Learning Multi-Step Reasoning by Solving Arithmetic Tasks

Tianduo Wang and Wei Lu
StatNLP Research Group
Singapore University of Technology and Design
{tianduo_wang, luwei}@sutd.edu.sg

Abstract

Mathematical reasoning is regarded as a necessary ability for Language Models (LMs). Recent works demonstrate large LMs' impressive performance in solving math problems. The success is attributed to their Chain-of-Thought (CoT) reasoning abilities, i.e., the ability to decompose complex questions into step-by-step reasoning chains, but such ability seems only to emerge from models with abundant parameters. This work investigates how to incorporate relatively small LMs with the capabilities of multi-step reasoning. We propose to inject such abilities by continually pre-training LMs on a synthetic dataset **MsAT** which is composed of **M**ulti-**s**tep **A**rithmetic **T**asks. Our experiments on math word problem tasks show the effectiveness of our method in enhancing LMs' math reasoning abilities.[1]

1 Introduction

Making Language Models (LMs) perform mathematical reasoning is a valuable, yet challenging research objective [5, 3]. Recently, we have witnessed large LMs' impressive performance on math reasoning tasks via *chain-of-thought* prompting [21]. This method elicits large LM's ability to decompose a complex problem into several intermediate steps. However, it is believed that such ability only emerges from sufficiently large models (empirically more than 100B parameters) [21]. In this paper, we examine how to incorporate moderate-sized LMs, e.g., RoBERTa [11], with such multi-step reasoning ability via continual pre-training to improve the performance on math problems.

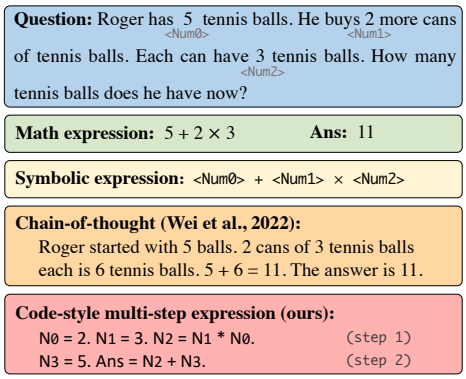

Figure 1: A math word problem example with different kinds of answers. In **Question**, <Num0>, <Num1>, and <Num2> are special tokens used for masking numbers.

Correctly understanding numbers is a pre-requisite of mathematical reasoning abilities. But Wallace et al. [19] shows that medium-sized LMs have a deficiency in numerical comprehension. To overcome this issue, previous works inject numerical reasoning skills into LMs following two approaches. The first is masking numbers with special tokens, and generating symbolic expressions with a structured neural decoder [24, 7]. An example of such expression is provided in Fig. 1. The second strategy continually pre-trains LMs on synthetic numerical tasks, which requires models to learn how to perform computation involving numbers [4, 16].

[1] Our code and data are released at https://github.com/TianduoWang/MsAT.

37th Conference on Neural Information Processing Systems (NeurIPS 2023) Workshop on MATH-AI.

However, both approaches suffer from critical limitations. For symbolic methods, they neglect the information carried by the numbers, which could provide crucial hints for solving math problems [23, 10]. As for continual pre-training methods, LMs' arithmetic skills are not reliable. Previous works indicate that such skills are highly influenced by the training data [17] and hard for extrapolation [19].

Motivated by these shortcomings, we propose to first pre-train LMs on a synthetic dataset called MSAT (Multi-step Arithmetic Tasks) before fine-tuning on downstream tasks. To make sure LMs capture the information carried by the numbers, we keep the numbers in the questions instead of masking them. To avoid making LMs conduct computation internally, MSAT encourages LMs to generate a series of intermediate steps that can be executed by an external Python interpreter. Experiments on three math word problem datasets with two backbone models demonstrate the effectiveness of our method in enhancing LMs' math reasoning performance.

2 Method

Our method appends a continual pre-training stage before fine-tuning LMs on downstream tasks. The continual pre-training serves two purposes: first, we tokenize numbers digit-by-digit to improve LMs' numerical comprehension; second, we make LMs learn multi-step reasoning skills from the proposed synthetic task.

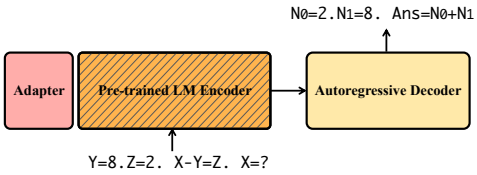

Figure 2: An illustration of the continual pre-training process on our Seq2Seq model. We attach adapter modules to each layer of LM encoder and fix LM's parameters (shaded area) during pre-training. Tokens $\texttt{N}_0$, $\texttt{N}_1$, `Ans` in the output are the variable names only used by the decoder.

2.1 Digit tokenization for numbers

Sub-word tokenization methods, e.g., byte pair encoding (BPE) [18], is one of the reasons why LMs poorly understand numbers [19]. BPE-based tokenizers split text based on the token frequency in the training corpus, which can be counter-intuitive when dealing with numbers. For example, numbers "520" and "521" will be tokenized into ["520"] and ["5", "21"] respectively by the `RoBERTaTokenizer` of the `Transformers` library [22]. Such inconsistent tokenization strategy for numbers undermines LM's numerical understanding ability. Hence, we tokenize numbers digit-by-digit for both pre-training and fine-tuning.

2.2 Multi-step Arithmetic Tasks (MSAT)

The core of our method is the synthetic task MSAT where LMs can learn multi-step reasoning skills. MSAT can be formulated as a Seq2Seq task: the input of a MSAT example describes an arithmetic question, while the output is a reasoning chain leading to the answer. Specifically, each input sequence is composed of three components: *question context*, *equation*, and *question variable*. Equation is a sequence of symbols and operators ($+$, $-$, $\times$, $\div$, $=$) that builds equality relationship between symbols. Given an equation, only one of the symbols is set as the question variable, while other symbols will be listed in question context with their numerical values.

The output sequence of MSAT is constructed in a code-style multi-step reasoning format. Each step consists of two sub-steps: *variable assignment* and *calculation*. In variable assignment, numbers appear in the input sequence are assigned to the variable names that are exclusive for decoder. In calculation, a new variable is generated from the calculation of the existing variables. This makes our outputs become executable Python code so that the numerical answer can be calculated by an external Python interpreter. Both inputs and outputs of MSAT are generated purely automatically. Details about the construction of MSAT are provided in Appendix A.1.

2.3 Pre-training via adapter-tuning

Directly training on synthetic data that are largely different from the natural language corpus harms LMs' language prowess [4]. Therefore, we adopt a two-stage tuning strategy [20] to inject reasoning skills into LMs. Specifically, we perform adapter-tuning [6] on MSAT and then jointly fine-tune adapter and LM backbone on downstream tasks. It mitigates catastrophic forgetting because LM's original parameters are largely preserved during adapter-tuning [6]. Fig. 2 shows an overview of

Table 1: Accuracy (%) comparison between large language models (LLMs), backbone model baselines, and our method. Δ: performance gap compared with the symbolic mask baselines.

Model	MAWPS		ASDiv-A		SVAMP	
	Acc.	Δ	Acc.	Δ	Acc.	Δ
Large language models [21]	(PaLM 540B)		(code-davici-002)		(PaLM 540B)	
w/ Chain-of-Thought prompting	93.3		80.4		**79.0**	
Seq2Seq models						
RoBERTaGen [9]						
w/ symbolic masks	88.4		72.1		30.3	
w/ digit tokenization	84.1	(-4.3)	71.9	(-0.2)	27.6	(-2.7)
MsAT-RoBERTaGen (Ours)	**91.6**	(+3.2)	**81.8**	(+9.7)	**39.8**	(+9.5)
DAG structured models						
DeductReasoner [7]						
w/ symbolic masks	92.0		85.0		45.0	
w/ digit tokenization	91.6	(-0.4)	84.1	(-0.9)	44.4	(-0.6)
MsAT-DeductReasoner (Ours)	**94.3**	(+2.3)	**87.5**	(+2.5)	**48.9**	(+3.9)

the proposed pre-training method. We adopt models with encoder-decoder architecture to verify the effectiveness of our method. More details about the backbone models are provided in Section 3.1.

3 Experiments

3.1 Experimental setup

Existing datasets We consider three commonly-used MWP datasets: MAWPS [8], ASDiv-A [13], and SVAMP [15]. We report five-fold cross-validation results for both MAWPS and ASDiv-A and test set accuracy for SVAMP following previous practice [9, 7]. We provide more details about these datasets in Appendix A.2.

Models We consider both sequence-to-sequence (Seq2Seq) models and directed acyclic graph (DAG) structured models as our backbone models. For Seq2Seq model, we choose RoBERTaGen [9], an encoder-decoder model with RoBERTa as the encoder combined with a Transformer decoder. For DAG structured model, we choose DeductReasoner [7] that combines RoBERTa with a DAG decoder. In their original implementation, both models replace numbers with symbolic mask tokens. Hence, we additionally consider a baseline for each backbone model that uses actual numbers with digit tokenization. We name the models that are based on these two backbone models and pre-trained with our method as MsAT-RoBERTaGen and MsAT-DeductReasoner respectively. We also compare our models to large LMs, e.g., PaLM [2] and Codex [1], with chain-of-thought prompting [21]. All models are evaluated via greedy decoding. More implementation details, e.g., training hyperparameters, are provided in Appendix C.

3.2 Main results

Table 1 compares our models with backbone model baselines and large LMs. On all datasets, digit tokenization baselines consistently perform worse than their symbolic mask counterparts, indicating the deficiency of the numeracy comprehension of the original RoBERTa model. However, the models trained with MsAT surpass both baselines by a large margin, which demonstrates the effectiveness of our pre-training method.

Compare with large language models We also observe that, on relatively simple tasks, i.e., MAWPS and ASDiv-A, RoBERTa-based models can outperform large LMs. But for the more challenging task SVAMP, there is still a large performance gap. We believe this is because SVAMP requires models to have a better understanding of natural languages. Jie et al. [7] also reports that varying LM encoders results in significant performance disparities on SVAMP, indicating that SVAMP performance is closely tied to model's natural language capabilities.

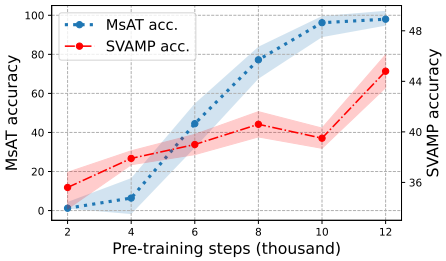

Figure 3: Performance on MSAT and SVAMP with respect to the pre-training steps. Results are obtained from 3 different runs.

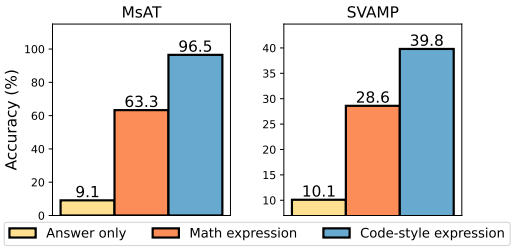

Figure 4: Comparison between different output expression formats. Results are obtained from our Seq2Seq model (with code-style expressions) and its variants.

4 Pre-training analysis

In this section, we provide a careful analysis of our pre-training method from various perspectives to understand why it works. Additional analysis on the difficulty of MSAT and adapter-tuning is provided in Appendix B.

Pre-training task performance We visualize how the performance of pre-training task MSAT and one of the MWP tasks SVAMP changes with pre-training steps in Figure 3. It can be observed that the performance on both synthetic and natural language tasks tends to improve gradually as the number of pre-training steps increases. Figure 3 demonstrates that LMs are capable of learning multi-step reasoning gradually from the synthetic task MSAT. The acquired multi-step reasoning ability can subsequently be transferred to the downstream MWP solving tasks, enhancing performance during the fine-tuning phase.

Effect of producing intermediate steps While it is a common practice to train LMs towards directly producing the numerical answers of the arithmetic questions [4, 16], a recent work shows that LMs' arithmetic skills are not reliable [17]. To explore whether LMs can learn reasoning skills from MSAT without intermediate steps, we pre-train LMs on a variant of MSAT by replacing step-by-step output sequences with only numerical answers. Fig. 4 compares this model (answer only) with our model (code-style). Its poor performance on both MSAT and SVAMP confirms the necessity of producing intermediate reasoning steps during pre-training.

Structured code-style expression We next investigate the importance of applying the structured code-style reasoning expressions by comparing it with the less formatted math expressions. We argue that, compared with math expressions that only contain numbers and operators, our code-style expressions are more suitable for multi-step reasoning due to the structure information in the output sequences. Our experiments in Fig. 4 demonstrate the superiority of the code-style output expressions. We can see that models with math expressions perform consistently worse than models with code-style multi-step reasoning format on both pre-training task MSAT and MWP solving task SVAMP.

5 Conclusion

We propose a novel synthetic pre-training task, MSAT, to incorporate LMs with multi-step reasoning skills that improve performance on MWP tasks. This pre-training task encourages LMs to generate intermediate reasoning steps instead of predicting final numerical answers directly. Our experiments show that the proposed method is effective in improving the moderate-sized LM's performance on MWP solving tasks.

References

[1] Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. d. O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., et al. (2021). Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

[2] Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., et al. (2022). Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*.

[3] Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., et al. (2021). Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

[4] Geva, M., Gupta, A., and Berant, J. (2020). Injecting numerical reasoning skills into language models. In *Proceedings of ACL*.

[5] Hendrycks, D., Burns, C., Kadavath, S., Arora, A., Basart, S., Tang, E., Song, D., and Steinhardt, J. (2021). Measuring mathematical problem solving with the math dataset. In *Proceedings of NeurIPS*.

[6] Houlsby, N., Giurgiu, A., Jastrzebski, S., Morrone, B., De Laroussilhe, Q., Gesmundo, A., Attariyan, M., and Gelly, S. (2019). Parameter-efficient transfer learning for nlp. In *Proceedings of ICML*.

[7] Jie, Z., Li, J., and Lu, W. (2022). Learning to reason deductively: Math word problem solving as complex relation extraction. In *Proceedings of ACL*.

[8] Koncel-Kedziorski, R., Roy, S., Amini, A., Kushman, N., and Hajishirzi, H. (2016). Mawps: A math word problem repository. In *Proceedings of NAACL*.

[9] Lan, Y., Wang, L., Zhang, Q., Lan, Y., Dai, B. T., Wang, Y., Zhang, D., and Lim, E.-P. (2021). Mwptoolkit: An open-source framework for deep learning-based math word problem solvers. *arXiv preprint arXiv:2109.00799*.

[10] Liang, Z., Zhang, J., Wang, L., Qin, W., Lan, Y., Shao, J., and Zhang, X. (2022). MWP-BERT: Numeracy-augmented pre-training for math word problem solving. In *Findings of NAACL*.

[11] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

[12] Loshchilov, I. and Hutter, F. (2019). Decoupled weight decay regularization. In *Proceedings of ICLR*.

[13] Miao, S.-y., Liang, C.-C., and Su, K.-Y. (2020). A diverse corpus for evaluating and developing english math word problem solvers. In *Proceedings of ACL*.

[14] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Proceedings of NeurIPS*.

[15] Patel, A., Bhattamishra, S., and Goyal, N. (2021). Are NLP models really able to solve simple math word problems? In *Proceedings of NAACL*.

[16] Pi, X., Liu, Q., Chen, B., Ziyadi, M., Lin, Z., Gao, Y., Fu, Q., Lou, J.-G., and Chen, W. (2022). Reasoning like program executors. In *Proceedings of EMNLP*.

[17] Razeghi, Y., Logan IV, R. L., Gardner, M., and Singh, S. (2022). Impact of pretraining term frequencies on few-shot reasoning. In *Proceedings of ICML*.

[18] Sennrich, R., Haddow, B., and Birch, A. (2016). Neural machine translation of rare words with subword units. In *Proceedings of ACL*.

[19] Wallace, E., Wang, Y., Li, S., Singh, S., and Gardner, M. (2019). Do NLP models know numbers? probing numeracy in embeddings. In *Proceedings of EMNLP-IJCNLP*.

[20] Wang, T. and Lu, W. (2022). Differentiable data augmentation for contrastive sentence representation learning. In *Proceedings of EMNLP*.

[21] Wei, J., Wang, X., Schuurmans, D., Bosma, M., Chi, E., Le, Q., and Zhou, D. (2022). Chain of thought prompting elicits reasoning in large language models. In *Proceedings of NeurIPS*.

[22] Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., et al. (2020). Transformers: State-of-the-art natural language processing. In *Proceedings of EMNLP*.

[23] Wu, Q., Zhang, Q., Wei, Z., and Huang, X. (2021). Math word problem solving with explicit numerical values. In *Proceedings of ACL-IJCNLP*.

[24] Xie, Z. and Sun, S. (2019). A goal-driven tree-structured neural model for math word problems. In *Proceedings of IJCAI*.

A Additional information about datasets

In this section, we provide additional details about the datasets that we used in the experiments. The dataset statistics is provided in Table 2.

A.1 Construction of MSAT

The proposed MSAT is a synthetic Seq2Seq task where the inputs describe arithmetic questions and outputs are the solutions represented by a code-style multi-step reasoning format. Both inputs and outputs of MSAT can be generated automatically. To construct an example of MSAT, we first generate the input sequence and then produce the output solution accordingly. In all, we generate 85,000 examples and split them into 80,000 and 5,000 for training and evaluation respectively.

Input sequence construction We start by preparing a set of equation templates and each equation template contains no more than 3 binary operators ($+$, $-$, $\times$, and $\div$). By enumerating the possible combinations of operators, we obtain $4 + 4^2 + 4^3 = 84$ equation templates in total. The first step to construct an input arithmetic question is to instantiate an equation from an equation template. For example, given an equation template "`<Num0> + <Num1> = <Num2>`", we assign each variable a value that makes the equality hold and a variable name selected from the capitalized letters. The numbers in the questions are sampled from `0` to `10,000`. The last step is to randomly pick a variable as the question variable. Therefore, the resulting input arithmetic question may look like: "`A=1. C=3. A+B=C. B?`"

Output sequence construction Given an equation and a question variable, the output is first constructed as a math expression leading to the value of the question variable. Notice that an equation can be represented as a binary tree where the variables are the terminal nodes and operators are the non-terminal nodes. Hence, the output can be produced by a "tree inversion" algorithm (see Figure 5) from an equation and a question variable.

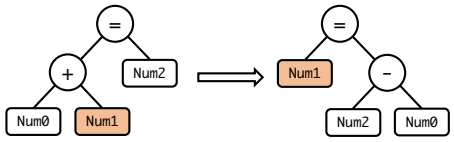

Figure 5: An illustration of the "tree inversion" algorithm that produces an output expression from an arithmetic question. The question variable is highlighted.

A.2 Existing datasets

MAWPS [8] It is a popular benchmark dataset for math word problems. We use the five-fold split provided by Lan et al. [9] for evaluation.

ASDiv-A [13] This is an English math word problem task containing various linguistic patterns and problem categories. We obtain the data and five-fold split from Patel et al. [15].

Table 2: Existing dataset statistics.

Dataset	# Data	Avg. input length	Avg. output reasoning steps
MAWPS	1,987	30.3	1.4
ASDiv-A	1,217	32.3	1.2
SVAMP	1,000	34.7	1.2

SVAMP [15] It is a challenge set created for MWP model robustness evaluation. The examples in SVAMP are from ASDiv-A with deliberately designed variations. Such variations include: changing questions, adding irrelevant information, etc. Following the evaluation protocol suggested by Patel et al. [15], we train our models over 3,138 training examples from a combination of MAWPS and ASDiv-A.

B Ablation studies

Difficulty level of MSAT Leveraging synthetic data for pre-training provides the advantage of enabling highly customizable difficulty levels for the training data. Here we define the difficulty level of a reasoning task as the averaged reasoning steps that are required to solve the problems. From Fig. 6, we see that pre-training LMs on MSATs that are harder than downstream tasks generally leads to better results. It's important to note that, broadly speaking, the difficulty level of a reasoning task,

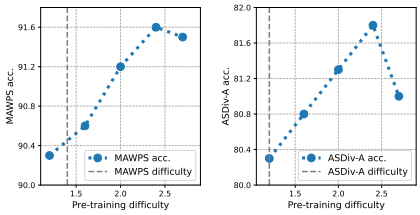

Figure 6: Performance on MAWPS and ASDiv-A with respect to pre-training difficulty. The difficulty levels of two MWP tasks are also added for reference.

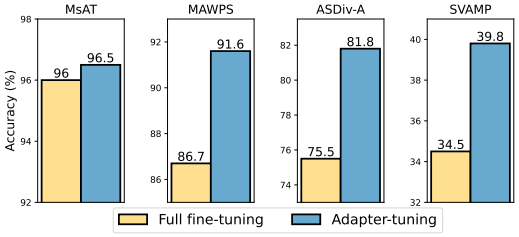

Figure 7: MSAT and downstream task performance comparison between full fine-tuning and adapter-tuning during pre-training.

particularly those involving natural language, is not solely determined by the number of reasoning steps. One example is that, though both ASDiv-A and SVAMP have an averaged reasoning steps of 1.2 (see Table 2), SVAMP is considered more difficult as it requires high-level natural language understanding [15].

Perform adapter-tuning on MSAT Tuning all parameters of LM encoders on synthetic data that are largely different from the pre-training corpus may lead to catastrophic forgetting [4]. To explore the importance of performing adapter-tuning on MSAT, we create a variant of our method in which we perform full fine-tuning on MSAT. We compare this variant with our models in Fig. 7. It can be observed that both full fine-tuning and adapter-tuning can achieve good performance on MSAT, but adapter-tuning outperforms fine-tuning on all downstream MWP datasets, which demonstrates the benefits of performing adapter-tuning on MSAT.

C Implementation details

Our method is implemented in Python 3.8 with HuggingFace's Transformers [22] and PyTorch [14] libraries. All experiments can be conducted on one NVIDIA RTX 6000 GPU with 22 GB memory. For our MSAT-ROBERTAGEN and MSAT-DEDUCTREASONER, we build the backbone models following the implementation provided by Lan et al. [9] and Jie et al. [7] respectively. The encoders for both models are initialized with the pre-trained weights of RoBERTa$_{\texttt{base}}$. The adapter modules [6] are added to each layer of the encoders with a bottleneck dimension of 64. More details about the mdoel architectures are provided in Table 3. We provide the hyperparameters for both pre-training and fine-tuning in Table 4.

Table 3: Architectures hyperparameters.

	ROBERTAGEN	DEDUCTREASONER
# Params.	139.71 M	142.40 M
# Attention heads	8	-
Hidden dim.	768	768
Feedforward dim.	1024	768
# Layers	2	-
Activation	ReLU	ReLU
Dropout	0.1	0.1
Label smoothing	0.05	-
# Constants	17	17

Table 4: Training hyperparameters.

	PRE-TRAINING	FINE-TUNING
Batch size	32	16
Max steps	10,000	50,000
Optimizer	AdamW [12]	
Weight decay	0.01	0.01
Max grad norm	0.1	1.0
Learning rate	3e-5	1e-5
LR scheduler	Linear	Linear