divides problems into different stages and solves them from coarse to fine. This method is widely applied in computer vision (Gangaputra and Geman, 2006; Pedersoli et al., 2011; Wen et al., 2019) and natural language process (Mei et al., 2016; Choi et al., 2017). The special coarse-to-fine method in this paper is based on end-to-end framework. It has two seq2seq models, generating target data from a coarse stage to a fine stage. Xia et al. (2017) proposed *polish mechanism* with two levels of decoders. The first decoder generates a raw sequence and the second decoder polishes and refines the raw sentence with deliberation. Their model performs excellently on machine translation and text summarization, which is also the first time to use this kind of coarse-to-fine model. Wang et al. (2018b) used a similar framework to write paper abstracts. In the fields of semantic pars-

Question	Woodpeckers can eat 645 pests per day, and frogs can eat 608 pests in 8 days. How many insects do woodpeckers eat more than frogs every day?
Equation	$x = 645 - 608/8$
Template	$x = \langle num1 \rangle - \langle num2 \rangle / \langle num3 \rangle$
Numbers	$\{\langle num1 \rangle : 645, \langle num2 \rangle : 608, \langle num3 \rangle : 8\}$
Question	The garment factory originally planned to make 1080 sets of suits, which would be completed in 20 days. Actually they finished 72 sets per day, How many sets they produced everyday more than the original plan?
Equation	$x = 72 - 1080/20$
Template	$x = \langle num3 \rangle - \langle num1 \rangle / \langle num2 \rangle$
Numbers	$\{\langle num1 \rangle : 1080, \langle num2 \rangle : 20, \langle num3 \rangle : 72\}$
Question	The shirt factory produced 640 shirts in the past 4 days, but now it produces 350 shirts per day. How many shirts it produces each day more than it used to ?
Equation	$x = 350 - 640/4$
Template	$x = \langle num3 \rangle - \langle num1 \rangle / \langle num2 \rangle$
Numbers	$\{\langle num1 \rangle : 640, \langle num2 \rangle : 4, \langle num3 \rangle : 350\}$
Sketch	$x = \langle num \rangle - \langle num \rangle / \langle num \rangle$

Table 2: Examples of math word problems with different equations but same sketches.

ing, Dong and Lapata (2018) applied a two-level coarse-to-fine model in text2logic, text2python and text2SQL. This framework also shows significant improvement in these parsing tasks.

3 Problem Formulation

In this work, we aim at generating structured languages. Each instance contains a piece of text in natural language with m words $\{w_i\}_1^m$, generating the target output $\{e_i\}_1^{|e|}$. We learn sketches $\{s_i\}_1^{|s|}$ from $\{w_i\}_1^m$. We decompose the probability distribution $p(e|w)$ into a combination of sketch's conditional probability:

$$p(e|w) = p(e|s, w)p(s|w) \tag{1}$$

In this paper, we compute $p(e|s, w)$ and $p(s|w)$ step by step:

$$p(s|w) = \prod_{t=1}^{|s|} p(s_t|s_{1,\ldots,t-1}, w) \tag{2}$$

$$p(e|s, w) = \prod_{t=1}^{|e|} p(e_t|e_{1,\ldots,t-1}, s, w) \tag{3}$$

3.1 Framework

The coarse2fine model consists of four main components: text encoder, sketch decoder, sketch encoder, and final decoder.

During the coarse stage, the text encoder and sketch decoder predicate sketch by computing $p(s|w)$ step by step. Then, in fine stage, the sketch encoder will encode the sketch and the decoder takes the text encoder's output and sketch to compute the probability distribution. The framework is shown in Figure 1.

3.2 Coarse Stage

In the coarse stage, a basic seq2seq architecture is used to generate sketches. Firstly, question text is split into tokens and sent into an embedding layer. Then, a two-layer bidirectional LSTM (Hochreiter and Schmidhuber, 1997) will read the embedded word one-by-one and produce a sequence of hidden states $\{q_i\}_1^m$:

$$q_i^f = LSTM(emb(w_i), q_{i-1}^f) \tag{4}$$

$$q_i^b = LSTM(emb(w_i), q_{i+1}^b) \tag{5}$$

$$q_i = [q_i^f, q_i^b] \tag{6}$$

Each decoding step, the embedding vector of previously symbol and previous hidden state are sent to LSTM. Then, we calculate the probability distribution of the sketch through attention mech-

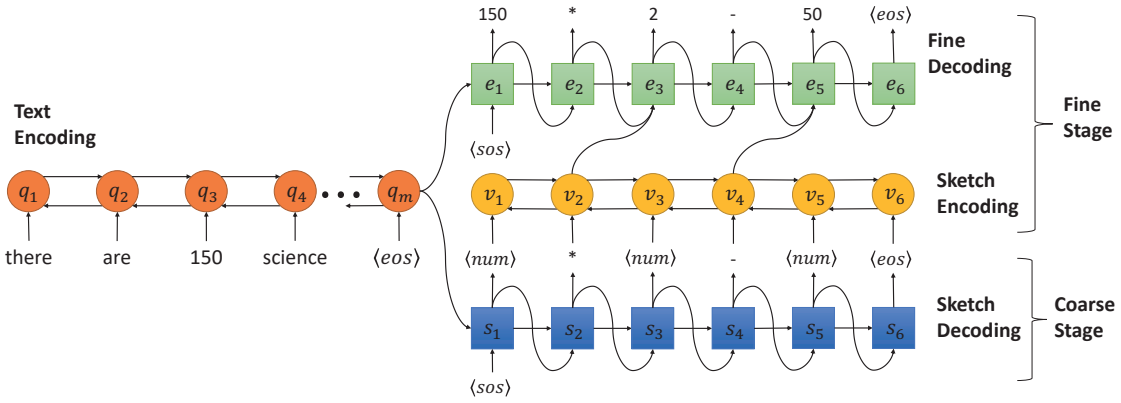

Figure 1: Improved Coarse-to-Fine Model framework.

anism:

$$h_j = LSTM(emb(s_{j-1}), h_{j-1}) \tag{7}$$

$$a_{ji} = softmax(q_i h_j) \tag{8}$$

$$c_j = \sum_{i=0}^{m} a_{ji} h_j \tag{9}$$

$$p(s_j|s_{1,\dots,j-1}, w) = softmax(U[tanh(W[c_j; h_j] + b_{attn})] + b) \tag{10}$$

h_j is the decoder hidden state in j step, a_{ji} is the attention score, c_j is the context vector. Beyond that, U, W, b_{attn} and b are model's parameters.

3.3 Fine Stage

In this stage, the fine decoder uses the sketch as assistant information to predict the final result. At the beginning of the fine stage, a bidirectional LSTM is used as sketch encoder to encode the sketch. After taking the encoded sketch as a part of the input, the fine decoder has perceptions of the whole sketch in low-level meaning. The process of sketch encoding is similar to question encoding, while the difference is that the input of (4) and (5) are changed to sketch symbols $\{s_i\}_1^{|s|}$.

Then, the fine decoder will use the encoded sketch to help its decoding process. As shown in Figure 1, fine decoder shares the same common text encoder with sketch decoder. The decoding process of fine stage is also similar to sketch decoding (7)-(10), but the input of LSTM is designed as follows:

$$i_t = \begin{cases} \lambda \cdot v_{t-1} + (1-\lambda) emb(e_{t-1}), & \\ \qquad e_{t-1} \text{is determined by} s_{t-1} & \\ emb(e_{t-1}), \quad \text{otherwise} & \end{cases} \tag{11}$$

$$h_t = LSTM(i_t, h_{t-1}) \tag{12}$$

If e_{t-1} is determined by s_{t-1}, the input is the combination of the embedding of e_{t-1} and the sketch encoder's output v_{t-1}. Otherwise, it is set as the embedding of e_{t-1}. We assume the number of sketch decoding outputs is the same shape as the ones from fine decoding. So the e_t and the s_t are aligned one by one. λ is a hyper parameter that controls the combination of v_{t-1} and $emb(e_{t-1})$, ranging from 0 to 1. It indicates how much the fine decoder is guided by sketches. If λ is 1, the process of fine decode will be strictly guided by the sketch. Once the sketch is wrong, the fine decoder has little possibilities to generate a correct result. On the contrary, if λ is 0, the coarse stage will become useless and our model will degrade into one stage model. Like equations (7)-(10), we compute the final probability distribution according to w and s step by step. And $p(e_t|e_{1,\dots,t-1}, s, w)$ is calculated analogously equation (10).

3.4 Model Training

As shown in equation (1), our goal is to maximize the likelihood of sketches and optimize the final results. It can be trained in a supervised way with gold sketches and results. The objective function

aims to maximize L:

$$L = L_{ske} + L_{res} \tag{13}$$

$$L_{ske} = \sum_{(s,w)} \log p(s|w) \tag{14}$$

$$L_{res} = \sum_{(s,w,e)} \log p(e|s,w) \tag{15}$$

(s, w, e) belongs to training pairs. When the model is in testing mode, the final result is computed according $\hat{s} = \arg\max_{s'} p(s'|w)$ and $\hat{e} = \arg\max_{e'} p(e'|w, s)$. s' and e' are sketch candidates and result candidates.

4 Experiments

4.1 Dataset

Text-to-Logic In this task, we conduct our model on GEO dataset, which contains 880 sentences and their corresponding logical queries. Following Dong and Lapata(2018)'s work, we extract the sketches from λ-calculus-based meaning representations. These sketches ignore the arguments and variables and concentrate on operators and logic structures. "$" means an ignored argument and "#" represents an omitted token.

Text-to-Code We chose the Django dataset which has 18805 pairs of natural language expression texts and python codes. We get the sketches by replacing the objects, numbers, functions, and variables with their type names. The symbols of the basic framework are reversed, such as keyword and operators.

MWP Math23k is one of the most popular math word problem datasets which has 23,162 Chinese algebra problems. Each item contains a question in Chinese, a math equation and a answer to the question. To get sketches, we use a placeholder $\langle num \rangle$ to replace the detail numbers in math equations, so sketches only include operators ("+-*/") and $\langle num \rangle$. Another large-scale MWP dataset is Dolphin23K, but its authors just release a construction tool, so we can't get the standard data. All the experiments on it are finished by the dataset's author and they never release the code. Because we can't evaluate the result fairly, we give up conducting our experiment on Dolphin23K.

Examples of original data and sketches of these three datasets are shown in Table 1.

Dataset	Train	Dev	Test
GEO	600	100	180
Django	16000	1000	1805
Math23k	21162	1000	1000

Table 3: Statics of datasets

Task	Emb	Hidden	Epoch	LR
Text2logic	150	250	50	0.005
Text2code	200	300	150	0.005
MWP	128	512	150	0.01

Table 4: Model parameters and training settings

4.2 Preprocess

To compare the result equally, we made our preprocessing in accord with Dong and Lapata (2018)'s experiment as much as possible. For GEO, we followed Dong and Lapata's work, transforming all words into lower type and replacing the entity mentions with a sign and a counting number. And for Django, we chosed to use the processed data given by Yin and Neubig (2017). They tokenized and POS tagged sentences using NLTK. In MWP, we followed Wang et al.'s work. To reduce the influence of OOV, we normalized numbers as the order of their appearance. Examples of some processed cases of Math23k are shown in Table 2, who have same sketches.

4.3 Results

We has compared our improved coarse2fine model with different published models. The optimizer is Adam and many details of training and testing are shown in Table 3 and Table 4. To compare the result equally, we chose the same model parameters as Dong and Lapata (2018) in semantic parsing tasks. Accuracy in this paper (except the MWP) is calculated by comparing the generated result to the gold result(sketches, logic expressions and Python codes), while, in MWP, the accuracy means that whether the math formula predicated by our model it is equal to the given answer.

The results of text2logic are presented in Table 5. Dong and Lapata has experimented the seq2seq and seq2tree method in this task. Seq2tree is a novel framework that has the ability to generate a sequence in hierarchical tree structure. It has an excellent performance in semantic parsing tasks. And Rabinovich et al. has used an abstract syntax tree to generate logic expressions. In our ex-

Model	Acc
Seq2seq (Dong and Lapata, 2016)	84.6%
Seq2tree (Dong and Lapata, 2016)	87.1%
Asn (Rabinovich et al., 2017)	85.7%
Asn+supatt (Rabinovich et al., 2017)	87.1%
One stage	85.0%
Coarse2fine (Dong and Lapata, 2018)	88.2%
Improved coarse2fine	**88.6%**

Table 5: Results of text2logic on GEO

Model	Acc
Seq2seq+unk replacement	45.1%
Seq2tree+unk replacement	39.4%
Lpn+copy (Ling et al.)	62.3%
Snm+copy (Yin and Neubig, 2017)	71.6%
One stage	69.5%
Coarse2fine (Dong and Lapata, 2018)	74.1%
Improved coarse2fine	**76.1%**

Table 6: Results of text2code on Django

periment, the sketch decoder can get 89.3% accuracy and the highest accuracy of logic expression is 88.6% when λ is 0.9. Compared to Dong and Lapata's model, our accuracy rise by 0.4%.

Table 6 reports the results of text2code task. The accuracy of sketches in this task is 77.4%. First two lines are Dong and Lapata's experiments with seq2seq model and seq2tree model. In addition, Ling et al. has designed Latent-Predictor-Network with copy mechanism. And a syntactic neural model also shows good performance in code generation (Yin and Neubig, 2017). As we can see, our model has an outstanding performance in Django dataset. We achieve 76.1% accuracy when λ is 0.6.

The results of MWP are shown in Table 7. As mentioned in Section 2.2, models can be classified into various categories according to the way they get the equation. Classification models get templates through classifier based on Bi-LSTM or Self-Attention (Robaidek et al., 2018). Generation models get equations based on seq2seq models. They are improved with many assistant components. Chiang and Chen (2018) has used a stack, so they could build a tree structure math formula by pushing and popping generated items. Wang et al. (2019) has designed a template-based model which could predict the tree structure formula from bottom to up. In our experiment, sketches have 68.3% accuracy, whereas the accuracy of our

Model	Acc
Self-Attention (Robaidek et al., 2018)	56.8%
Bi-LSTM (Robaidek et al., 2018)	57.9%
Seq2seq+stack (Chiang and Chen, 2018)	65.8%
T-RNN (Wang et al., 2019)	66.9%
Coarse2fine	66.7%
Improved coarse2fine	**67.9%**

Table 7: Result of MWP on Math 23K.

model reaches 67.9% when λ is 0.3.

4.4 Further Test and Analysis

We present the performance with different hyper parameter λ in Table 8. For semantic parsing tasks, our model has a more obvious improvement in text2code. When λ is 1, the model is equal to Dong and Lapata's model and it has 74.9% accuracy. With the decrease of λ, the influence of sketch declines. In the range of (1, 0.6), the lower λ gives the fine decoder more chances to generate a correct result. When λ is lower than 0.6, correct sketches' help will be decreased, which leads to a poor result . The model should keep λ in a appropriate value, so it can take coarse2fine model's advantage and have chances to predict a correct result even the sketch is wrong.

We are the first one to apply coarse2fine method in MWP task. To check whether using sketches makes a good contribution to this task, we suspend the coarse stage and give gold sketches to sketch encoder[1]. The hyper parameter λ is set to 1 so the equation generation will be strictly guided by the sketch. We compared it with one stage model (Wang et al., 2018a). As shown in Figure 2, our model can improve its accuracy highly and shows faster convergence speed after applying gold sketches. Also, its accuracy hits 77.0% under such circumstances.

5 Conclusions

We propose an improved coarse-to-fine generating model in this paper, which takes the advantages of using sketches to help the generating process. When the sketches have mistakes, our model still has a chance to generate a correct result, which will be conducive to the final accuracy. Besides, it is a general framework for many tasks and easy

[1]Giving gold sketches to the sketch encoder in the former two tasks has been accomplished by Dong and Lapata. The accuracies are 93.9% and 83.0%

Task	λ	1	0.8	0.6	0.4	0.2
Text2logic		88.2%	88.5%	87.8%	86.5%	85.2%
Text2django		74.9%	76.0%	76.1%	75.2%	74.9%
MWP		66.7%	66.9%	67.4%	67.8%	67.8%

Table 8: Accuracy of three tasks with different hyper parameters.

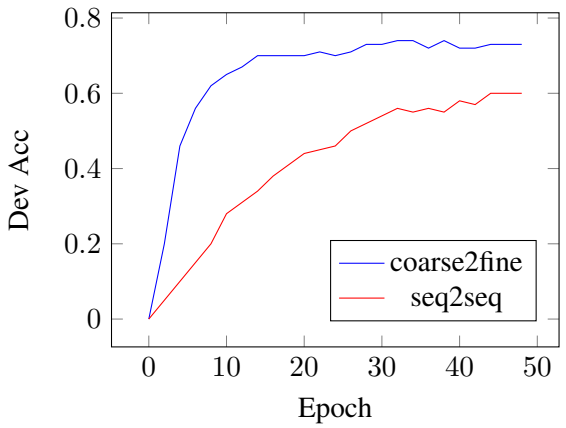

Figure 2: Training record of coarse2fine model and seq2seq model.

to follow. We have conducted our model in many generation tasks (text2logic, text2code, MWP). As a result, compared with the basic model, our accuracy has increased by 0.4%, 2.0%, 1.2% respectively in these three tasks.

Acknowledgments

We thank the anonymous reviewers for their helpful comments on this paper. This work was partially supported by National Natural Science Foundation of China (61572049 and 61876009). The corresponding author is Sujian Li.

References

Jacob Andreas, Andreas Vlachos, and Stephen Clark. 2013. Semantic parsing as machine translation. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 47–52.

Ting-Rui Chiang and Yun-Nung Chen. 2018. Semantically-aligned equation generation for solving and reasoning math word problems. *arXiv preprint arXiv:1811.00720*.

Eunsol Choi, Daniel Hewlett, Jakob Uszkoreit, Illia Polosukhin, Alexandre Lacoste, and Jonathan Berant. 2017. Coarse-to-fine question answering for long documents. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 209–220.

Li Dong and Mirella Lapata. 2016. Language to logical form with neural attention. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 33–43.

Li Dong and Mirella Lapata. 2018. Coarse-to-fine decoding for neural semantic parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 731–742.

Sachin Gangaputra and Donald Geman. 2006. A design principle for coarse-to-fine classification. In *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition-Volume 2*, pages 1877–1884. IEEE Computer Society.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Danqing Huang, Jing Liu, Chin-Yew Lin, and Jian Yin. 2018. Neural math word problem solver with reinforcement learning. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 213–223.

Danqing Huang, Shuming Shi, Chin-Yew Lin, and Jian Yin. 2017. Learning fine-grained expressions to solve math word problems. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 805–814.

Wang Ling, Edward Grefenstette, Karl Moritz Hermann, Tomáš Kociskỳ, Andrew Senior, Fumin Wang, and Phil Blunsom. Latent predictor networks for code generation.

Hongyuan Mei, TTI UChicago, Mohit Bansal, and Matthew R Walter. 2016. What to talk about and how? selective generation using lstms with coarse-to-fine alignment. In *Proceedings of NAACL-HLT*, pages 720–730.

Marco Pedersoli, Andrea Vedaldi, and Jordi Gonzàlez. 2011. A coarse-to-fine approach for fast deformable object detection. In *CVPR 2011*, pages 1353–1360. IEEE.

Maxim Rabinovich, Mitchell Stern, and Dan Klein. 2017. Abstract syntax networks for code generation and semantic parsing. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1139–1149.

Benjamin Robaidek, Rik Koncel-Kedziorski, and Hannaneh Hajishirzi. 2018. Data-driven methods for solving algebra word problems. *arXiv preprint arXiv:1804.10718*.

Lappoon R Tang and Raymond J Mooney. 2000. Automated construction of database interfaces: Integrating statistical and relational learning for semantic parsing. In *Proceedings of the 2000 Joint SIGDAT conference on Empirical methods in natural language processing and very large corpora: held in conjunction with the 38th Annual Meeting of the Association for Computational Linguistics-Volume 13*, pages 133–141. Association for Computational Linguistics.

Lei Wang, Yan Wang, Deng Cai, Dongxiang Zhang, and Xiaojiang Liu. 2018a. Translating a math word problem to a expression tree. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1064–1069.

Lei Wang, Dongxiang Zhang, Jipeng Zhang, Xing Xu, Lianli Gao, Bingtian Dai, and Heng Tao Shen. 2019. Template-based math word problem solvers with recursive neural networks.

Qingyun Wang, Zhihao Zhou, Lifu Huang, Spencer Whitehead, Boliang Zhang, Heng Ji, and Kevin Knight. 2018b. Paper abstract writing through editing mechanism. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 260–265.

Yan Wang, Xiaojiang Liu, and Shuming Shi. 2017. Deep neural solver for math word problems. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 845–854.

Yang Wen, Bin Sheng, Ping Li, Weiyao Lin, and David Dagan Feng. 2019. Deep color guided coarse-to-fine convolutional network cascade for depth image super-resolution. *IEEE Transactions on Image Processing*, 28(2):994–1006.

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

Yingce Xia, Fei Tian, Lijun Wu, Jianxin Lin, Tao Qin, Nenghai Yu, and Tie-Yan Liu. 2017. Deliberation networks: Sequence generation beyond one-pass decoding. In *Advances in Neural Information Processing Systems*, pages 1784–1794.

Pengcheng Yin and Graham Neubig. 2017. A syntactic neural model for general-purpose code generation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 440–450.