Analysis of Elementary Math Word Problems Based on AI Deep Learning

Mingzhe Li[1,*]

[1]Deerfield Academy, Franklin County, Massachusetts, United States of America
[*]Corresponding author

Abstract: *Natural language processing (NLP) has greatly advanced in machine learning, but math education software lacks AI integration for solving math word problems in English. We propose using the BertGen pre-trained Transformer model, along with the MAWPS dataset augmented by our dataset augmenter. The Transformer model, with its multi-head attention mechanisms, excels at capturing long-range dependencies and referential relationships, crucial for math word problems at the primary school level. Our accuracy tests and performance on different datasets validate the effectiveness and generalizability of our approach. Moreover, our augmented dataset outperforms smaller unaugmented datasets, while maintaining diversity. The math word problem augmenter can be adapted for other math problem sets, supporting future research in the field.*

Keywords: *Math word problem, Natural language processing, Transformer model*

1. Introduction

Computational intelligence has become an increasingly popular topic in the 21st century. Discussions on the topic span from computer science to philosophy. Can computers think? Can they learn? This question has troubled scientists and philosophers alike since the invention of the machine. Some philosophers have proposed in the famous Chinese Room thought experiment that computers within our current definition does not "understand" meaning, but merely take inputs and give outputs as they are programmed to do. Since they cannot understand, they cannot learn.

Due to this constriction, interpreting natural human language and responding accordingly has been something modern computers has struggled with. However, it is crucial to a computer's interactive experience and thus its accessibility to humans with no background in coding. This problem becomes especially serious in education, as most people in need of guided education in math or languages most likely haven't had access to computer programming.

In our project specifically, we want to focus on solving math word problems, referred to as MWPs, which are math problems expressed in natural language such as the following: "Mary has 10 apples initially but she gave away 5 apples to John. How many apples does she have now?" Computers have been able to easily solve unimaginably complicated integral expressions, but they often cannot fathom a simple question like the above about Mary giving 5 apples to John because of the aforementioned lack in human language understanding. Therefore, we settled for a more purposed and efficient AI learning model designed only to solve math problems.

With that goal in mind, we have constructed a learning AI with the goal to solve simple math word problems, or MWPs, by approaching the problem on two major fronts: generating a dataset of MWPs and training a deep learning AI model on this dataset. We started off with a reinforcement deep learning model, DQN Word Problem Solver but with some preliminary experimentation, we found that it wasn't very effectively, having a low accuracy and taking a long time to train. We hypothesize that since math problems are on a strict right or wrong basis, there is no middle ground. However, our optimizer model's reward system gave some reward for a close enough answer, which might have led the model to seek an easy and unambitious route which didn't bode well in its accuracy.

Therefore, we decided to try supervised deep learning models, specifically the Transformer, proposed in 2017 by Vaswani et al. and used in various models such as GPT-3 and BERT.

2. Related Work

Up till now, there has been a lot of attempts tackling the problem of solving MWPs with supervised deep learning. Researchers differ over the amount of supervision they use in their transformer models, ranging from those advocating very strong supervision or semantic parsing, by outlining the syntactical logic behind the problem and creating corresponding equation templates for the model to fill in [1] to those working with relatively weak supervision, only providing the final numeric answers [2] and improving the model through specific inner means such as looking for possible mistakes and fixing them [3]. Our training method takes the middle ground approach, providing the equations for learning but does not restrict the form of the equations with a template for sake of generalization.

The final component of our experiment is a data set with sufficient size for training a model but also with enough variety to ensure training quality. There are publicly available data sets in two languages, Chinese and English.[4] The Chinese problem sets, namely APE210K and Math23K are enormous containing 210488 and 23161 problems respectively while the English datasets all have under 3000 problems and thus is not enough for training a deep learning model[5-6] .We had decided that our model would be in English, so we generated our own problem set using a math problem generator that takes the MAWPS dataset, containing around 2373 unique problems, and replaces the words and numbers in it, preserving the variety but also enlarging the dataset so that it is sufficient for training.

We hope that by this endeavor we prove that it is possible for computers to understand mathematical language to a certain extent and hopefully, like Duolingo, implement MWP solving AI in real world applications.

3. Method

3.1. Creating a Usable Data Set

Since, as stated above, we didn't find any math word problem data sets in the English language that is both various and vast, we made our own data set with the goal to ensure quantity but at the same time preserve diversity.

To this end, we created a dataset augmenter that takes a random problem from an existing MWP data set, in our case MAWPS, and replaces certain words, mostly names, objects, and certain verbs in the problem text with another word of the same syntactical category with slight artificial restrictions. It also replaces all the key numbers with a new number with similar scale and decimal accuracy.

However, the implementation of this augmenter is not a simple one. We used python's nltk package to identify the part of speech of every word in a math word problem and replace certain word types (proper nouns) with other words of the same type. We also had lists storing common synonymous MWP verbs such as {give, grant, offer, etc.} and replace them accordingly.

3.1.1. Replacing Names, Objects and Certain Verbs

However, math word problems also have recurring names and objects that are crucial to the relationship expressed, and if we just replaced every word with a random one, we would completely lose the mathematical relationship we want to preserve. For every name or object our augmenter replaces, we store the (*replaced, replacing*) pair in an array that would be checked for every subsequent word in the problem. If one of the *replaced* words is found, we would then replace it with the *replacing* word stored in the array.

3.1.2. Replacing Numbers

For numbers in the problem, we underwent a similar process. MWPs often have meaningless number distractions that often confuses a computer, so our augmenter checks for recurrence of the number in the parsed solving equation. If it is present in the equation, which would mean that it is a crucial number, we then store the index of the two numbers in both the problem or the equation in another array. At the end of the iteration, our augmenter then replaces all the identical numbers with a newly generated number with the same scale and accuracy. This new number is the original number scaled by a random multiplier between 1/5 and 5, with logarithmic distribution, ensuring balance of probability from 5^{-1} to 5^{1}. The new number also keeps the decimal accuracy of the original number. This process ensures that every crucial recurring word or number in the problem is substituted with the same replacement that would also keep the structure and sense of the problem. We chose to replace the number only at the end instead of during the process like the word replacement because this logarithmic scale might generate for a crucial number *a* the same number to another crucial number *b* which is later on in the question; this would leave 2 *b*s in the solving equation. Then when we iterate to *b*, *b* would end up replacing both *b* and ruining the question. An example of our dataset would be the following in json format , as shown in Table 1:

Table 1: Sample Problem in My Dataset.

ID	chal-26658
Body	Joshua had 34 quarters in his bank. His dad offered him 9 more quarters
Question	How many quarters does he have now?
Equation	(34 + 9)
Answer	43
Type	Addition

3.1.3. Resulting Dataset

In the end, we acquired an enormous dataset of 50,000 math word problems. Figure 1 is a chart comparing the diversity and size of our dataset, mawps_augmented with the original mawps and asdiv-a datasets.

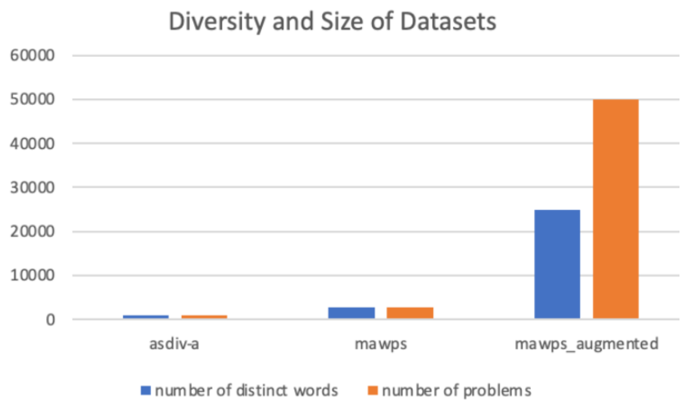


Figure 1: Diversity and Size of Datasets.

We believe that our augmentation would aid the efficient establishment of pattern recognition in our model without having the risk of the model overfitting to a small dataset. The size of the dataset would also make the training for each epoch more smooth and stable. In addition, this dataset augmenter, with some minor manual edits, can be used to effectively enlarge any math word problem data set (or combinations of them) which could significantly help the field in their model training.

3.2. Attempt with Deep Reinforcement Learning Model

We first looked at reinforcement learning models, experimenting with the DQN_Problem Solver [7] which applies deep reinforcement learning to an equation tree representation system. It first extracts all relevant features, in this case mathematical quantities from the word problem, then it creates all permutations of possible expression trees that represent the logical relationship described in the word problem with the operations (+, -, *, /, $-_{inv}$, $/_{inv}$) as learnable parameters, and finally ranks these trees based on a deep reinforcement learning algorithm to select the best possible expression tree representation. Figure 2 is the model's architecture. The final sentence of a caption must end with a period.

The reward system is based on the correctness of the operations: if the operation is correct, it returns positive reward; otherwise it returns negative reward.

However, with preliminary experimentation, we discovered that deep reinforcement learning seemingly reduces loss far too slow and to a lesser extent, retaining around a 0.2 loss after almost 30 epochs. We theorize that this might be because the feature extraction process at the beginning of the learning is too unreliable as the DQN model tries to create a list of present mathematical quantities with rule-based interventions, including individual feature (units after numbers), pair feature (similar syntax surrounding another quantity) and question feature (quantities with specific words such as "more" or "less" around it). It then applies deep reinforcement learning on that list of quantities. We think that in reality, feature extraction in math word problems might be the hardest thing for a computer to master, especially in higher-level math problems. And thus should warrant a learning algorithm rather than a simple rule-based approach. Thus, we turned to another deep learning model that we thought could better understand the word relationships in a sequence, the Transformer.

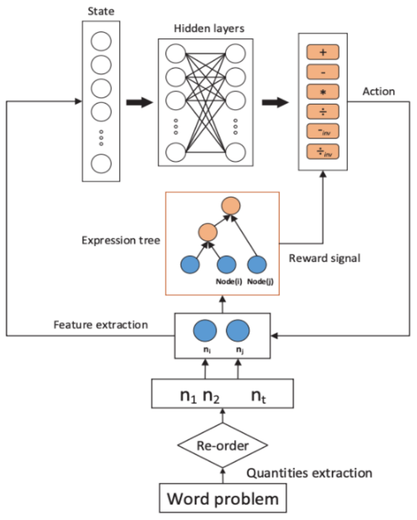


Figure 2: DQN model's architecture

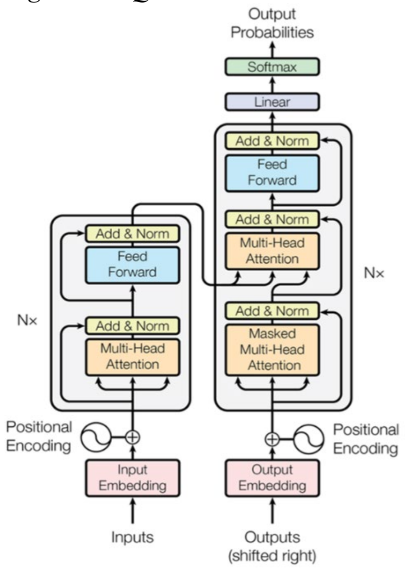


Figure 3: Transformer model's architecture

3.3. Transformer Model

We applied the Transformer Deep Learning model, proposed by Vaswani et al.[5], to create our problem solver. Transformers are unique in that in place of recurrent or convolutional neural networks they employ attention mechanisms which we thought would be better suited to learning the mathematical relationships induced by the wording of a math word problem. We first used a standard transformer model designed to receive a math word problem and output a solving equation, then we tried the pre-trained BertGen model committed to a task of equation output.

3.4. Traditional Transformer Model

3.4.1. Attention Mechanism

Our model uses self-attention which bases the output matrix on the attention weights assigned to those relationships based on the individual importance of tokenized.The specific attention function used in our model is Scaled Dot-product Attention with an input of a set of three learnable vectors: queries, keys and values. The query represents the token itself. The key is used to compute the similarity score between the query vector of the current token and the key vector of all the other tokens to arrive at an attention distribution. Finally this attention distribution is used to obtain a weighted sum of the value vectors for a final output for the token, which would suggest how much "attention" should be delegated to this specific token based on all the other tokens in the sequence. The Scaled Dot-product Attention function specifically uses a dot product of the queries and the keys scaled down by sqrt{d_k} before applying a SoftMax function and multiplying by the values, shown as follows in following formula[8]:

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_K}}\right)V$$

Our model also applies positional encodings in the input of the attention functions so that the relative order of the elements in the sequence can be taken into account of by the learning model.

We then apply multiple parallel instances of this attention function with different learned projected inputs of queries, keys and values, in every layer of its encoder and decoder, in a process known as Multi-head Attention. We concatenate the different outputs of these parallel functions, and project them again to achieve a single output. This process would allow the model to learn different dependencies between different parts of the sequence and develop more complex relationships, as well as make it generally more flexible. In our first model, we use 8 parallel attention functions.

Finally, our model has an encoder and decoder system designed for natural language processing. They both have 6 layers, in each of which Multi-head Attention is applied on the respective inputs to achieve an output of hidden states. The original sequence is filtered through this system of learning networks, shown below. This process transforms the input sequence into an output sequence of variable length, so it is ideal for our task, where we are transforming a math word problem into a simple equation. Figure 3 is the model architecture of our Transformer[9-10]:

3.4.2. Optimization and Learning

Our model uses the Adam (Adaptive Moment Estimation) optimizer for its learning and loss minimization. Adam is a combination of stochastic gradient descent and RMSprop. It utilizes the gradient of the loss function with respect to its parameters (SGD) to learn but it also adjusts its learning rate for each parameter according an average of the gradients and their squared values (RMSprop). This normalizes the learning rate and makes the algorithm less dependent on the initial choice of learning rate. In addition, it has a momentum mechanism, based on previous gradient values, which helps the function converge faster. Figure 4 is the specific learning function:

$$\text{lrate} = d_{model}^{-0.5} \cdot \min(\text{step num}^{-0.5}, \text{step num}\cdot\text{warmup steps}^{-1.5})$$

3.5. BertGen Model

In addition to the traditional transformer, we also experimented with the pre-trained BertGen model. BertGen is a transformer model, but it has several additional features that we thought would be desirable for our specific purpose of solving math word problems.

3.5.1. Pre-trained Perks

BertGen is a special archetype of the BERT Transformer and a pre-trained model, which means that it has already been trained on a large amount of unlabelled text and just needs to be fine-tuned to a specific task. This means that BertGen has already learned general linguistic patterns which would undoubtedly be helpful in comprehending math word problems. In addition, BertGen, as well as BERT, has a special pre-training objective known as MLM (Masked Language Modelling) where the input sequence is randomly masked and the model is asked to predict the original tokens. This has helped Bert to be especially effective in context interpretation which again would be helpful for our purposes.

3.5.2. Difference from Standard BERT Model

The difference of BertGen from the standard Bert Transformer model is that first it isn't trained in NSP (next sentence prediction) but only in MLM (masked language modelling), which makes it more proficient at capturing relationships between input tokens to generate a designated special output without having to worry about continuation of the input. This makes BertGen more suited to the task of solving math word problems than BERT, as the equation and solutions we are trying to generate are not continuations of the input but rather responses to it.

4. Experimentation and Results

4.1. Dataset Pre-processing

Our data set, as mentioned before, is an augmented version of MAWPS. However, eve before we augmented it, we had to manually adjust and universalize the format of the solving equation and then convert it into a json file. Ultimately, we created a vast data set of 50,000 problems that we used for training and testing.

For the training of both the traditional transformer and BertGen, we allocated 47,500 problems for the training dataset while the other 2,500 we isolated for testing. We lowercased, spacified and tokenized the problems and equations and recorded all of their vocabulary in two dictionaries for the model to assign parameters to.

4.2. Traditional Transformer Training

4.2.1. Training Curve

The traditional transformer learned the math word problems fairly quickly with our dataset, climbing to a 90+ accuracy in training after only 5 epochs. The full training accuracy curve in % of test set problems solved VS epoch number is shown below in Figure 4:

As can be seen, the Transformer attains 90% accuracy quickly in 5 epochs and then enters a period of slow steady improvement. And with 13 epochs, our model was able to achieve a Corpus_Bleu score of 0.97511 and an accuracy of 91.9% on our testing dataset.

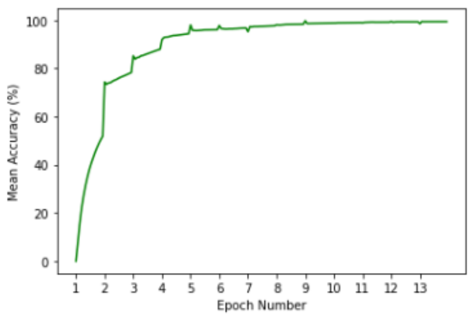


Figure 4: The test set problems solved VS epoch number.

4.3. BertGen Training

4.3.1. Comparison of Different Training Dataset for BERT

The BertGen model performed similarly well and even better than the Transformer model. We also used it to compare our dataset and the existing dataset ASDIV-a to see which one is more efficient in training. We had thought that our dataset would make the traiing process more smooth and stable. This is confirmed in our training, with Figure 5 showing the accuracy across epoch curves of our BertGen model using the asdiv-a dataset compared to if it uses our augmented dataset.

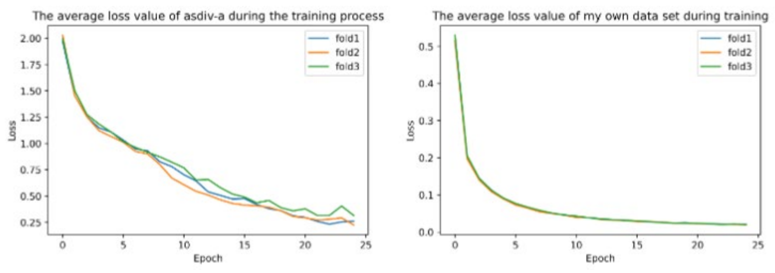


Figure 5 The accuracy across epoch curves of our BertGen model using the different datasets.

For the final results, we can also observe a drastic edge for our dataset in both equation and value accuracy, shown in Figure 6:

It is evident that our dataset performed better in all regards than the asdiv-a dataset by 25 epochs, achieving a 97.1% accuracy in its testing set.

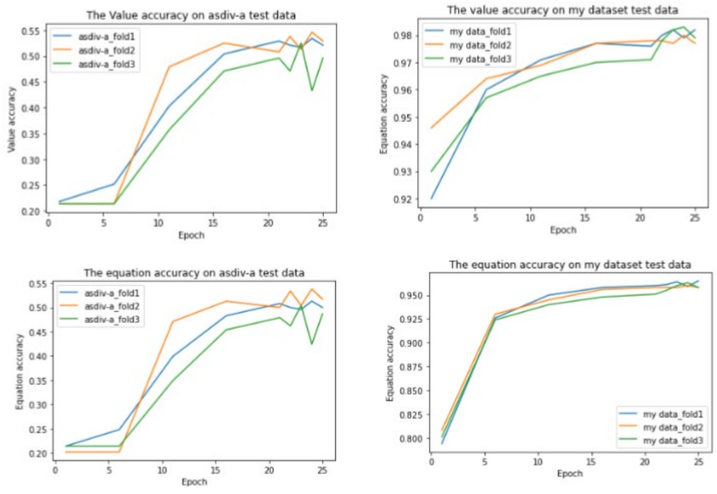


Figure 6: Our dataset in both equation and value accuracy.

4.4. Generalizability Check with ASDiv Dataset

Although both of our models performed really well. We also wanted to see how generalizable our model can be, so we took 960 problems covering the four operations (addition, subtraction, multiplication and common-division) from another data set, ASDiv that was not involved in our training in any way and checked our model's accuracy with that dataset. Undoubtedly, asdiv-a had drastically different forms for its problems, so we knew that our accuracy would never be as high as for our own testing dataset.

4.4.1. Accuracy Results with Both Models

Indeed, our transformer model only correctly solved 45.73% of the total 960 problems while our BertGen model solved 61.7% of the problems. This is likely due to the fact that BertGen is a pre-trained model and so already had linguistic basis that made it easier for it to adapt to an entirely

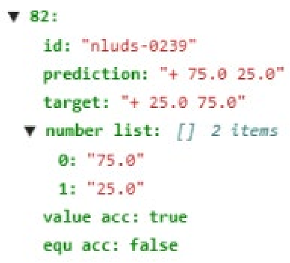
4.4.2. Speculation for Cause of Lower Accuracy in Generalization in the Bert Model

Going through the incorrect problems, we noticed some common mistakes. The value accuracy for Bert is 0.617 while the equation accuracy is 0.57. This is mostly because for some problems, mostly addition and multiplication Bert correctly arrived at the answer but differed from the standard equation in its form, like switching the order of the numbers which would yield the same answer in addition or multiplication, but not so in subtraction and division. An addition multiplication order switch like this , as shown in Table 2:

Table 2: Sample Problem in My Dataset.

ID	nluds-0239
Grade	3
Source	http://www.k5learning.com
Body	Rangers from Flora Natural Park and Wildlife Reserve also joined the activities on that day. They planted 75 redwood trees and 25 cypress trees to replace the trees that were destroyed during a recent forest fire.
Question	How many trees did the rangers plant?
Solution-Type	Addition
Answer	100 (trees)
Formula	25+75=100

Bert's Solution VS Target Solution:



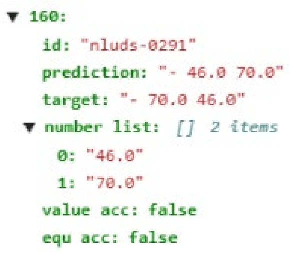
```
▼ 82:
   id: "nluds-0239"
   prediction: "+ 75.0 25.0"
   target: "+ 25.0 75.0"
 ▼ number list: [] 2 items
     0: "75.0"
     1: "25.0"
   value acc: true
   equ acc: false
```

In the case of subtraction, this would yield a value inaccuracy as well as an equation inaccuracy , as shown in Table 3:

Table 3: Sample Problem in My Dataset.

ID	nluds-0291
Grade	3
Source	http://www.k5learning.com
Body	Alicia loves collecting art. She has a whole house filled with all the art she obtained since she was a little kid. When she decided to move away, she started donating some of her art to different museums. First she had 70 art pieces, but she gave 46 away.
Question	How many medieval art pieces were left with hers?
Solution-Type	Subtraction
Answer	24 (medieval art pieces)
Formula	70-46=24

Bert's Solution VS Target Solution:



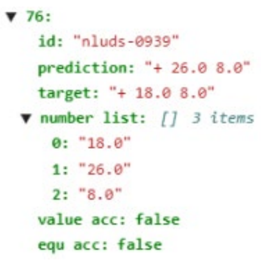
```
▼ 160:
   id: "nluds-0291"
   prediction: "- 46.0 70.0"
   target: "- 70.0 46.0"
 ▼ number list: [] 2 items
     0: "46.0"
     1: "70.0"
   value acc: false
   equ acc: false
```

Apart from order mistakes, Bert seemed to have some trouble dealing with irrelevant quantities, sometimes mistaking them for elements in the equation. For example in the following problem , as shown in Table 4:

Table 4: Sample Problem in My Dataset.

ID	nluds-0939
Grade	3
Source	http://www.mathplayground.com
Body	Mrs. Randall has taught third grade for 18 years. She has 26 students this year. She also taught second grade for 8 years.
Question	How many years has Mrs. Randall been teaching?
Solution-Type	Addition
Answer	26 (years)
Formula	18+8=26

Bert's Solution VS Target Solution:



```
▼ 76:
   id: "nluds-0939"
   prediction: "+ 26.0 8.0"
   target: "+ 18.0 8.0"
 ▼ number list: [] 3 items
     0: "18.0"
     1: "26.0"
     2: "8.0"
   value acc: false
   equ acc: false
```

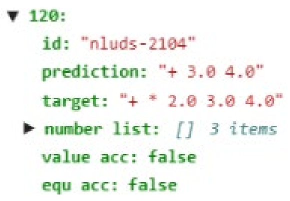
Bert placed more importance on 26 years instead of 8 students and thus resulted in a mistake.

Finally, Bert had considerable trouble with multiple quantities, most of the time selecting two and discarding the rest, , as shown in Table 5:

Table 5: Sample Problem in My Dataset.

ID	nluds-2104
Grade	3
Source	http://www.commoncoresheets.com
Body	A baker made 2 batches of chocolat chip cookies. Each batch had 3 cookies in it. Then he made an additional 4 oatmeal cookies just in case someone didn't want chocolate chip.
Question	How many cookies did he bake total?
Solution-Type	Multiplication then Addition
Answer	10 (cookies)
Formula	2*3+4=10

Bert's Solution VS Target Solution:



```
▼ 120:
   id: "nluds-2104"
   prediction: "+ 3.0 4.0"
   target: "+ * 2.0 3.0 4.0"
  ▶ number list: [] 3 items
   value acc: false
   equ acc: false
```

As can be seen, Bert ignored the 2 and only considered 3 and 4.

4.4.3. Possible Solution

All of these problems can be easily solved on the surface if we added a similarly augmented asdiv-a dataset onto our dataset as Bert would be trained on these problems. That would definitely be an improved model, but we wouldn't be able to compare its generalizability in the same fashion, so we decided against doing that since we believe that we have already made it evident that Bert can learn most math word problem datasets it is given. The last problem of Bert not dealing well with multiple quantities can certainly be solved with a more thorough dataset that contains more problems on multiple quantity MWPs. On the other hand, Bert did outperform the traditional transformer model by 34.9\% in this generalizability test, so that has shown that pre-training did help.

5. Conclusion

In this paper, we first introduced the field of applied AI learning in accessible education, then proposed the traditional transformer model and the BertGen model for solving math word problems while detailing the dataset augmentation we did and finally the results we achieved, including a generalizability test on a completely separate dataset. We have shown that both the traditional transformer and BertGen achieved a high accuracy on its testing set, 91.7% and 97.1% respectively. We have also shown that they still achieved a 45.7% and 61.7% accuracy on the separate asdiv-a dataset after being trained by ours, proving that these models are generalizable, albeit having a lower accuracy with completely unfamiliar problems.

For the future, apart from trying to improve generalizability, we are excited for the real-world applications of our program as a tool for education, for mathematical learning for primary schoolers around the world. This program and future improvements of it might reduce the cost of education around the world. From a technical standpoint, we have proven that deep learning can effectively help the computer “learn” how to do simple problems, proposed a new dataset augmenter to help future researchers create varied and large datasets while opening the door to problem solvers for harder problems in the future.

References

[1] Kushman, N., Artzi, Y., Zettlemoyer, L., Barzilay, R. (2014, June). Learning to automatically solve algebra word problems. In Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers) (pp. 271-281).
[2] Chatterjee, O., Waikar, A., Kumar, V., Ramakrishnan, G., Arya, K. (2021). A weakly supervised model for solving math word problems. arXiv preprint arXiv:2104.06722.

[3] Cheng Y , Li B .Computer data analysis and processing based on character recognition and deep learning technology[C]//ICASIT 2020: 2020 International Conference on Aviation Safety and Information Technology.2020.DOI:10.1145/3434581.3434727.
[4] Shi, S., Wang, Y., Lin, C. Y., Liu, X., Rui, Y. (2015, September). Automatically solving number word problems by semantic parsing and reasoning. In Proceedings of the 2015 conference on empirical methods in natural language processing (pp. 1132-1142).
[5] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Polosukhin, I. (2017). Attention is all you need. Advances in neural information processing systems, 30.
[6] Zhao, W., Shang, M., Liu, Y., Wang, L., Liu, J. (2020). Ape210k: A large-scale and template-rich dataset of math word problems. arXiv preprint arXiv: 2009.11506.
[7] Wang, Y., Liu, X., Shi, S. (2017, September). Deep neural solver for math word problems. In Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing (pp. 845-854).
[8] Koncel-Kedziorski, R., Roy, S., Amini, A., Kushman, N., Hajishirzi, H. (2016, June). MAWPS: A math word problem repository. In Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (pp. 1152-1157).
[9] Wang, L., Zhang, D., Gao, L., Song, J., Guo, L., Shen, H. T. (2018). MathDQN: Solving Arithmetic Word Problems via Deep Reinforcement Learning. Proceedings of the AAAI Conference on Artificial Intelligence, 32(1). https://doi.org/10.1609/aaai.v32i1.11981
[10] Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.