# Math Multiple Choice Question Solving and Distractor Generation with Attentional GRU Networks

Neisarg Dave*
Pennsylvania State University
nud83@psu.edu

Riley Bakes*
Pennsylvania State University
rob5372@psu.edu

Barton Pursel
Pennsylvania State University
bkp10@psu.edu

C. Lee Giles
Pennsylvania State University
clg20@psu.edu

## ABSTRACT

We investigate encoder-decoder GRU networks with attention mechanism for solving a diverse array of elementary math problems with mathematical symbolic structures. We quantitatively measure performances of recurrent models on a given question type using a test set of unseen problems with a binary scoring and partial credit system. From our findings, we propose the use of encoder-decoder recurrent neural networks for the generation of mathematical multiple-choice question distractors. We introduce a computationally inexpensive decoding schema called character offsetting, which qualitatively and quantitatively shows promise for doing so for several question types. Character offsetting involves freezing the hidden state and top k probabilities of a decoder's initial probability outputs given the input of an encoder, then performing k basic greedy decodings given each of the frozen outputs as the initialization for decoded sequence.

## Keywords

Math Question Solving, Distractor Generation, Math Multiple Choice Questions, Mathematical Language, Math Education

## 1. INTRODUCTION

### 1.1 Problem Statement

Here we focus on the needs of mathematics educators in high school and early university education, One of the most tedious jobs for a teacher is to create exams and quizzes and grade them. The more time they spend on these tasks, the less time they spend teaching students. An automated system capable of creating reliable math questions of consistent difficulty level, creating solutions, generating distractors for them, and finally be able to grade them is the holy grail of educational automation. In this paper we focus on solving the math questions and generating distractors for Multiple Choice Questions.

*Both authors contributed equally.

Neisarg Dave, Riley Owen Bakes, Bart Pursel and C. Lee Giles "Math Multiple Choice Question Solving and Distractor Generation with Attentional GRU Networks". 2021. In: Proceedings of The 14th International Conference on Educational Data Mining (EDM21). International Educational Data Mining Society, 422-430. https://educationaldatamining.org/edm2021/
*EDM '21* June 29 - July 02 2021, Paris, France

Questions in mathematics are different from other subjects such as English, History, or Economics. In mathematics and by extension in all STEM fields, questions and answers not only are composed of natural text but are often accompanied by symbolic equations, expressions, inequalities or relational information. Ganeslingam [6] postulates in his work that these non-textual objects not only augment the context of the textual part but also derive their context from it. These non-textual objects are not part of the natural language and hence require special treatment. On the semantic level, mathematical questions require an underlying understanding of rules before a question can be solved. A mere comprehension is not sufficient. To solve a simple problem in arithmetic, the fundamental understanding of the four operators is necessary.

In this work we experiment with a network which has had historical success on natural language processing problems and test its ability to generalize mathematical knowledge from an open source data set contributed by [22] consisting of elementary focused question types. Alternatively as a second problem, for some question types, we examine whether models which fail to generalize to the test set may have their incorrect solutions leveraged as 'good' distractor options for multiple choice questions like those seen in multiple choice questions on a math quiz. As following with the precedent of the data set contributor [22], mathematical expressions are presented using Python's operator syntax.

In summary we show the following:

- Insight in the ability of an encoder decoder attentional GRU to extract semantic and syntactic meaning from mathematical expressions.
- Simultaneously test whether these model's incorrect predictions may be leveraged to auto generate multiple choice question distractors commonly seen in lower education exams. Continuing with this potential application, experiment with the practice of character offsetting–a modified greedy decoding schema which pushes the networks to predict four separate sequences instead of a single output thus providing a complete set of distractors.

- Qualitatively measure which question types show the highest potential for leveraging character offsetting for the purpose of multiple choice distractor generation.

## 2. RELATED WORK

**Math Word Problems**

Early applications of machine learning and deep learning based methods in math questions attempted to convert math word problems into equations [10]. Here we rely on the capability of neural network to identify the textual and equation parts from the question and model them correctly in order to solve them. Much of the work in math word problems relied on extracting equations from text and then solving them using symbolic solver libraries such as Sympy.

Subhro Roy and Dan Roth created expansion trees [20] and Unit Dependency Graphs [21] from arithmetic word problems. Kushman et. al. [11] mapped word problems to equations using canonical templates and handled ambiguity using probabilistic models. Tencent AI Lab [29] first used deep neural networks for solving math word problems. They used the Seq2Seq model with LSTM units for mapping math word problems to the equations. MathDQN [27] proposed using Deep Q-Learning to map math word problems to solvable equations. Deepmind [15] attempted to solve math word problems using the program induction technique, which would also generate a rationale for choosing an answer. This method did not involve mapping problems to equations but the reasoning in text form. Recent work focused on using recursive neural networks for evaluating equations [30] [28] mapped from word problems.

Polozov et. al. [19] and Liu et. al [16] proposed methods for generating math word problems. Liu et. al. [16] used Gated Graph Neural Networks with Variational Autoencoders to generate questions from knowledge graphs of mathematical concepts and symbolic expressions. [19] take programming approach for encoding requirements from tutors and students to create logical graphs with the help of ontology. This logical graph is then used to create expressions and sentences with the help of primitive templates.

Lample et. al. [12] showed solutions for questions in differential equations, differential calculus, and integral calculus and used transformer networks [26] to solve calculus questions and compared their results with traditional solvers like Mathematica and Matlab. In contrast, Saxton et. al. [22] created a codebase to generate math problems across fifty-six classes and solve them using deep neural networks. Saxton et. al. compared problem solving abilities of Seq2seq networks with transformers. Though transformers showed better results than the the recurrent models, but Saxton et. al. commented that the improvement in performance was largely due to the higher capacity of transformer networks to remember rather than their ability to solve.

Here we use the codebase created by Saxton et. al. [22] to generate math questions. Even though the codebase in its original form cannot generate distractors, it can be modified to create distractors using simple rules. In comparison to other datasets [15] which contain distractors, we chose to use the codebase since it provided more control over the generation of questions and also the templates used have a simpler language and an equation with each question.

In a classroom and tutoring settings math questions are more open ended. Erikson et. al [5] tested the capability of XGBoost, Random Forests and LSTMs in analyzing the open ended answers in mathematics. These models were created to assist the teacher rather than complete automated grading. Michalenko et. al [17] used LSTMs to solve polynomial factorization problems. They created their dataset from Wolfram Alpha. They use the trained network for automated grading and personalized feedback system.

**Distractor Generation (DG)**

In multiple choice questions, the options which are not the answer are called distractors, because their job is to distract a student from a given correct answer. Distractor generation has been studied for non-mathematical subjects, especially English (Susanti et. al. [23]) and other domain-specific tasks (Aldabe et. al. [2]). Distractor generation for scientific subjects like physics, chemistry, biology, and economics was explored by Linag et. al. [13]. They [13] used a two-stage model with a classifier and a ranker to filter out the relevant distractors. Linag et. al. [14] explored distractors for fill in the blank type questions using GAN networks.

**Partial Credit Scoring**

Similar in spirit to our partial credit scoring system, Pho et. al. [18] attempt to automatically score the *quality* of manually created English multiple choice distractors using various semantic and syntactic criteria including WordNet. This is fundamentally different from our problem however as we seek to automate the generation of the distractors themselves and simultaneously sought out a metric to help measure the fundamental reasonableness of those distractors.

## 3. EXPERIMENTS

### 3.1 Training Data

The data set used in this paper [22] had the express purpose of being a large scale training and testing framework for benchmarking models on mathematical reasoning. The framework consists of both training and testing sets. The training set consists of 39 different math problem types and variants of 17 of those add the element of mathematical composition to the problem's statements for a total of 56 question types organized into 8 different domains–probability, polynomials, numbers, measurement, comparison, calculus, arithmetic, and algebra. Each question type within a domain is split into three training sets, easy, medium and hard of 666,666 question answer pairs, for a total of 2 million examples per question type.

Difficulty measures the relative complexity of coefficients in the expressions generated. As an example compare from the polynomial evaluation set the easy: 'Let u(q) = q**2 - 6*q - 10. Calculate u(8).', medium: 'Let s(v) = v**3 + 47*v**2 +471*v + 142. Give s(-33)', hard: 'Let h(a) = -177071*a - 4957992. What is h(-28)?' and an actual related college algebra exam question [25]: 'Evaluate the function f(x) = 3 + (x-5)**(1/2) at x = 9.'. It is relatively clear that examples from medium and hard are unlikely to appear on a low level math examination. For this reason the majority of experiments rely on the easy train set variants. It was our hypothesis that as the curriculum provided by these sets are much more in line with the expected complexity of questions appearing on actual low level exams that the models would thus be more likely to generate better distractors (or even the correct answer) when provided such an exam's questions as input.

The primary test method within the framework proposed by [22] is a data set referred to as interpolation. Every question type has an associated interpolation test set. The set consists of $10^5$ question answer pairs *likely* unseen in either the easy, medium and hard train sets of the associated question. The guarantee of lack of repeated questions comes from a lower bound on the probability of a questions repeated generation. A training question at most has a $10^{-8}$ chance of reappearing in the test set. [22] also release a secondary test set referred to as extrapolation, which measures generalization of core concepts across multiple question types. However, as we specifically were interested in single question trained models for the express purpose of multiple choice question distractor generation this test set was unused.

## 3.2 Rule Based Distractor Generation

Evaluation of distractors is not an exact process. For a given question there can be any number of distractors, some good, other bad. There is usually a very loose concensus on what constitutes a good distractor. Also no algorithm exists that can gauge the "goodness" of distractor. However, expert educators have a keen sense of judging the distractor by their teaching and research experience. Educators usually know where students make mistakes, and leverage that to generate good distractors. For simple high school questions, we can simulate this process by creating rules that mimic the mistakes made by students. These rules then can be embedded with a mathematical solver to produce distractors for given question. A simple set of rules can be created to modify the solution steps to generate distractors for questions containing mathematical equality or inequality. Commonly used rules are:

- *Change One Sign* : Randomly pick one coefficient or constant in equality/ inequality and multiply by $-1$.
- *Change Two Signs* : Randomly pick two coefficients or constants of equality/inequality and multiply by $-1$.
- *Most Frequent Number* : Use the most frequent number in the equality/inequality as a distractor
- *Nearest Multiple* : Randomly pick a coefficient or a constant in equality/inequality and change it to the nearest multiple of 2, 3 or 5.
- *Random Drop* : Randomly drop one of the coefficients or constant in equality/inequality
- *Invert Range* : Invert the solution range of the inequality, e.g. change $[0, 1]$ to $(-\infty, 0) \cup (1, \infty)$
- *Trivial Solution* : For inequality problems, one of the distractors can be chosen from $\{\phi\}$, $(-\infty, \infty)$, or *No Solution.* For equations, choices are from 0, $-1$ or 1
- *Flip brackets* : Change an open bracket in answer to closed and vice and versa. In the question, order of operations can be changed by changing the position of brackets.

These rules can be coded as python functions and then selected one or two rules at random to modify the steps involved in solving the question. Symbolic library like sympy can be used for generating and solving the math questions. The library developed by deepmind [22] can create math questions across various domains with varying difficulty level. We modify their codebase to extend its capability to generate the distractors based on the stated rules. Table 1 shows few examples of rule based distractor generation.

| Question | Answer | Distractors |
|---|---|---|
| Let $-\frac{2p}{3} - \frac{2}{3} \geq \frac{2p}{5} - \frac{4}{5}$. What is $p$? | $-\infty < p \leq \frac{1}{8}$ | $\frac{11}{2} \leq p < \infty$<br>$3 \leq p < \infty$<br>$\frac{4}{23} \leq p < \infty$ |
| Find all solutions to $\frac{3}{2} - \frac{w}{6} \geq -\frac{2w}{11} - \frac{14}{11}$. | $-183 \leq w < \infty$ | $-\frac{61}{4} \leq w < \infty$<br>$-\infty < w \leq \frac{61}{4}$<br>$-\infty < w \leq \frac{47}{2}$ |
| Solve the polynomial inequality: $51 - 3f \neq -f - \frac{1}{6}$ | $f \neq \frac{307}{12}$ | $f \neq -\frac{305}{24}$<br>$f \neq -\frac{305}{18}$<br>$f \neq -\frac{46}{3}$ |

Table 1: Distractors generated using rules

Distractors generated using rules act as a form of reference distractors. For qualitative evaluation of distractors generated by neural networks, we will look at both the distractors side by side in table 3.

## 3.3 Experiment Detail

Two principle experiments can be identified. An attentional [3] encoder decoder GRU [4] is trained on a single question type for the entire 666,666 easy train set. Keeping with the spirit of the framework released by [22] we after training a model on a specific question set test the models on their respective question's interpolate test rather than a subset of the train set.

Simultaneously, during the second round of data collection with the GRU, when the model is scored on the interpolate set we perform character offsetting (see 3.3.1) and ask the model to predict 3 distractors in addition to a primary solution sequence. Two different scores were calculated for a model's performance on the test set–the first, a complete binary accuracy where credit is assigned if and only if the entire primary greedy decoded sequence matches the true solution sequence. And second, a partial credit score which is calculated by subtracting from 1 the normalized Levenshtein edit distance between the predicted and true solution. Normalized in this context means the ratio of the edit distance to the max sequence length of either the true or predicted solution sequence. Thus for a given Levenshtein distance $d$ for solution sequence $S$ and prediction sequence $P$ we have partial credit defined as

$$\text{partial credit} = 1 - \frac{d(P, S)}{\max(\text{len}(P, S))} \quad (1)$$

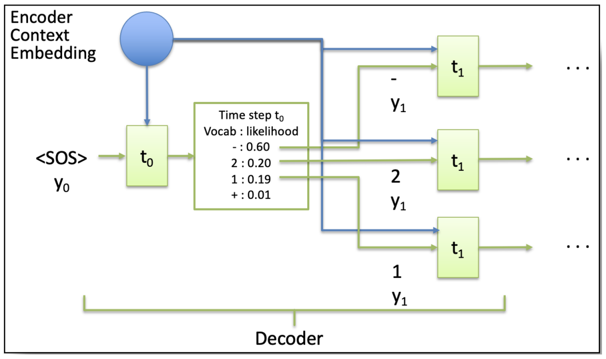

Figure 1: Example of character offsetting given an encoder's context embedding from the input question '-2 + 1'. Note decoder hidden states not shown in diagram. <SOS> signifies the start of sequence character.

It is important to acknowledge the widely different possible interpretations for what partial credit could mean in this context and why we choose the definition we did. In a typical educational setting, partial credit is assigned when the student demonstrates sufficient understanding of the problem albeit fail to arrive at the final correct solution. This is difficult to measure in model outputs–consider a correct sequence being $-2$ and a predicted sequence of 2, the partial credit score would be $1 - \frac{1}{2} = 50\%$. In real life a teacher may realize a student failed to report a final sign change and assign significant credit. Such thorough review is impossible given the $100,000$ examples within a single question type's test set (and also impossible considering the model's inability to provide secondary and tertiary decision making steps)–and so we formulate the above definition as an attempt to empirically measure the ability for a model to predict *similar* expressions to the correct one. We emphasize this is not any attempt to measure the models comprehension of the mathematics it is predicting on. As for why this is important; the ultimate goal is to generate multiple choice question distractors which generally exhibit some form of similarity to the correct solution. We discuss defining a good distractor more fully in section 3.3.2.

### 3.3.1 Character Offsetting

We propose a modified greedy decoding schema called character offsetting for generating multiple choice question distractors. In a typical greedy decoding scheme for an encoder decoder sequence to sequence model an input sequence (in our case, a string literal representation of a math problem) is given to the encoder which generates a context embedding [24]. Then character by character the decoder outputs a response sequence based on this context. At every time step of the output sequence's prediction, the previously predicted character and hidden states, as well as the encoder's context output, are re-fed into the decoder. The actual output of the decoder at any step is a probability array the size of the model's vocabulary [8]. The highest value corresponds to the most likely next character in the sequence, at least according to the model's weightings. In a greedy decoding at every step we simply take the most probable character and append it to the final output. Prediction is halted once the model outputs the end of sequence character as the most probable next step [8].

In character offsetting we freeze the initial decoder returned probability array and hidden states. We now ask the decoder to generate four total prediction sequences–one being your expected greedily decoded output, a second with the second most likely character from the frozen initial probability array as the sequence's starting character, and similarly a third and fourth. Each time a new sequence is attempted, we reset the hidden state to the saved initial hidden tensor. This was found for several question types to generate diverse and reasonable incorrect outputs. Table 4 provides a qualitative ranking based on question type for this task.

### 3.3.2 Difficulty in Defining a Good Distractor

Defining a good distractor is a non-trivial endeavor, and we make no claim to have accomplished this in this paper. Rather we discuss *qualities* typically considered when trying to formulate distractors for a multiple choice assessment.

Some qualities are readily apparent–general reasonability of a distractor as a possible solution to the question posed is perhaps the most fundamental requirement [7]. Measuring reasonability may be accomplished in several ways. Difference in value between a distractor and the true solution are potentially a good baseline–a distractor should be within a context specific similar magnitude as the true solution to avoid immediate exclusion. For lower level maths such as the the problem types discussed we believe this to be typically within a magnitude difference.

## 3.4 Model Parameters and Training Procedure

The model experimented with was an encoder decoder attentional GRU trained on a single NVIDIA 1080TI GPU for a single train-easy curriculum question type from the [22] data set. The models encoder and decoders had an embedding layer of size 512, with the decoder having 16 attentional heads. Initially what was attempted for a given training question type was an encoder decoder hidden size of 2048 on a batch size of 256. If the 1080TI GPU memory was insufficient given a training question type then we alternated between dropping encoder size and batch size. The parameters for a given question type are recorded in table 4. 150 training epochs were performed.

We follow most of the parameters used in [22]–the Adam optimizer [9] was selected for minimizing the sum of the log probabilities of the correct character with learning rate $lr = 6 * 10^{-4}$, and $\beta_1 = 0.9$, $\beta_2 = 0.995$, and $\epsilon = 10^{-9}$ and absolute gradient clipping of 0.1. The model leveraged teacher forcing during training and used 0.9/0.1 split of training data into a train and validation set.

## 4. RESULTS AND ANALYSIS

## 4.1 Attentional GRU on the Interpolate Set

### 4.1.1 Performance Considerations

It should be noted that these models have removed from them the greater context provided by the train-medium and train-hard data sets, of which interpolate attempts to test understanding for as well. It is possible as well that the hard or medium sets better generalize to interpolate for specific question sets. A small test seems to support this–we let the model train on the hard variant of algebra_linear_1d and scores improved from 3.9% to 44.3%.

| Metric | Mean Score |
|---|---|
| Binary | 0.065 |
| Partial Credit | 0.679 |

Table 2: Comparison of attentional GRU partial credit and binary scoring performance averaged across all questions not disqualified in 4.2.1. This includes low potential questions excluded in table 4.

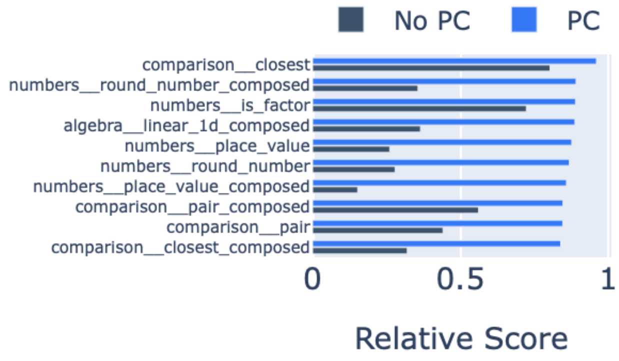

Figure 2: Top ten scoring questions when evaluated with the partial credit (PC) metric and no partial credit (binary) (No PC) score.

### 4.1.2 *Analysis: GRU Performance Binary Scoring*

Performance when scored with no partial credit varied widely. Of the top ten scoring question types 5 are from the comparison module type. Based on overall poor performance we are skeptical that the models extrapolate true mathematical semantic meaning–rather they likely just determine meta solution strategies. For example, one hypothesis for the comparison task success is that the notion of magnitude is readily apparent based on the length of the input sequence. Two observations about this–the first is that this requires the model to have gained the ability to isolate critical numeric subsequences within a larger question prompt. Secondly, providing contrary evidence to this hypothesis of sequence length metagaming, is that even for examples comparing long decimal sequences of lesser magnitude to short whole integers of greater magnitude we observe successful predictions. Lastly, the implication of signage in comparison was understood well by the models.

Generally it would appear that magnitude is an easier concept for statistical pattern recognition to abstract. Most difficult for the network was the evaluation of polynomials and other algebraic expressions, and arithmetic. Binary scores in all these categories were low–even given the considerations provided in section 4.1.1.

### 4.1.3 *GRU Performance Partial Credit Scoring*

Measurement with the partial credit metric (table 2, figure 2) demonstrates greater consistency and performance and shows promise for the ability of the attentional GRU to capture the essence of a reasonable response and work as a distractor generator for multiple choice questions–especially for types of problems the model performed poorly on using a binary scoring metric. However we note some flaws in the measurement. An example: in algebra_linear_2d_composed the model would frequently predict a single negative sign, a safe prediction and given the length of correct outputs is typically only one or two characters this led to a significant boost in score while answers remained meaningless. Interestingly in the question set's non-composed variant algebra_linear_2d the model's outputs are mostly meaningful and the partial credit score seemingly justified. To supplement the partial credit score, a qualitative examination of the reasonability of model's outputs compared to their empirical partial credit scoring is provided in table 4.

## 4.2 Multiple Choice Question DG

### 4.2.1 *Considerations*

The [22] framework releases a wide range of question types posed in diverse formats. Not discussed until now is that several formats are not conducive towards training models. Take for example questions from the comparison_closest set which are themselves posed as multiple choice questions–

'Which is the nearest to -955? (a) -3/4 (b) 0.2 (c) 17/3 (d) 3/5 (e) 0.5'

A model is supposed to predict either a, b, c, d, or e. Of the data sets released only four were found to use such a format for some or all questions within the set. A similar problem; six sets were either partially or fully posed as simple true and false questions. Naturally such questions are removed from consideration from our goal of distractor generation.

Partial credit was found to be an effective indication for many question types of whether a model's principle predicted sequence captures what a reasonable response should look like. Some faults exist however–consider the question type comparison_sort. An example: 'Sort -1, 0.3, -6, -24/11, 3, 5, 1 in descending order.' with primary prediction '5, 3, 1, -0.3, -1, -24/11, -6'. Observe the $-0.3$ in the prediction output–a value which is not even an option given in the problem statement! Such an example would not make a worthwhile distractor as it fails to test for the mathematical notion this question fails to over a high partial credit score. In table 4 we provide a qualitative review per question type of character offset predictions for multiple choice question distractor generation. By comparing to their respective partial credit score we find that generally a high score is an indicator for character offset predictions to also be reasonable.

### 4.2.2 *Character Offset DG: Interpolate Sampling*

The following are a couple of curated model responses–the order of the distractors matches the ordering of the probability of the initial character offset. So the first value listed is the model's primary greedily decoded prediction, the second is the sequence generated when we force the initial character to be the second most probable, and so on. If the model predicts the correct solution it is bolded.

'solve -2*s - 40 = -2*j, 53*j - 62*j + 245 = 4*s for s.'
output: **5**, -5, 4, 1

'let i(a) = -a**2 + 1319*a - 22130. calculate i(17).'
output: **4**, -4, 5, 14

Here we see a pair of ideal outputs from the algebra_linear_2d and polynomials_evaluate sets respectively. Not only did the model successfully predict the correct solution sequences of 5 and 4, but also produced noteworthy distractors. Observe the similar magnitude and the prediction of -5 and -4–sign changes of the true solutions and a powerful arithmetic skills check.

Quantitative measurement of the potential for character offsetting in generating multiple choice question distractors is a difficult endeavor due to the lack of formal definition of the problem. However, we observe general groupings of questions which seem to have potential for the use of character offsetting as a computationally cheap method of doing so. Table 4 is a qualitative ranking when given a random subset of outputs for each interpolate question type, whether offset predictions are not reasonable (low, not reported), whether at least one offset sequence is reasonable (medium), or whether most offset sequence predictions are reasonable (high). We view a prediction as unreasonable if it is mathematically meaningless or as mentioned in 4.2.1 either unreasonable given the problem's context or disqualified due to format. We find a higher partial credit score of the primary predictions frequently but not always aligns with whether offset predictions would also be reasonable.

Table 3 shows the comparison of the model generated distractors and rule based distractors for five questions. We generated four distractors for each question from the model and generated ten distractors from the rule based system. In the given table we show the most matching distractors. As we can see we get two matches each for questions (1), (4) and (5). There are three matching distractors for question (3) while there is no matching distractor for question (2). Despite being no matching distractor, our model gets the form of the distractors correct, and the predicted distractors at a glance can be used on a real quiz. In question (4), we can see that the model gets the multiplicity of 120 right and tries to stay around it. From the above examples we can see that our model first tries to get the form of the answer correct and then aims for computational compositionality.

### 4.2.3 *Character Offset DG: Standard Exam Sampling*

We sample several actual standardized exam questions from the SAT, ACT, and a college algebra midterm. Questions are altered before being fed to a model so that mathematical syntax matches Python's. Interestingly models appear resilient to significant changes in the question's formulation. Correct exam solutions are bolded, and generated options are in order of the probability of the initial character offsetting. The exam distractors are provided for comparison below but are removed before the question is fed to a model.

'What is the greatest common factor of 42, 126, and 210? A) 2 B) 6 C) 14 D) 21 **E) 42**'
output: 42, 6, 21, 12

An interesting example [1] as the numbers_gcd data set only ever presents two values to find the gcd of, while the above presents three. Not only does the model predict the correct solution, but two distractors also used in the actual exam.

'Evaluate the function f(x) = 3 + (x-5)**(1/2) at x = 9. A) 1 **B) 5** C) 6 D) 7'
output: 5, 49, -5, 9

Again we observe relatively reasonable responses given question formulations which diverge significantly from the templates trained on. This question [25] is similar to a polynomials_evaluate type. However, the difference is no fractional powers exist in the train-easy set–yet even with the power symbol being replaced by the unknown character the model still generates valid distractors (and the correct solution, but this is clearly by chance as the model has no knowledge of roots).

## 5. CONCLUSION

### 5.1 Summary

Two experiments quantitatively showed that a GRU has mixed results when attempting to solve elementary math problems. Our alternative goal of multiple choice distractor generation for several question types typically found in pre-undergraduate education by applying a modified greedy decoding schema referred to as character offsetting was successful. Evaluation using an edit distance based partial credit scoring metric as opposed to a binary one demonstrates greatly increased consistency and performance for capturing a reasonable response. We found the following:

- Generally the easiest math problem types for a GRU is comparison tasks which is not surprisingly since this is a fundamental problem encountered early in education. It would appear the ability for GRUs to abstract mathematical knowledge is minimal.
- The ability for networks to capture the essence of a reasonable response for several question types is shown. Leveraging the proposed practice of character offsetting we show that these networks can cheaply generate distractor options for multiple choice questions.

### 5.2 Future Work

It would be interesting to compare a beam search decoding's non principle predicted sequences to those produced by character offsetting and whether for certain question types more worthwhile distractors are produced. The general capability for character offsetting to produce at least one worthwhile distractor for the medium potential questions listed in table 4 hint that with some refinements to the decoding schema or training parameters could potentially become high potential question types.

## 6. ACKNOWLEDGEMENTS

We gratefully acknowledge support from Teaching and Learning with Technology at The Pennsylvania State University.

## Supplementary Material - Appendix

| S.No. | Question | Answer | Distractors (Model) | Distractors (Rule Based) |
|---|---|---|---|---|
| 1 | express $-105c^2 - 5c^3 + 7c - 50c + 41c + 332c^2$ in the form $rc^3 + ic^2 + b + uc$ and give $u$. | $-2$ | $-1$<br>$0$<br>$3$<br>$2$ | $-9$<br>$0$<br>$-16$<br>$2$ |
| 2 | let $k(w) = 2w^2 - 4w^2 + 3w^2$. let $z = -29 + 33$. let $r(o) = -4 - 39o^2 + 84o^2 + z$. give $r(k(s))$. | $45s^4$ | $-s^4$<br>$8s^4$<br>$9s^4$<br>$18s^4$ | $-45s^4$<br>$45s^4 - 4$<br>$135s^4$<br>$0$ |
| 3 | let $u$ be $1/(114/56 - 2)$. suppose $-3d = -3p + 24$, $-16d - u = -4p - 13d$. solve $c - 3z + 2 = -c$, $0 = -4c - pz - 4$ for c. | $-1$ | $1$<br>$4$<br>$2$<br>$0$ | $1$<br>$-2$<br>$2$<br>$0$ |
| 4 | let v be $4/(-14) + -1 * (-596)/28$. let $a = v - 15$. what is the third derivative of $-6b^2 - 9b^6 + 23b^a - 8b^6$ wrt $b$? | $720b^3$ | $-120b^3$<br>$120b^3$<br>$60b^3$<br>$240b^3$ | $-120b^3$<br>$120b^3$<br>$-720b^3$<br>$360b^3$ |
| 5 | let $a(h) = 2h^2 - 9h + 4$. suppose $-26w - 20w = -55w + 126$. what is the remainder when $a(-7)$ is divided by $w$? | $11$ | $18$<br>$8$<br>$21$<br>$9$ | $7$<br>$8$<br>$3$<br>$9$ |

Table 3: Qualitative Comparison of Distractors Generated using Neural Network Model and Rule Based System

| Potential | Question | Encoder Hidden Size | Batch Size | Partial Credit Score | Mean Score |
|---|---|---|---|---|---|
| | algebra_linear_2d_composed | 2048 | 128 | 0.837 | |
| High | algebra_linear_2d | 2048 | 256 | 0.811 | 0.766 |
| | algebra_linear_1d_composed | 2048 | 128 | 0.887 | |
| | algebra_linear_1d | 2048 | 256 | 0.694 | |
| | algebra_sequence_next_term | 2048 | 128 | 0.774 | |
| | arithmetic_mul_div_multiple | 2048 | 256 | 0.772 | |
| | arithmetic_nearest_integer_root | 2048 | 256 | 0.730 | |
| | polynomials_evaluate_composed | 2048 | 128 | 0.754 | |
| | polynomials_evaluate | 2048 | 128 | 0.709 | |
| | polynomials_expand | 512 | 128 | 0.642 | |
| | polynomials_coefficient_named | 2048 | 128 | 0.733 | |
| | numbers_gcd_composed | 2048 | 128 | 0.760 | |
| | numbers_gcd | 2048 | 256 | 0.762 | |
| | numbers_lcm_composed | 2048 | 128 | 0.737 | |
| | numbers_div_remainder_composed | 2048 | 128 | 0.800 | |
| | numbers_div_remainder | 2048 | 256 | 0.762 | |
| | numbers_place_value_composed | 2048 | 128 | 0.859 | |
| | algebra_sequence_nth_term | 1024 | 128 | 0.507 | |
| Medium | arithmetic_add_or_sub | 2048 | 256 | 0.501 | 0.624 |
| | arithmetic_mul | 2048 | 256 | 0.451 | |
| | arithmetic_div | 2048 | 256 | 0.559 | |
| | arithmetic_mixed | 2048 | 256 | 0.688 | |
| | arithmetic_add_sub_multiple | 2048 | 256 | 0.755 | |
| | arithmetic_add_or_sub_in_base | 2048 | 256 | 0.682 | |
| | calculus_differentiate_composed | 1024 | 128 | 0.589 | |
| | calculus_differentiate | 512 | 128 | 0.730 | |
| | measurement_time | 2048 | 256 | 0.838 | |
| | numbers_lcm | 2048 | 256 | 0.564 | |

Table 4: Qualitative ranking of the potential for models to use character offsetting for generating distractors based on observed predictions on interpolate. Questions not listed are those whose predictions were generally unreasonable as defined in 4.2.2 or disqualified due to formatting mentioned in 4.2.1. Model specifications are included as well. Decoder hidden size was 2048 for all models. Encoder/Decoder embedding dimension and number of attentional heads was 512/512 and 16 respectively.

## 7. REFERENCES

[1] ACT. Practice test questions: Math, 2020.

[2] I. Aldabe and M. Maritxalar. Automatic distractor generation for domain specific texts. In *International Conference on Natural Language Processing*, pages 27–38. Springer, 2010.

[3] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2015.

[4] K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014.

[5] J. A. Erickson. Natural language processing for open ended questions in mathematics within intelligent tutoring systems. In *EDM*, 2020.

[6] M. Ganesalingam. The language of mathematics. In *The language of mathematics*, pages 17–38. Springer, 2013.

[7] H. C. Goodrich. Distractor efficiency in foreign language testing. *TESOL Quarterly*, 11(1):69–78, 1977.

[8] A. Graves. *Supervised Sequence Labelling with Recurrent Neural Networks*. Four volumes. Springer, 2014.

[9] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015.

[10] R. Koncel-Kedziorski, H. Hajishirzi, A. Sabharwal, O. Etzioni, and S. D. Ang. Parsing algebraic word problems into equations. *Transactions of the Association for Computational Linguistics*, 3:585–597, 2015.

[11] N. Kushman, Y. Artzi, L. Zettlemoyer, and R. Barzilay. Learning to automatically solve algebra word problems. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 271–281, Baltimore, Maryland, June 2014. Association for Computational Linguistics.

[12] G. Lample and F. Charton. Deep learning for symbolic mathematics. *CoRR*, abs/1912.01412, 2019.

[13] C. Liang, X. Yang, N. Dave, D. Wham, B. Pursel, and C. L. Giles. Distractor generation for multiple choice questions using learning to rank. In *Proceedings of the Thirteenth Workshop on Innovative Use of NLP for Building Educational Applications*, pages 284–290, 2018.

[14] C. Liang, X. Yang, D. Wham, B. Pursel, R. Passonneau, and C. L. Giles. Distractor generation with generative adversarial nets for automatically creating fill-in-the-blank questions. In *Proceedings of the Knowledge Capture Conference*, page 33. ACM, 2017.

[15] W. Ling, D. Yogatama, C. Dyer, and P. Blunsom. Program induction by rationale generation: Learning to solve and explain algebraic word problems. *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2017.

[16] T. Liu, Q. Fang, W. Ding, Z. Wu, and Z. Liu. Mathematical word problem generation from commonsense knowledge graph and equations. *ArXiv*, abs/2010.06196, 2020.

[17] J. J. Michalenko, A. S. Lan, and R. G. Baraniuk. Personalized feedback for open-response mathematical questions using long short-term memory networks. In X. Hu, T. Barnes, A. Hershkovitz, and L. Paquette, editors, *Proceedings of the 10th International Conference on Educational Data Mining, EDM 2017, Wuhan, Hubei, China, June 25-28, 2017*. International Educational Data Mining Society (IEDMS), 2017.

[18] V.-M. Pho, A.-L. Ligozat, and B. Grau. Distractor quality evaluation in multiple choice questions. In C. Conati, N. Heffernan, A. Mitrovic, and M. F. Verdejo, editors, *Artificial Intelligence in Education*, pages 377–386, Cham, 2015. Springer International Publishing.

[19] O. Polozov, E. O'Rourke, A. M. Smith, L. Zettlemoyer, S. Gulwani, and Z. Popović. Personalized mathematical word problem generation. In *IJCAI 2015*, May 2015.

[20] S. Roy and D. Roth. Solving general arithmetic word problems. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1743–1752, Lisbon, Portugal, Sept. 2015. Association for Computational Linguistics.

[21] S. Roy and D. Roth. Unit dependency graph and its application to arithmetic word problem solving. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI'17, page 3082–3088. AAAI Press, 2017.

[22] D. Saxton, E. Grefenstette, F. Hill, and P. Kohli. Analysing mathematical reasoning abilities of neural models. *CoRR*, abs/1904.01557, 2019.

[23] Y. Susanti, T. Tokunaga, H. Nishikawa, and H. Obari. Automatic distractor generation for multiple-choice english vocabulary questions. *Research and Practice in Technology Enhanced Learning*, 13(1):15, 2018.

[24] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, page 3104–3112, Cambridge, MA, USA, 2014. MIT Press.

[25] University Department of Mathematics. Math 021 sample exams, exam 1 sample exam a, 2020.

[26] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[27] L. Wang, D. Zhang, L. Gao, J. Song, L. Guo, and H. T. Shen. Mathdqn: Solving arithmetic word problems via deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[28] L. Wang, D. Zhang, J. Zhang, X. Xu, L. Gao, B. T. Dai, and H. T. Shen. Template-based math word problem solvers with recursive neural networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):7144–7151, Jul. 2019.

[29] Y. Wang, X. Liu, and S. Shi. Deep neural solver for math word problems. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 845–854, Copenhagen, Denmark, Sept. 2017. Association for Computational Linguistics.

[30] K. Zaporojets, G. Bekoulis, J. Deleu, T. Demeester, and C. Develder. Solving math word problems by

scoring equations with recursive neural networks. *arXiv preprint arXiv:2009.05639*, 2020.