"So what should I do next?" – Learning to Reason through Self-Talk

Arianna Yuan
Department of Psychology
Stanford University
Stanford, CA 94305
xfyuan@stanford.edu

Abstract

Goal-oriented dialogs are commonly viewed as communication between two or more interlocutors for exchanging information. However, studies in cognitive sciences have found that people tend to use self-dialogs to guide problem solving. Children trained on self-talk during problem solving, especially self-questioning, demonstrate performance improvement in those tasks. Inspired by these cognitive science research, we propose a learning paradigm in which a dialog agent learns to carry on a goal-oriented conversation with itself as it tries to solve some reasoning task. The conversation is learned from human tutoring dialogs. We use math word problem solving as a concrete example to illustrate the pipeline of this approach, including data collection, model setup and experiments. The specific advantages of introducing self-dialogs to machine reasoning are also discussed.

1 Introduction

Imagine that you are sitting in front of your desk and trying to write your NIPS paper. You probably would ask yourself "what should I include in the Introduction? Should I mention [...] in the related work? Should I use a figure or some pseudocode to present the overall workflow?" While conversations have been traditionally viewed as communication between two or more people, a broader definition of conversations should include self-talk, which constitutes a significant part of our thinking process. Several researchers have proposed that human thinking can be viewed as communication with oneself [1]. Empirical studies have found that teaching students self-questioning skills improves performance in reading comprehension tasks [2] as well as mathematical problems solving [3]. As noted by Özsoy and Ataman, "through the introduction of internal dialogues, self-questioning enables them (the students) to systematically analyze the given information about the problem and manage appropriate cognitive skills." [4].

Inspired by these previous cognitive science studies, we hypothesize that AI agents might also benefit from self-dialogs in problem solving. Particularly, if an agent can learn how to talk to itself from human data, it might acquire the schemas of questions commonly asked under certain task contexts, and use those questions as heuristics for searching a vast solution space. However, from a computational perspective, the benefit of self-talk is less obvious. After all, goal-oriented dialogs usually imply information-asymmetry between interlocutors, i.e., one or both interlocutors have some information that the other person does not share. For example, we query Siri for restaurants recommendations because we do not know what options are available and Siri does not know our preferences. As there will never be information-asymmetry between one and oneself, how can internal dialogs ever be useful for an AI agent?

There are two answers to that. Let us first look at how human benefit from self-talk. Due to our limited working memory capacity, we usually need to select a small portion of information from all

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

the information available in our brain and actively maintain it in our working memory in order to draw useful inference [5]. Asking good questions to ourselves helps us selectively attend to those relevant information. This applies to AI agents as well. Using the terminology in neural networks with attention [6], learning to ask good questions and to answer them helps the agents generate better attention vectors over their memory representations.

The second reason is that, internal dialogs with self-questioning provides us with the scaffolding that ultimately leads to the solution. In human tutoring practice, tutors usually ask simpler questions at the beginning, the answer to which would put the student one step closer to the ultimate solution. In the AI world, when the search space is huge, an AI agent may have difficulty finding the path to the ultimate solution. However, if it learns to produce the type of questions human teachers commonly ask in the same situation, it may be able to find the final solution incrementally. Let us go back to the example mentioned at the beginning of this paper. It might be implausible to plan ahead all the details of a NIPS paper. However, as we give answers to questions such as "what to include in Discussion?", we use pre-learned question schemas to conduct a self-dialog and decompose the big goal to sub-goals that are easier to solve.

Therefore, in the current paper we propose a learning paradigm in which an agent learns to conduct a goal-oriented self-dialog in order to solve some tasks. To illustrate this idea, we use mathematical word problem solving as a concrete example. Automatic math word problem solving has a long tradition in AI [7, 8]. Previous work can be categorized either as symbolic approaches [9] or statistical approaches [10, 11]. Most of the previous methods involve transforming the verbal description of the problems to some formal structures using feature extraction [12]. Only very recently did researchers begin to use deep learning techniques to tackle this problem [13, 14]. For instance, Wang and colleagues applied deep neural networks to generate latent programs that would in turn generate rationals for answers to multiple-choice math problems [14]. Their approach is very relevant to our proposal and we will review that paper in more details in the next section.

Although the symbolic approaches are very data-efficient, they are quite fragile and cannot handle the full complexity of natural language [15]. On the other hand, the end-to-end trained neural network models are hard to interpret. Therefore, we propose to collect human tutoring dialogs for math word problem solving and to train AI agents to jointly learn self-talk and solving the math problems. Since our model requires human generated tutoring dialogs, it is more costly than previous models using only problem descriptions and solutions. However, our goal is to show that we could leverage on self-dialogs to enable end-to-end training for sophisticated machine reasoning tasks and at the same time have control over the interpretability of the model. Also, if the agent is able to imitate human tutors to ask scaffolding questions, it would have a broader impact on intelligent tutoring systems [16].

2 Related Work

Task Compositionality. A large body of machine learning literature has been devoted to task decomposition, i.e., how to solve a task by divide-and-conquer. For instance, in hierarchical reinforcement learning, agents learn to accomplish the final goal by achieving subgoals. However, the automatic discovery of subgoals is notoriously hard. Similarly, in the deep learning literature, researchers have proposed to assemble neural modules networks that solve subtasks in order to solve the original tasks. For instance, in one recent paper by Andreas and colleagues, they use modules that are able to perform "find(tie)", "describe(color)" for visual question answering task [17]. However, those subtasks are specified by the researchers and tailored for the visual question answering task. For both the hierarchical reinforcement learning and the neural modules network approaches, if we can use human tutoring dialogs to automatically induce the appropriate subgoals/subtasks, the human labor and expertise required to engineer the proper subgoals will be dramatically reduced.

Explanability of neural networks. Several research efforts have been devoted to understanding how neural network works. Recently developed methods include representation erasure [18], translating distributed message representation in neural models to natural languages [19], selecting text segments as rationals for neural predictions in sentiment analysis and question retrieval [20], generating natural language rationals for answers to complicated math problems [14]. Our approach is mostly aligned with those studies that aim at interpreting the inner mechanisms of neural networks

Table 1: A problem from the MAPWS repository and a sample dialog.

	Faye and her mom were picking carrots from their garden. Faye picked 23 and her mother picked 5. If only 12 of the carrots were good, how many bad carrots did they have?
Q	How many carrots did Faye and her mom pick together?
A	23 + 5 = 28
Q	How many of them were good?
A	12
Q	If the carrots were either good or bad, how many bad carrots did they have?
A	28 - 12 = 16

Table 2: A problem from the Dolphin18K Dataset and a sample dialog.

	Larry Junior is 27 years younger than Larry Senior and 29 years older than Little Larry. their ages total 100. What is each Larry's age?
Q:	If we denote Larry Juniors' age as x, what is Larry Senior's age?
A:	x + 27
Q:	What is Little Larry's age?
A:	x - 29
Q:	Can you sum them up to get their total ages?
A:	x + x+ 27 + x - 29
Q:	Is their total ages given in the problem?
A:	Yes
Q:	Can you assign the value to the expression of their total ages?
A:	x + x+ 27 + x - 29 = 100

via natural languages, as we use self-dialogs generated by the models to understand how they perform the tasks.

Neural Program Induction. Our model is also closely related to the neural program synthesis literature. Since the introduction of the Neural Programmer Interpreter [21], using deep neural networks to induce programs has been a hot topic in machine learning [22, 23]. Our method build upon Wang and colleagues' previous work on program induction for math word problem solving [14]. One of the key differences is that their goal is to find latent programs that generate the rational, but our goal is to find latent programs that generate self-dialogs.

3 Task

In this section we provide an overview of the Amazon Mechanical Turk experiment that will be used to collect the human tutoring dialog data. Turkers are pre-screened to ensure that they have good enough mathematical ability to perform the task. We then give qualified Turkers math word problems from the MAWPS repository [24] and the Dolphin18K dataset [15]. These datasets have been used in previous automatic word problem solving studies. We pair Turkers into dyads [25] and assign one of them the role of the Teacher and the other one the role of the Student. We instruct both the Teacher and the Student to simulate a tutoring scenario. Particularly, we ask the Teacher to ask scaffolding questions and the Student to give correct responses to the Teacher's questions. In the end we ask the Student to provide the final numeric answer to the problem. We also require that the last response in the dialog is the equation that solves the problem. Trials in which Turkers give wrong final numeric answers will be excluded. Turkers will be shown a couple of dialog examples before the real experiment begins. A simple example from the MAWPS repository and a more complicated one from the Dolphin18K Dataset are shown in Table 1 and Table 2, respectively.

4 Model Architecture and Experiments

We extend a previous model developed by Das and colleagues for visual dialog generation in a reference game [26]. In that paper, the goal is to generate a dialog between a Q-bot and an A-bot

so that the Q-bot can identify the target image from a lineup of candidate images. In our setup, the goal is to generate a self-dialog grounded in the problem description so that in the end the agent can output a correct equation for the problem. In the rest of the paper, we will use "questions" to refer to the questions asked in a self-dialog and "problems" to refer to the math word problems.

The model architecture is shown in Figure 1. The model is composed of several hierarchical encoder-decoder RNNs with attention. The decoders used in our model are not classic LSTM RNNs as in [26], since they output latent programs as in [14] rather than English word sequences. The execution of the latent programs results in natural language sentences. We next describe each model component in more details.

The History Encoder and the Reasoning Encoder. The History Encoder is a LSTM RNN that encodes the state S_t. Since this is an internal dialog, both the History Encoder of the Q-bot and the A-bot have access to the encoding of the math word problems e^M. At each round t of the dialog, the state $S_t^Q = [e^M, R_1, \ldots, R_{t-1}]$, where R_t is the output of the Reasoning Encoder that encodes the $< q_t, a_t >$ pairs at round t. Note that in [26] they have separate R_t for the Q-bot (R_t^Q) and the A-bot (R_t^A), due to the information-asymmetry between the two agents. This is no longer necessary in our self-talk scenario. However, it still worth having a different state representation for the A-bot since the newly generated question needs to be integrated into the A-bot's dialog history. Therefore, $S_t^A = [e^M, R_1, \ldots, R_{t-1}, q_t]$.

Word Problem Encoder, Question Encoder and Answer Encoder. We use LSTM RNNs for these encoders with the same hyper-parameters described in [26].

Question Decoder, Answer Decoder and Equation Decoder. For the decoders, we use the approach described in [14]. Let us take the Answer Decoder as an example. At each time step i, the model needs to output an operation (e.g., identity function, arithmetic function, etc.), its corresponding arguments, and a decision on whether the outcome of executing that operation should be added to the output y^A or its memory m^A. To generate the jth argument, a latent variable $q_{i,j} \in$\{SOFTMAX,COPY-INPUT,COPY-OUTPUT\} is introduced. If $q_{i,j} =$ SOFTMAX, then the jth argument is sampled from the vocabulary. If $q_{i,j} =$ COPY-INPUT, the argument is obtained by copying an item from the input, i.e., the dialog history. If $q_{i,j} =$COPY-OUTPUT, then the argument is selected from the output y^A generated so far or the memory m^A. After the decoders has output the operation and its corresponding argument, depending on the decision variable r_i, the outcome of the execution of that operation is either added to the output y^A or its memory m^A. For more details about the decoder we refer the reader to [14]. For the Question Decoder and the Equation Decoder, we use similar networks as the Answer Decoder.

Word Problem Decoder We add another training objective, i.e., the $< q_t, a_t >$ pair at each round should be consistent with the original problem description. To achieve this goal, we ask the Q-bot to recover the original word problem from the dialog history. We feed the recovered representation of the word problem at round t (e_t^M), the encoding of the problem (e^M) as well as the encodings of some distractor problems (e^{M_D}) to a discriminator network. We require the discriminator network to distinguish the true problem from the distractors given the information recovered from the dialog history, i.e., maximize the probability $P(M|e^M, e^{M_D}, e_t^M)$ that it chooses the true problem in contrast to the alternatives given the information recovered from the dialog. The details of the discriminator network are shown in Equation 1. We describe how we generate the distractor problems in the next section.

$$
\begin{aligned}
s_M &= \omega^T \phi(W_1 e_t^M + W_2 e^M) \\
s_{M_d} &= \omega^T \phi(W_1 e_t^M + W_2 e^{M_d}) \\
P(M|e^M, e^{M_D}, e_t^M) &= \frac{e^{s_M}}{(\sum_{d \in D} e^{s_{M_d}}) + e^{s_M}}
\end{aligned}
\tag{1}
$$

, where $y = \phi(x)$ is a rectified linear activation function.

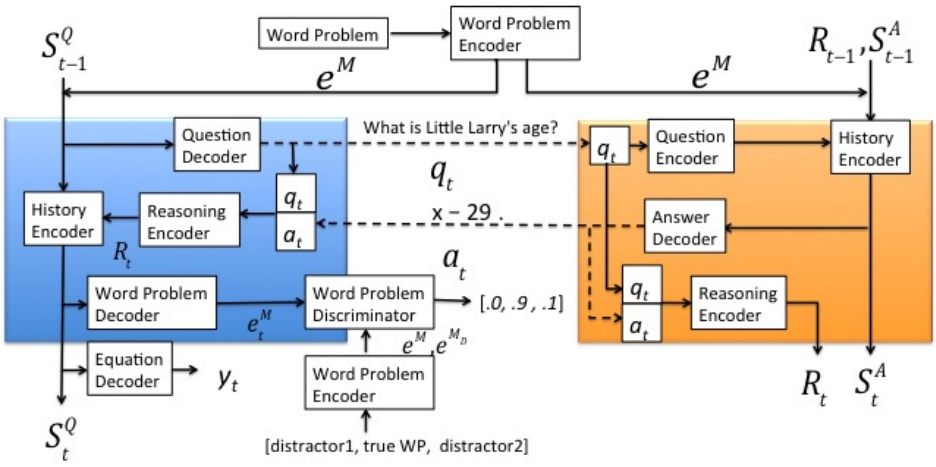

Figure 1: Model Architecture

5 Training and Evaluation

5.1 Training

During the training phase, we use the same objective as in [14], i.e., given input x we optimize the marginal probability over all programs in Z that would generate y:

$$p(y|x) = \sum_{z \in Z} \mathbb{1}(e(z) == y) p(z|x) \tag{2}$$

, where $\mathbb{1}(e(z) == y)$ is the characteristic function that takes value 1 if the execution of z is y and 0 otherwise. The detailed method of optimizing this probability function can be found in [14]. There is another loss term in our model, which is the cross-entropy loss of identifying the true word problem from a list of candidate problems including the true problem and the distractor problems. We use two strategies to generate distractor problems. The first one is name-swapping. We use the Stanford Named Entity Recognizer to find word pairs with the same NER category (e.g., Person) and then swap them. For example, "Mary gave Tom 5 apples" will become "Tom gave Mary 5 apples". Typically this will change the meaning of the sentences and the corresponding equations. The second strategy is swapping numbers in the original problems, which will also result in changes of the corresponding equations. Using these two strategies, we can generate distractor problems that look superficially similar to the original ones, but semantically are quite different. The dialog history should encode enough information to enable the Word Problem Discriminator Network to distinguish the original word problem from the distractor problems given the dialog history.

5.2 Evaluation

We propose three ways to evaluate our model performance. The first one is the average sentence level perplexity and the BLEU score [27] for questions and answers. The second one is the evaluation metrics in [17], which is only for the evaluation of the answers. In that paper, Das et al. generate 100 candidate answers including the correct answer, the top-30 most frequent answers in the training dataset, some plausible answers that are answers to questions similar to the current question and some random answers. They use the log-likelihood scores in their generative decoder networks to rank candidate answers and report the mean reciprocal rank of the ground truth answers, recall@k and mean rank of the ground truth answers. The last evaluation metrics is the solution accuracy. We use a solver to solve the final equation that the system outputs and calculate the mean accuracy of the final numeric answers.

Several model ablation experiments will be conducted to test the contribution of each component of the training data, especially whether the model benefits from self-talk. These ablation experiments include a) Removing dialog training data and generating final equations directly from the problem

descriptions. b) When generating answers, use only the current question and the problem description, with no dialog history. c) Removing latent program induction and directly generating questions and answers by sampling tokens from the vocabulary. d) Some combinations of the above model ablations. We expect the model to have better performance when self-dialog and latent program induction are included.

For the error analysis, we will look at whether the errors in the equation are consistent with incorrect answers given by the model during the self-talk. This would give us a sense of why the model outputs some wrong equations. We will also examine whether the attention vectors are tuned by the questions, i.e., whether the model learns to attend to relevant information in the dialog history and the problem description when prompted with different questions.

6 Discussion

In this paper we propose a new learning paradigm in which a dialog agent learns to self-talk in order to become a better problem solver. Although we use math word problem solving as an example to illustrate the potential of introducing self-dialogs in machine reasoning, the application of this can be very broad. For instance, we can envision a trouble-shooting agent that learns to self-talk from human expert dialogs in order to figure out technical issues of operation systems. This approach would be particularly beneficial when the search space of solutions are hierarchical and huge.

Also, in the current paper we only use supervised learning for our model. However, we could use reinforcement learning to encourage more flexibility in the form of the final equations. This is particularly useful for automatic math word problem solving since equations that have the exact same solutions may take various forms. Forcing the agent to generate the ground truth equation, which takes only one form, is obviously not the best idea. One solution is to frame the decoders as policy network and define "generating an equation that yields the correct solution" as rewards. We can then update the policy parameters using the REINFORCE algorithm [28].

References

[1] Anna Sfard. *Thinking as Communicating*. Cambridge University Press, 2008.

[2] Laurice M. Joseph, Sheila Alber-Morgan, Jennifer Cullen, and Christina Rouse. The effects of self-questioning on reading comprehension: A literature review. *Reading & Writing Quarterly*, 32(2):152–173, 2015.

[3] Marjorie Montague. Self-regulation strategies to improve mathematical problem solving for students with learning disabilities. *Learning Disability Quarterly*, 31(1):37–44, 2008.

[4] Gokhan Ozsoy and Aysegul Ataman. The effect of metacognitive strategy training on mathematical problem solving achievement. *International Electronic Journal of Elementary Education*, 1(2):68–83, 2009.

[5] Henry Gleitman, James Gross, and Daniel Reisberg. *Psychology*. Norton & Company, Inc. - New York, 2011.

[6] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[7] Daniel G Bobrow. Natural language input for a computer problem solving system. 1964.

[8] Christian Liguda and Thies Pfeiffer. Modeling math word problems with augmented semantic networks. In *International Conference on Application of Natural Language to Information Systems*, pages 247–252. Springer, 2012.

[9] Yefim Bakman. Robust understanding of word problems with extraneous information. *arXiv preprint math/0701393*, 2007.

[10] Nate Kushman, Luke S. Zettlemoyer, Regina Barzilay, and Yoav Artzi. Learning to automatically solve algebra word problems. In *Proceedings of the Annual Meeting on Association for Computational Linguistics*, 2014.

[11] Mohammad Javad Hosseini, Hannaneh Hajishirzi, Oren Etzioni, and Nate Kushman. Learning to solve arithmetic word problems with verb categorization. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 523–533, 2014.

[12] Shuming Shi, Yuehui Wang, Chin-Yew Lin, Xiaojiang Liu, and Yong Rui. Automatically solving number word problems by semantic parsing and reasoning. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1132–1142, 2015.

[13] Yan Wang, Xiaojiang Liu, and Shuming Shi. Deep neural solver for math word problems. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 856–865, 2017.

[14] Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. Program induction by rationale generation: Learning to solve and explain algebraic word problems. *arXiv preprint arXiv:1705.04146*, 2017.

[15] Danqing Huang, Shuming Shi, Chin-Yew Lin, Jian Yin, and Wei-Ying Ma. How well do computers solve math word problems? large-scale dataset construction and evaluation. In *Proceedings of the Annual Meeting on Association for Computational Linguistics*, 2016.

[16] Benjamin D. Nye, Arthur C. Graesser, and Xiangen Hu. AutoTutor and family: A review of 17 years of natural language tutoring. *International Journal of Artificial Intelligence in Education*, 24(4):427–469, 2014.

[17] Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Neural module networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 39–48, 2016.

[18] Jiwei Li, Will Monroe, and Dan Jurafsky. Understanding neural networks through representation erasure. *arXiv preprint arXiv:1612.08220*, 2016.

[19] Jacob Andreas, Anca Dragan, and Dan Klein. Translating neuralese. *arXiv preprint arXiv:1704.06960*, 2017.

[20] Tao Lei, Regina Barzilay, and Tommi Jaakkola. Rationalizing neural predictions. *arXiv preprint arXiv:1606.04155*, 2016.

[21] Scott Reed and Nando De Freitas. Neural programmer-interpreters. *arXiv preprint arXiv:1511.06279*, 2015.

[22] Danfei Xu, Suraj Nair, Yuke Zhu, Julian Gao, Animesh Garg, Li Fei-Fei, and Silvio Savarese. Neural task programming: Learning to generalize across hierarchical tasks. *arXiv preprint arXiv:1710.01813*, 2017.

[23] Jacob Devlin, Rudy Bunel, Rishabh Singh, Matthew Hausknecht, and Pushmeet Kohli. Neural program meta-induction. *arXiv preprint arXiv:1710.04157*, 2017.

[24] Rik Koncel-Kedziorski, Subhro Roy, Aida Amini, Nate Kushman, and Hannaneh Hajishirzi. Mawps: A math word problem repository. In *HLT-NAACL*, pages 1152–1157, 2016.

[25] Robert XD Hawkins. Conducting real-time multiplayer experiments on the web. *Behavior Research Methods*, 47(4):966–976, 2015.

[26] Abhishek Das, Satwik Kottur, José MF Moura, Stefan Lee, and Dhruv Batra. Learning cooperative visual dialog agents with deep reinforcement learning. *arXiv preprint arXiv:1703.06585*, 2017.

[27] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the Annual Meeting on Association for Computational Linguistics*, pages 311–318. Association for Computational Linguistics, 2002.

[28] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.