Math Operation Embeddings for Open-ended Solution Analysis and Feedback

Mengxue Zhang[1], Zichao Wang[2], Richard Baraniuk[2], Andrew Lan[1]*
[1]University of Massachusetts Amherst, [2]Rice University

ABSTRACT

Feedback on student answers and even during intermediate steps in their solutions to open-ended questions is an important element in math education. Such feedback can help students correct their errors and ultimately lead to improved learning outcomes. Most existing approaches for automated student solution analysis and feedback require manually constructing cognitive models and anticipating student errors for each question. This process requires significant human effort and does not scale to most questions used in homeworks and practices that do not come with this information. In this paper, we analyze students' step-by-step solution processes to equation solving questions in an attempt to scale up error diagnostics and feedback mechanisms developed for a small number of questions to a much larger number of questions. Leveraging a recent math expression encoding method, we represent each math operation applied in solution steps as a transition in the math embedding vector space. We use a dataset that contains student solution steps in the Cognitive Tutor system to learn implicit and explicit representations of math operations. We explore whether these representations can i) identify math operations a student intends to perform in each solution step, regardless of whether they did it correctly or not, and ii) select the appropriate feedback type for incorrect steps. Experimental results show that our learned math operation representations generalize well across different data distributions.

Keywords

Embeddings, Feedback, Math expressions, Math operations

1. INTRODUCTION

Math education is of crucial importance to a competitive future science, technology, engineering, and mathematics (STEM) workforce since math knowledge and skills are required in many STEM subjects [11]. One important way to help struggling students improve in math is to diagnose errors from student answers to math questions and deliver personalized support to help them correct these errors [1]. In short-answer questions, feedback of various types [39] can be deployed according to the specific incorrect final answers students submit, while in open-ended questions, feedback can be deployed at intermediate solution steps according to the specific actions they take and their outcomes [22]. In traditional educational settings, this feedback process relies on teachers going over student work, identifying errors, and providing feedback [15], which results in a labor-intensive process and a slow feedback cycle for students. Such a setting is even more limited as a result of the COVID-19 pandemic, which introduced new barriers to face-to-face interactions between teachers and students.

In intelligent tutoring systems, a more scalable approach to math feedback is to automatically deploy feedback based on students' final answers or certain incorrect intermediate solution steps. For example, in ASSISTments [12], teachers can create hints and feedback messages for specific incorrect student answers to short-answer questions that they anticipate [28], which the system can automatically deploy when students submit these incorrect answers. This crowdsourcing approach efficiently scales up teachers' effort so that they can benefit a large number of students without putting in additional effort. In many other systems such as Cognitive Tutor [34] and Algebra Notepad [27], researchers use cognitive models to anticipate student errors as results of buggy production rules or insufficient knowledge on key math concepts [20, 24]. They then develop corresponding feedback for intermediate solution steps in multi-step questions (e.g., those on equation solving). This cognitive model-based approach requires significant effort by domain experts and has shown to be highly effective in large-scale studies.

However, these approaches for student feedback are still limited in their generalizability to many math questions deployed in daily homeworks and practices. For the teacher crowdsourcing approach, hint and feedback messages have to be written for each individual question (or group of questions generated from the same template with different numerical values). For the cognitive model-based approach, a rigorous solution process has to be specified for each question with annotations on the math operations that should be applied at each solution step. However, questions used in many real-world educational settings do not come with such information; teachers simply adopt them from sources

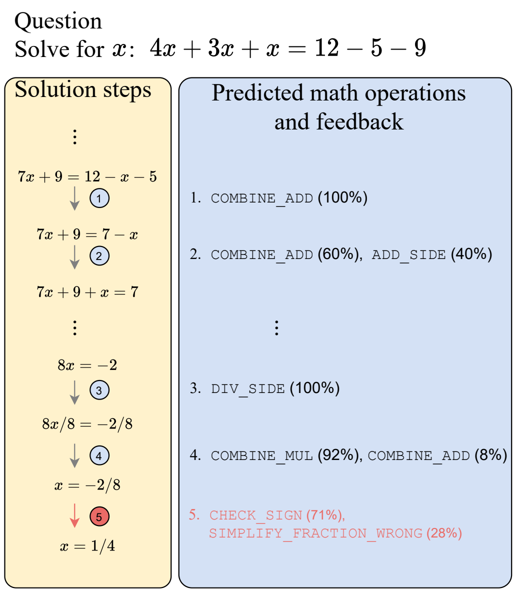
*This work is supported by the National Science Foundation under grant IIS-1917713.

such as textbooks and open education resources and assign them to students without developing corresponding feedback mechanisms. Moreover, past research has shown that a large portion of incorrect student answers cannot be anticipated by cognitive models [43], teachers/domain experts [8], or numerical simulations [37]. Therefore, it may be hard for high-quality feedback developed for questions used in intelligent tutoring systems to generalize to questions in the wild.

1.1 Contributions

In this paper, we develop *data-driven* methods that enable us to analyze step-by-step solutions to open-ended math questions. In contrast to existing methods that rely on a *top-down* approach, i.e., defining the structure of the solution process and anticipating student errors, we propose a *bottom-up approach*, i.e., using learned representations of *math expressions* and *math operations* to predict i) math operations in student solution steps and ii) the appropriate feedback for incorrect solution steps. We restrict ourselves to the specific domain of *equation solving* where the solution process consists of applying specific math operations between math expressions in consecutive steps; other sub-domains of math such as algebra word problems [45] and questions involving graphs and geometry [16] are left as future work. Specifically, our contributions are:

- First, we characterize math operations by how they *transform* math expressions in the math embedding space in each solution step. We leverage recent work on learning *math symbol embeddings* from large-scale scientific formula data [46] to encode math expressions in student solutions: each math expression is mapped to a point in the *math embedding vector space.* We use synthetically generated data as well as solution steps generated by real students to learn the representation of each math operation. We explore several methods for learning both implicit and explicit math operation representations: a classification-based method that does not explicitly impose a structure on math operations, a linear model that assumes each operation is characterized by an *additive vector* in the embedding space, and a nonlinear model where math operations live in their own, interconnected embedding spaces.
- Second, we apply these math operation representation learning methods to a real-world student step-by-step solution dataset collected while student learn equation solving in an intelligent tutoring system, Cognitive Tutor [34]. We validate our math operation representation learning methods via two tasks: i) predicting the specific math operation the student intended to apply in a solution step from the math expressions before and after the step and ii) predicting the appropriate feedback deployed to students from the incorrect math expressions they enter. Quantitative results show that tree embedding-based math expression encoding methods outperform other encoding methods since they are able to explicitly capture the semantic and structural characteristics of math expressions. They also have better generalizability across different data distributions and remain effective across different question difficulty levels and even when student solutions steps contain errors.



Question
Solve for x: $4x + 3x + x = 12 - 5 - 9$
Solution steps
Predicted math operations and feedback
$7x + 9 = 12 - x - 5$
$7x + 9 = 7 - x$
$7x + 9 + x = 7$
$8x = -2$
$8x/8 = -2/8$
$x = -2/8$
$x = 1/4$
1. COMBINE_ADD (100%)
2. COMBINE_ADD (60%), ADD_SIDE (40%)
3. DIV_SIDE (100%)
4. COMBINE_MUL (92%), COMBINE_ADD (8%)
5. CHECK_SIGN (71%), SIMPLIFY_FRACTION_WRONG (28%)

Figure 1. Demonstration of the generalizability of our math operation representations to other data sources for a solution process provided on Algebra.com. Our methods can successfully predict the math operations applied in each step and the appropriate feedback type in an incorrect step.

1.2 Use Case

Before diving into the technical details, we first illustrate a potential use case for our math operation representation learning methods and corresponding operation/feedback classifiers. Our goal is to transfer expert designs in intelligent tutoring systems for math education to questions in the wild. Specifically, we apply the math operation representations learned from student solution steps and corresponding labels (step name, feedback message) in the highly structured Cognitive Tutor system to environments that are not highly structured. Figure 1 shows the solution process to an equation solving question on Algebra.com[1] and the corresponding math operation and feedback predictions at each step. We see that our math operation representation learning methods can accurately predict the math operations applied in solution steps 1, 3, and 4 using the operation names provided in the Cognitive Tutor system. Even in step 2 where two different math operations are combined into a single step, i.e.,

$$\begin{aligned}&7x + 9 = 7 - x\\ &\downarrow \text{ADD } x \text{ TO BOTH SIDES}\\ &7x + 9 + x = 7 - x + x\\ &\downarrow \text{COMBINE TERMS ON RIGHT SIDE}\\ &7x + 9 + x = 7,\end{aligned}$$

despite only training on steps in Cognitive Tutor that involve only one math operation, the classifier is able to recognize both of them with high predictive probability for both. We

[1]The original question and the solution process can be found at `https://www.algebra.com/algebra/homework/equations/Equations.faq.question.4872.html`.

also change one of the solution steps, i.e., step 5, to make it incorrect and test our feedback classifier. In this case, the classifier is able to recognize the error in this step and find the corresponding feedback types in Cognitive Tutor. This potential use case demonstrates the utility of our math operation representation learning methods: by transferring knowledge learned in well-designed, highly-structured systems such as Cognitive Tutor, especially on what feedback to deploy for each student error, to other domains such as online math Q&A sites, we are *scaling up* the effort domain experts put into the design of these feedback mechanisms.

2. RELATED WORK

One related body of work in math education that studies student solution processes to identify student strategies and assess errors. Specifically, [33] uses inverse Bayesian planning to learn solution strategies (i.e., policies) in equation solving and capture student misunderstandings in a Markov decision process framework. Our work focuses on a different aspect of the solution process: the representation of the math expressions at each solution step and the modeling of the transitions between different math expressions under math operations. [9] uses basic math operations to construct programs to understand errors that students make in their solutions to arithmetic questions. Our work focuses on equation solving, which is a more difficult problem in which students responses are are more diverse and are less structured than arithmetic calculations.

Another related body of work focuses on learning representations of student answers to short-answer questions. [21] analyzes incorrect student answers across multiple questions, learn representations of errors, and generalize misconception feedback across questions. Our work analyze the full math expressions in intermediate solution steps while their work represents short answers according to the frequency they occur in an answer pool. [8] uses trained word embeddings to represent short answers for automated grading purposes. Our work focuses on learning transitions of math expressions across solution steps instead of learning representations of only the final answer.

In domains other than math education, there exist methods for automated feedback generation, including programming [30, 31, 40] and essays [35]. However, transferring these methods to math solutions is not trivial since i) open-ended math solutions are less structured than programming code and ii) data-driven representations of math symbols have not been developed until recently [46] whereas such representations have been studied for a long time in natural language processing [6, 7, 26].

Another body of remotely-related work focuses on using computer vision techniques to identify math expressions from images for similar math expression retrieval [29], turning hand-written math expressions into LaTeX [47], and automatically identifying and correcting student errors [14]. These works often bypass the inherent structure of math expressions and directly use an end-to-end model for their tasks, which means that they cannot be used to analyze student knowledge. Nevertheless, these techniques can be used to build large-scale datasets containing hand-written student solutions which we can use in the future.

3. BACKGROUND: EMBEDDING MATH EXPRESSIONS INTO VECTOR SPACES

In this section, we provide an overview of a recent method that we developed to embed math expressions into a vector space, i.e., a math embedding space. Doing so turns discrete, symbolic math expression representations into continuous, distributed representations [2], which enables us to manipulate math expressions in a manner compatible with modern machine learning methodologies.

Our embedding method is a *tree-structured* encoder illustrated in Figure 2. The key observation is that any math expression has a corresponding symbolic *tree-structured* representation in the *operator tree* format. In the operator tree, the non-terminal (non-leaf) nodes are math operators, i.e., addition and subtraction, and terminal (leaf) nodes are numbers or variables; See Figure 2 for an illustration. Thus, an operator tree explicitly captures the semantic and structural properties of a math expression. A number of existing works have demonstrated the superior performance of using operator tree representations of math expressions compared to other math expression representations in applications such as automatic math word problem solving [32, 48, 51] and math formulae retrieval [5, 25, 49, 50].

Therefore, we built a math expression encoder that leverages the operator tree representation of math expressions. Specifically, during the encoding process, it first converts a math expression into its corresponding tree format, using the parser introduced in [5]. It then linearizes the tree by depth first search that enables us to process nodes as a sequence in which each math symbol is associated with its own trainable embedding. Next, it leverages positional encoding, similar to [44, 38], to retain the relative position of each node in the tree. The output of our encoder is a fixed-dimensional embedding vector that represents the input math expression, which we will use to learn representations of math operations for the math operation classification and feedback prediction tasks. We pretrain the encoder on a large corpus of math expressions extracted from Wikipedia and arXiv articles and demonstrated superior performance in reconstructing math expressions (and scientific formulae) and retrieving similar expressions. See the anonymized version of our work at [46]. We will refer to the trained encoder as the *math expression encoding method* in what follows.

4. LEARNING REPRESENTATIONS OF MATH OPERATIONS

In this section, we detail methods we use to learn both implicit and explicit math operation representations by studying how they transform math expressions in each solution step in the math embedding space. In these methods, we leverage the math expression encoding method developed in our prior work that we reviewed above to embed math expressions into vectors and work with these embedding vectors. However, since these embeddings are trained on math expressions that are very different from those occurring in actual student solution steps, we use an additional trainable, fully-connected neural network to adapt these embeddings to our dataset, following a popular approach in natural language processing [13]. Specifically, we have $\mathbf{e} = g_{\gamma}(\mathbf{m})$ where $\mathbf{m}$ and $\mathbf{e}$ are the embedded vector of a math expression

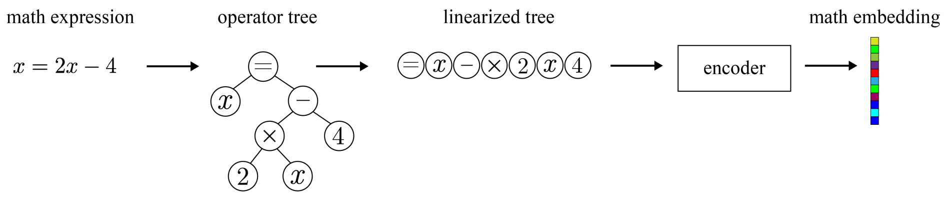


Figure 2. Illustration of the math expression encoding method that we employ in this work.

in our dataset before and after the adaptation, respectively. $\boldsymbol{\gamma}$ denotes the set of parameters in the fully-connected network that we will learn during the training process.

We define a step in a student's solution to open-ended math questions as a tuple $(\mathcal{E}_1, \mathcal{E}_2, z)$, where $z \in \mathbf{Z}$ is the math operation applied in this step, with $\mathcal{Z}$ denoting the set of possible math operations. $\mathcal{E}_1 \in \mathbf{E}$ and $\mathcal{E}_2 \in \mathbf{E}$ denote the math expressions involved in this step before and after applying this math operation, i.e., the step can be expressed as $\mathcal{E}_1 \xrightarrow{z} \mathcal{E}_2$. $\mathbf{E}$ denotes the set of all unique math expressions (across all steps in a dataset). For simplicity, we assume that only one math operation is applied in each step; an extension to cases where multiple math operations is trivial and will be discussed in what follows. $\mathbf{e}_1 \in \mathbb{R}^D$ and $\mathbf{e}_2 \in \mathbb{R}^D$ are the fine-tuned embedding vectors that correspond to math expressions $\mathcal{E}_1$ and $\mathcal{E}_2$, respectively, where D is the dimension of the embedding.

4.1 Math Operation Classification

The first task we will study in this paper is to classify the math operation applied in a solution step given the math expression embeddings before and after appliying it, $\mathbf{e}_1$ and $\mathbf{e}_2$. The same notations and approaches also apply to our second task, feedback classification. This task can simply be solved using a supervised learning method, e.g., a regression model where the predicted probability of predicting a math operation $\hat{z}$ is given by

$$p(\hat{z} = z) = \mathrm{softmax}(\mathbf{v}_z^T [\mathbf{e}_1^T,\ \mathbf{e}_2^T]^T),$$

where softmax$(\cdot)$ is the softmax function for multi-label classification [10]. $\mathbf{v}_z$ is a parameter vector associated with each math operation z, which is used to compute an inner product with the concatenation of $\mathbf{e}_1$ and $\mathbf{e}_2$ before being fed into the softmax function. On a training dataset with given tuples $(\mathbf{e}_1, \mathbf{e}_2, z)$, we can learn the parameters $(\mathbf{v}_z)$ by minimizing the cross-entropy loss [10] between the predicted math operation $\hat{z}$ and the actual math operation. This approach can be seen as learning *implicit* representations of math expressions since they are captured by the classifier parameters.

4.2 Learning Math Operation Representations

The classification approach we detailed above can help us classify the math operation applied in a solution step but falls short on learning *explicit* representations of math operations. The latter is important, however, to help us understand students' math solution processes and diagnose their errors. We now detail a series of methods for us to learn explicit representations of math operations.

4.2.1 Translating embeddings

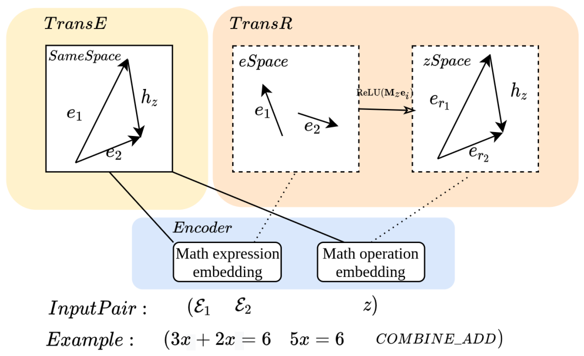


Figure 3. Illustration of the TransE and TransR frameworks. TransE puts the embeddings of equations e_1, e_2, and math operation z in the same embedding space, whereas TransR puts them in their own embedding spaces.

We will leverage the translating embedding (TransE) framework [3] that has found success in embedding entities and characterizing relationships between entities in multi-relational data. Our key assumption here in this framework is that math operations are *linear and additive*, i.e., the relationship between math expressions before and after a math expression satisfy

$$\mathbf{e}_2 \approx \mathbf{e}_1 + \mathbf{h}_z,$$

where $\mathbf{h}_z \in \mathbb{R}^D$ is the embedding of the *math operation* z. In other words, we assume that the effect of a math operation is characterized by the difference in the embedding vectors between the math expressions before and after it in a single step; adding it to the embedded vector of $\mathcal{E}_1$ results in the embedded vector of $\mathcal{E}_2$ after the step.

To learn these math operation embeddings from data, we use two loss functions. The first loss function promotes this linear and additive relationship between embeddings of the math expressions and operations on the training data. To this end, we define a distance function as $d(\mathbf{e}_1, \mathbf{e}_2, \mathbf{h}_z) = \|\mathbf{e}_1 + \mathbf{h}_z - \mathbf{e}_2\|_2^2$ and define the loss function as

$$L_1 = \sum_{(\mathcal{E}_1, \mathcal{E}_2, z)} d(\mathbf{e}_1, \mathbf{e}_2, \mathbf{h}_z).$$

The second loss function pushes counterfeit step tuples that are generated by replacing elements in an observed step tuple with other ones in the dataset to not satisfy the aforementioned linear and additive relationship. To this end, we minimize the pairwise marginal distance ranking-based loss given by

$$L_2 = \sum_{(\mathcal{E}_1,\mathcal{E}_2,z)\in\mathbf{S}} \sum_{(\mathcal{E}_1',\mathcal{E}_2',z')\in\mathbf{S}'_{(\mathcal{E}_1,\mathcal{E}_2,z)}} [\gamma + d(\mathbf{e}_1,\mathbf{e}_2,\mathbf{h}_z) - d(\mathbf{e}_1',\mathbf{e}_2',\mathbf{h}_z')]_+,$$

where $[x]_+ = x$ when $x > 0$ and 0 otherwise and $\gamma > 0$ is a hyper-parameter that controls the margin of the distance ranking. $\mathbf{S}$ denotes the set of steps in the dataset and $\mathbf{S}'_{(\mathcal{E}_1,\mathcal{E}_2,z)}$ is a set of counterfeit steps that are perturbed versions of the actual step $(\mathcal{E}_1,\mathcal{E}_2,z)$, generated by randomly replacing one of the triplet elements in the step by a different math expression or math operation from another step, i.e.,

$$\begin{aligned}\mathbf{S}'_{(\mathcal{E}_1,\mathcal{E}_2,z)} &\sim A\cup B\cup C,\\ \text{where } A &= \{(\mathcal{E}_1',\mathcal{E}_2,z) : \mathcal{E}_1' \neq \mathcal{E}_1 \in \mathbf{E}\}\\ B &= \{(\mathcal{E}_1,\mathcal{E}_2',z) : \mathcal{E}_2' \neq \mathcal{E}_2 \in \mathbf{E}\}\\ C &= \{(\mathcal{E}_1,\mathcal{E}_2,z') : z' \neq z \in \mathbf{Z}\}.\end{aligned}$$

Intuitively speaking, our objective encourages the distance function calculated on an actual tuple in the dataset to be smaller than that calculated on a perturbed version of it. Figure 3 illustrates the whole process.

The final loss function that we minimize is simply the combination of these two loss functions as $L = L_1 + L_2$. Using the learned embeddings of each math operation, we can classify them from the math expressions $\mathcal{E}_1$ and $\mathcal{E}_2$ using the nearest neighbor classifier, i.e., $\hat{z} = \text{argmin}_z d(\mathbf{e}_1,\mathbf{e}_2,\mathbf{h}_z)$.

4.2.2 *Learning Entity and Relation Embeddings*

Despite potentially exhibiting excellent interpretability, TransE's assumption that math operations are linear and additive in the math expression embedding space may be too restrictive. This assumption puts math operations are vectors in the same latent space where similar math expressions will be close to each other. However, different math operations are fundamentally different and can transform the same math expression into dramatically different math expressions that are far apart in the embedding space. For example, different math operations can focus on transforming different parts of the same math expression. The steps $(3+5+2x = x+1,\ 8+2x = x+1,$ combine similar terms$)$ and $(3+5+2x = x+1,\ 3+5+2x-x = x+1-x,$ subtract from each side$)$ have the same starting math expression $\mathcal{E}_1$. In the first step, only similar terms on the left hand side of the equation are combined, regardless of the other side of the equation. In the second step, we subtracted x from both sides of the equation, which is a consequence of the equality symbol in the equation, which means that subtracting the same term on both sides of the equation but not what exactly is on each side. Therefore, TransE's linear and additive assumption means that the resulting $\mathcal{E}_2$ in these steps will be very different due to the different math operations applied, which conflicts with the observation that they are very similar. To address this limitation, we explore the Learning Entity and Relation Embeddings (TransR) [23] model, which models math expressions and math operations in different spaces, i.e., there will be a shared embedding space for all math expressions but separate relation spaces for different math operations.

TransR learns the embeddings of math operations by projecting them to their corresponding relation spaces and then learning translations between those projected expressions. For each math operation z, we set a projection matrix $\mathbf{M}_z \in \mathbb{R}^{D\times D}$ that projects a math expression to its relation space. To make this projection nonlinear, we apply the rectified linear unit (ReLU) activation function [10] to it and define the corresponding distance function as

$$d_z(\mathbf{e}_1,\mathbf{e}_2,\mathbf{h}_z) = \|\text{ReLU}(\mathbf{M}_z\mathbf{e}_1) + \mathbf{h}_z - \text{ReLU}(\mathbf{M}_z\mathbf{e}_2)\|_2^2.$$

Correspondingly, the two loss functions in the TransR framework are given by

$$\begin{aligned}L_1 &= \sum_{(\mathcal{E}_1,\mathcal{E}_2,z)} d_z(\mathbf{e}_1,\mathbf{e}_2,\mathbf{h}_z),\\ L_2 &= \sum_{(\mathcal{E}_1,\mathcal{E}_2,z)\in\mathbf{S}} \sum_{(\mathcal{E}_1',\mathcal{E}_2',z')\in\mathbf{S}'_{(\mathcal{E}_1,\mathcal{E}_2,z)}} [\gamma + d_z(\mathbf{e}_1,\mathbf{e}_2,\mathbf{h}_z) - d_z(\mathbf{e}_1',\mathbf{e}_2',\mathbf{h}_z')]_+.\end{aligned}$$

The projection matrices $\mathbf{M}_z$, $\forall z \in \mathbf{Z}$ are included as part of the trainable parameters. The rest of the training and resulting math operation classification procedure remains unchanged from the TransE framework.

5. EXPERIMENTS

We now detail a series of quantitative and qualitative experiments that we have conducted to validate the learned representations of math operations. Using the Cognitive Tutor 2010 equation solving (CogTutor) dataset,[2] we focus on two tasks: i) classifying the math operation a student applies in a solution step and ii) classifying the feedback category corresponding to certain types of incorrect steps, from the math expressions the student enters before and after the step.

5.1 Dataset

We use the CogTutor dataset which we accessed via the PSLC DataShop [19]. The dataset contains detailed tutor logs generated as students in a school use the Cognitive Tutor system [34] for their Algebra I class. These logs contain the students' step-by-step solutions to equation solving problems, where each step is a tuple with three elements: a *math expression* $\mathcal{E}_1$ at the beginning of the step, the *step name* z, i.e., the math operation the student selected to apply to this math expression, and the resulting math expression $\mathcal{E}_2$ after the step. Students can select math operations from a built-in list in Cognitive Tutor: COMBIN_ADD, COMBINE_MUL, ADD_SIDE, SUB_SIDE, MUL_SIDE, DIV_SIDE, and DISTRIBUTE; see Table 1 for an illustration of these operations and some examples of the corresponding math operations before and after them in a step.

There are a total of $50,406$ steps in this dataset that can be further divided into three subsets according to their outcomes: `OK` ($43,413$ steps), `ERROR` ($6,377$ steps), and `BUG` ($5,744$ steps). The `OK` subset contains steps that are correct, i.e., the student both selected the correct math operation and arrived at the correct math expression. The `BUG` and `ERROR` subsets contain incorrect student steps, either because the operation they selected was incorrect or

[2]`https://pslcdatashop.web.cmu.edu/DatasetInfo?datasetId=660`

Step (Math operation)	Description	Example
COMBINE_ADD	combine two similar terms with add/sub operator	$3x + 2x \rightarrow 5x$
COMBINE_MUL	combine two similar terms with multiply/divide operator	$x * x \rightarrow x^2$
ADD_SIDE	add a math term on each side	$x = 1 \rightarrow x + 1 = 1 + 1$
SUB_SIDE	subtract a math term on each side	$x = 1 \rightarrow x - 1 = 1 - 1$
MUL_SIDE	multiply a math term on each side	$x = 1 \rightarrow x * 2 = 2$
DIV_SIDE	divide a math term on each side	$x = 1 \rightarrow x/2 = 1/2$
DISTRIBUTE	distribute(expand) the terms	$(x + 1)x \rightarrow x * x + x$

Table 1. Detailed descriptions and examples for each math operation in the CogTutor dataset.

because they selected the correct operation but did not apply it correctly, i.e., arriving at an incorrect math operation after the step. The difference between these two subsets is that `BUG` contains steps that fit one of the predefined error templates in the Cognitive Tutor system; in this case, the system can automatically diagnose the error and deploy a predefined feedback. On the other hand, `ERROR` contains incorrect steps that Cognitive Tutor could not automatically diagnose the underlying error. The `OK` subset can be further split into six predefined difficulty levels (named as ES_01,ES_02, ES_03 ,ES_04, ES_05, and ES_07), with $2,068$, $7,546$, $8,183$, $13,393$, $5,484$, and $2,801$ steps, respectively. We do not further split the `BUG` and `ERROR` subsets for the math operation classification task due to their limited sizes.

To learn the representation of math operations, we need examples of how they transform one math expression into another. However, the CogTutor dataset may not contain enough data that is rich in both quantity and diversity for neural network-based models to learn from. Therefore, we designed a synthetic data generator stemming from the math question answering dataset created by DeepMind [36]. The generator can generate steps by first generating the initial math expression and then applying math operations listed in Table 1 to arrive at a resulting math expression. We have full control over the generated steps through the entropy, degree, and flip parameters. Increasing entropy introduces more complexity to the math expressions as numerical constants generated get larger. Increasing the degree parameter introduces monomials of higher degrees and also adds more terms in the math expression. Finally, the flip parameter allows us to control which side of an equation has a higher chance to be more complicated than the other. Tuning these parameters within this flexible synthetic data generation method enables us to generate a large amount of steps that closely resembles those in the CogTutor dataset.

5.2 Methods

To fully evaluate the effectiveness of our math operation representations, we also experiment with two other ways of encoding math expressions commonly used in natural language processing tasks, in addition to the tree embedding-based and translation-based encoder that we introduced in Section 4.2. These two encoders include a gated recurrent unit (GRU)-based encoder [4] and a convolutional neural network (CNN)-based encoder [17]; we will use the output of these encoders to replace $[\mathbf{e}_1^T,\ \mathbf{e}_2^T]^T$ as input to the classifier detailed in Section 4.1.

Specifically, these two encoders first concatenates the two math expressions before and after the step, i.e., $\mathcal{E} = [\mathcal{E}_1, \mathcal{E}_2]$. For each character x_t in $\mathcal{E}$, we compute its embedding

$$\boldsymbol{x}_t = \mathbf{W}^T \text{onehot}(x_t)\,,$$

where $\mathbf{W}$ is a trainable embedding matrix. Using these character embeddings, the GRU encoder computes

$$\boldsymbol{h}_t = \text{GRU}_\theta(\boldsymbol{x}_t, \boldsymbol{h}_{t-1})\,,$$

where θ represents all the trainable parameters in GRU. We then replace $[\mathbf{e}_1^T,\ \mathbf{e}_2^T]^T$ with $\boldsymbol{h}_T$ as input to the classifier where T is the total number of characters in $\mathcal{E}$. Similarly, the CNN encoder computes

$$\boldsymbol{h} = \text{max\_pool}(\text{CNN}_\phi([\boldsymbol{x}_1, \cdots, \boldsymbol{x}_T]))\,,$$

where CNN_ϕ represents a 2D CNN with parameters ϕ and max_pool is a 1D max pooling operator. Combined, they return a fixed dimensional feature vector $\boldsymbol{h}$ that replaces $[\mathbf{e}_1^T,\ \mathbf{e}_2^T]^T$ as input to the classifier. For each of these two models, we learn its parameters jointly with the classification task using the cross-entropy loss that we described in Section 4.1.

Overall, we test five different methods for the math operation classification and feedback classification tasks. The first three methods use different encoding methods in conjunction with a classifier: i) using the GRU encoder to encode math expressions as input to the classifier, which we dub **GRU+C**, ii) using the tree embedding-based encoder instead, which we dub **TE+C**, and iii) using the CNN encoder instead, which we dub **CNN+C**. These methods do not learn explicit representations of math operations. The next two methods use the TransE and TransR frameworks to learn these representations using tree embeddings: iv) using tree embedding-based encoder as input to the TransE framework in conjunction with a nearest neighbor classifier, which we dub **TE+TransE**, and v) using the TransR framework instead of the TransE framework to study math operations in multiple relation spaces, which we dub **TE+TransR**.

5.3 Experimental Setup

We first test our math operation representation learning methods on the `OK` subset via 5-fold cross-validation, i.e., training on 80% of steps in the subset to learn representations of math operations and testing them on the remaining 20%. We also test the generalizability of the learned representations to incorrect steps, i.e., replace the test set with the `ERROR` and `BUG` subsets, and check whether we can still recognize the math operation a student applied in an incorrect step. The results are detailed in Section 5.4.1.

Since the distribution of math expressions in the `OK`, `ERROR` and `BUG` data subsets are mostly similar with minor differ-

	OK	ERROR	BUG
GRU+C	99.18 ± 0.23	93.87 ± 0.66	95.89 ± 0.63
TE+C	99.82 ± 0.04	93.30 ± 0.65	95.38 ± 0.62
CNN+C	95.37 ± 0.44	86.82 ± 1.38	91.02 ± 0.59
TE+TransE	96.27 ± 0.17	86.32 ± 1.23	84.21 ± 2.13
TE+TransR	99.17 ± 0.21	91.28 ± 1.12	91.31 ± 1.87

Table 2. Math operation classification accuracy for all methods training on the `OK` subset of the CogTutor dataset and testing on different data subsets. Accuracy is high across the board, while GRU-based encoding and tree embedding-based encoding in conjunction with a classifier result in the best performance.

ences, the previous experiment does not give us a good idea on the generalization ability of our math operation representation learning methods. Therefore, we further divide the `OK` subset into six smaller subsets, each corresponds to a different difficulty level (with different structure and complexity) according to questions within it, and test the generalizability of the learned math operation representations. The results are detailed in Section 5.4.2. In practice, solution step data generated by real students is often limited. Therefore, we conduct two more experiments to test whether synthetically generated steps can help us learn math operation representations that generalize to real data. First, we repeat the experiments above using synthetically generated steps as the training set. This synthetic training set consists of $1,000$ steps for each math operation defined in Table 1 (adding up to a total of $7,000$ across different difficulty levels). The results are detailed in Section 5.4.3. Second, to study the impact of synthetically generated data when real data is limited, we pre-train the math operation representations with synthetic data, fine-tune on a small amount of real data from each difficulty level in the `OK` subset, and test on the rest. The results are detailed in Section 5.4.4.

To test the ability of our learned math operation representations on recognizing student errors, we use them to classify feedback types provided by CogTutor in the `BUG` data subset. Examples of such errors include when a student calculated the wrong simplification result, used the wrong sign in front of terms, and applied useless/unlogical steps to solve the problem, etc. The results are detailed in Section 5.4.5.

We use Adam optimizer [18] with learning rate 0.001, batch size 64 and run 10 training epochs for each experiment. The math expression encoder outputs length-512 embedding vectors for each math expression, which we adapt to length-32 embedding vectors dimensions using a trainable fully-connected neural network. All of our experiments were conducted on a server with a single Nvidia RTX8000 GPU.

5.4 Results and Discussion

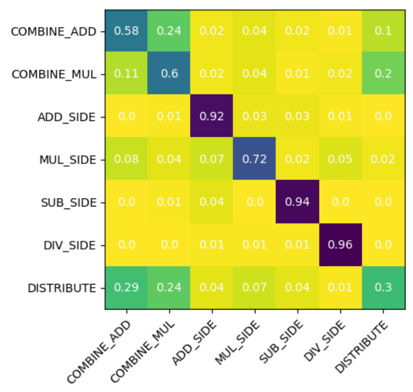
5.4.1 Generalizing to incorrect steps

Table 2 shows the averages and standard deviations of math operation classification accuracy for every method we experimented with using the `OK` subset as the training set. As expected, testing on the `ERROR` and `BUG` subsets result in slightly lower (5-10%) math operation classification accuracy for all methods since the training set does not contain incorrect steps. However, even on steps that are incorrect, these methods can still effectively identify the math operation a student intended to apply (with up to 95% accuracy), suggesting that they may be applicable to fully open-ended question solving solutions that are not highly structured, unlike those in Cognitive Tutor, to provide feedback to teachers on students' solution approaches.

We observe that using GRUs and tree embeddings as representations for math expressions and applying a classification method on top of these representations result in similar performances; GRUs slightly outperform tree embeddings in cases where we use the `ERROR` and `BUG` subsets as the test set while tree embeddings slightly outperform GRUs in the case where we use a part of the `OK` subset as the test set. Using CNNs to encode math expressions as input to a classifier results in worse performance, suggesting that they do not capture the semantic and structural information in math expressions as well as GRUs and tree embeddings. As expected, using tree embeddings under the TransE and TransR frameworks leads to worse performance than the first two methods, with TransE achieving low performance (especially on the `BUG` subset) and TransR achieving comparable performance to the classification-based methods on the `OK` subset but lower performance on the `ERROR` and `BUG` subsets. This result can be explained by the additional structural restriction that math operations are represented as linear and additive in some embedding space in the TransE framework, which makes it less robust against incorrect student solution steps. Using the TransR framework mitigates this problem due to its use of different relation spaces for each math operation.

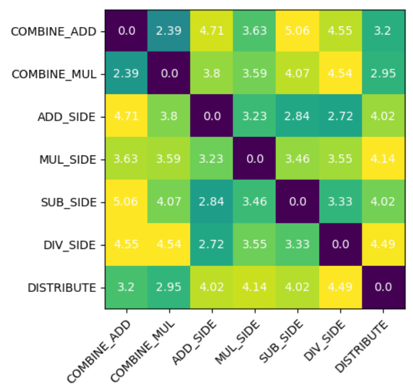
These methods perform similarly in the math operation classification task on real data largely due to the limited variation and complexity in the math expressions. The Cognitive Tutor system limits the degrees of freedom in a students' response by splitting an open-ended step into the separate actions of selecting a single math operation and entering the resulting math expression, which limits the variability in the data. In the next experiment, we see that when we control against different levels of complexity in these math expressions and forcing these methods to generalize across complexities, their performance vary significantly.

Figure 4 visualizes the confusion matrix for math operation classification on the `OK` subset and the pairwise euclidean distances between math operation embeddings learned via the TransE framework using tree embeddings for math expressions. Rows correspond to the true math operations applied in steps and columns correspond to predicted ones. Percentages in the confusion matrix (Figure 4a) are normalized w.r.t. the number of appearances of each math operation. We see that our math operation representation learning method captures some meaning of these operations (Figure 4b); the learned math operation embeddings capture the structural changes in math expression in ways that match our intuition. For instance, both COMBINE_ADD and COMBINE_MUL can be considered types of simplifications, so the Euclidean distance between the learned embeddings for these two operations is low. This observation is not surprising due to the similar nature of these operations. Moreover, COMBINE_ADD, COM-



	COMBINE_ADD	COMBINE_MUL	ADD_SIDE	MUL_SIDE	SUB_SIDE	DIV_SIDE	DISTRIBUTE
COMBINE_ADD	0.58	0.24	0.02	0.04	0.02	0.01	0.1
COMBINE_MUL	0.11	0.6	0.02	0.04	0.01	0.02	0.2
ADD_SIDE	0.0	0.01	0.92	0.03	0.03	0.01	0.0
MUL_SIDE	0.08	0.04	0.07	0.72	0.02	0.05	0.02
SUB_SIDE	0.0	0.01	0.04	0.0	0.94	0.0	0.0
DIV_SIDE	0.0	0.0	0.01	0.01	0.01	0.96	0.0
DISTRIBUTE	0.29	0.24	0.04	0.07	0.04	0.01	0.3

(a) Confusion matrix of math operations classification.



	COMBINE_ADD	COMBINE_MUL	ADD_SIDE	MUL_SIDE	SUB_SIDE	DIV_SIDE	DISTRIBUTE
COMBINE_ADD	0.0	2.39	4.71	3.63	5.06	4.55	3.2
COMBINE_MUL	2.39	0.0	3.8	3.59	4.07	4.54	2.95
ADD_SIDE	4.71	3.8	0.0	3.23	2.84	2.72	4.02
MUL_SIDE	3.63	3.59	3.23	0.0	3.46	3.55	4.14
SUB_SIDE	5.06	4.07	2.84	3.46	0.0	3.33	4.02
DIV_SIDE	4.55	4.54	2.72	3.55	3.33	0.0	4.49
DISTRIBUTE	3.2	2.95	4.02	4.14	4.02	4.49	0.0

(b) Euclidean distance between learned math operation embedding vectors.

Figure 4. Details of TE+TransE for the math operation classification task on the `OK` subset. These results match our intuition on how these math operations are related.

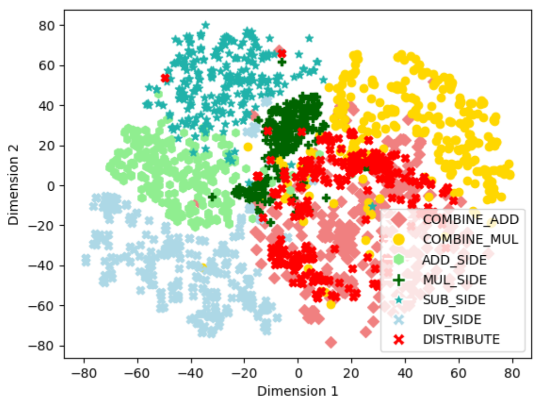


Figure 5. Visualization of learned math expression change for a randomly sampled subset of student solution steps in 2-D and corresponding operations (best viewed in color).

BINE_MUL, and DISTRIBUTE are often confused with one another. These results are also validated by a 2-D visualization (using t-SNE [42] as a dimensionality reduction method) of the learned math operation embeddings in Figure 5, where different math operations are mostly well separated except for COMBINE_ADD, COMBINE_MUL, and DISTRIBUTE. One possible explanation is that these operations are all applied to one side of the equation during a solution step, leaving one side of the equation unchanged, while the other operations, such as ADD_SIDE, SUB_SIDE, MUL_SIDE, and DIV_SIDE are all applied to both sides of the equation. Therefore, this result suggests that tree embeddings enable us to characterize a math operation by the *structural* change in math expressions before and after a solution step where it is applied. Furthermore, the classification accuracy for the DISTRIBUTE operation is significantly lower than that for other operations. This result is likely due to the fact that the number of steps with this operation is significantly lower than that for other operations.

5.4.2 *Generalizing to different difficulty levels*

In this experiment, we test the ability of our learned math operation representations to generalize to math expressions with different levels of complexity in questions at different levels of difficulty. Although they are all about equation solving, questions at different difficulty levels in Cognitive Tutor involve math expressions that look very different. For example, in the easiest level (ES_01), the equation that needs to be solved in a question looks like $x + 5 = 9$, with only a single variable and without numbers with decimals. In contrast, in the hardest level (ES_07), a question may contain coefficients with several decimal places and multiple variables, such as solve for m in the equation $m(k - n) = gs$. We only compare the GRU-based encoder and the tree embedding-based encoder in conjunction with a classifier since they are the best performing methods in the previous experiment. Table 3 lists the math operation classification accuracy for both methods after training on steps at different difficulty levels in the `OK` subset and testing on steps at other difficulty levels (including incorrect ones). We see that TE+C overall outperforms GRU+C in almost every case. This results suggest that tree embeddings are effective at capturing the structural property of a math expression. As a result, math operation representations based on tree embeddings excel at capturing the structural *change* in math expressions before and after applying a math operation, leading to better generalizability than GRU-based encoding that do not explicitly account for this change.

5.4.3 *Generalizing to different data distributions*

In this experiment, we test the ability of our methods to generalize from synthetically generated data to real student data. We train different math operation classification methods on the 2,000 synthetically generated steps and test them on steps generated by real students in the CogTutor dataset. Table 4 shows the mean and standard deviation for each method on each real data subset. We see that TE+C significantly outperforms GRU+C and CNN+C on all data subsets, which is in stark contrast to the previous experiment where the difference in performance across all methods is much smaller. This observation suggests that tree embeddings are more effective at capturing the semantic/structural effect of math operations on math expressions, thus generalizing better to different data distributions. Indeed, although the synthetically generated steps and the real steps have the same set of math operations, the distributions of numbers

Train on	Method	OK	ERROR	BUG
ES_01	GRU+C	58.82 ± 1.12	63.74 ± 1.13	66.02 ± 1.12
	TE+C	76.51 ± 0.62	84.24 ± 0.87	67.49 ± 1.10
ES_02	GRU+C	71.05 ± 1.12	76.66 ± 1.11	69.01 ± 1.14
	TE+C	87.89 ± 0.34	93.96 ± 0.72	80.44 ± 0.78
ES_03	GRU+C	82.39 ± 3.93	79.24 ± 1.47	80.01 ± 1.67
	TE+C	90.79 ± 1.12	93.83 ± 1.32	84.70 ± 1.54
ES_04	GRU+C	76.72 ± 0.14	71.35 ± 6.12	83.32 ± 2.24
	TE+C	94.65 ± 0.12	92.72 ± 1.32	90.99 ± 1.72
ES_05	GRU+C	81.74 ± 0.33	73.36 ± 1.69	78.36 ± 1.07
	TE+C	87.66 ± 0.25	80.00 ± 1.32	77.81 ± 0.99
ES_07	GRU+C	76.25 ± 3.21	73.15 ± 3.42	67.35 ± 3.62
	TE+C	79.44 ± 0.62	79.29 ± 0.72	72.53 ± 2.26

Table 3. Math operation classification accuracy after training on steps with different difficulty levels and testing on the OK ERROR, and BUG subsets. Tree embedding-based encoding outperforms GRU-based encoding.

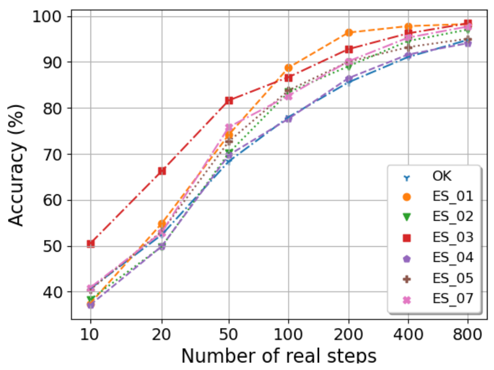


Figure 6. Math operation classification accuracy for the TE+C method when real data is limited. Using synthetically generated steps as a starting point, we already start with acceptable classification accuracy even with few real steps generated by students. The performance steadily improves after more real data becomes available.

	OK	ERROR	BUG
GRU+C	62.89 ± 3.93	64.06 ± 4.70	62.94 ± 2.24
TE+C	83.79 ± 0.14	75.49 ± 0.90	75.16 ± 0.55
CNN-C	51.12 ± 1.64	45.52 ± 0.98	59.82 ± 1.68
TE + TransE	80.17 ± 2.32	71.86 ± 3.24	72.32 ± 2.72
TE + TransR	82.22 ± 2.88	73.83 ± 3.46	74.85 ± 3.23

Table 4. Math operation classification accuracy for all methods training on $7,000$ synthetically generated steps and testing on different subsets of the CogTutor dataset. Tree embedding-based methods significantly outperform other methods, showing better ability to generalize to different data distributions.

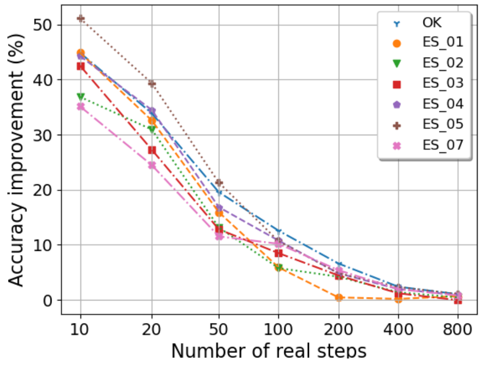


Figure 7. Math classification accuracy (difference in percentage) for TE+C, pre-training on synthetic data before fine-tuning on real data versus training only on real data. When real data is limited, pre-training on synthetic data results in significantly better performance.

$(1, 0.5, -7$, etc.) and variables (x, u, t, etc.), resulting in a mismatch between the data distributions. Tree embedding-based methods benefit from the tree-based representations of math expressions that can effectively capture structural information, making it easy for the learned embeddings of math expressions to generalize to unseen data.

5.4.4 *Generalizing from synthetic data*

Ideally, if there is a large amount of training data, i.e., steps generated by real students containing different types of math expressions and detailed labels on these steps such as the math operation(s) applied, the error(s) if a step is incorrect, and corresponding feedback, we can simply use that data to learn our math operation representations. However, in practice, the amount of real data is often limited. Figure 6 plots the performance of TE+C on all subsets of the CogTutor dataset, training on a portion of steps in the subset for training and testing on the rest. We see that the performance on math operation classification suffers considerably when we only have limited training data. Therefore, synthetically generated data can play a vital role in improving their performance under this circumstance; the strategy of fine-tuning models trained on synthetically generated data using a small amount of real data can be effective. Specifically, we start with a pre-trained math operation classification model on the 7000 synthetically generated steps and fine tune it on a small number of real steps by doing gradient descent on these steps for 10 epochs. Figure 7 plots the improvement in math operation classification accuracy for the fine-tuned model over the model that trains on only real data of various amounts on all data subsets. We see that the pre-trained models always performs better, with significant improvement when the real data is extremely limited. This result suggests that i) effectively leveraging synthetically generated data can mitigate the problem of limited real data and ii) our math operation representation learning methods are capable of generalizing across different data distributions (synthetic $\rightarrow$ real).

5.4.5 *Feedback type classification*

In this experiment, we evaluate our math operation representation learning methods on the feedback type classification task. These feedback items were automatically deployed by Cognitive Tutor for incorrect steps in the BUG subset. We pre-processed these steps and grouped the detailed feedback items according to the students' errors that each

Method	**Accuracy**
GRU+C	75.35 ± 1.41
TE + C	78.71 ± 1.74
CNN+C	67.23 ± 1.54
TE + TransE	69.15 ± 1.13
TE + TransR	73.21 ± 1.63

Table 5. Feedback type classification accuracy for all methods on the BUG subset. Tree embedding-based encoding outperforms other encoding methods while TransE and TransR frameworks do not reach similar performance levels due to shortage of training data.

feedback item addresses and narrowed it down to a total of 24 types that occur multiple times. We perform 5-fold cross validation on this subset. Table 5 shows the averages and standard deviations of feedback classification accuracy for all methods on this task across the five folds. We see that due to the limited size of the BUG subset (only $5,744$ steps) and the high number of classes (24), all method perform worse than they do on the math operation classification task. Specifically, we see that the tree embedding-based encoder in conjunction with a classifier performs best while GRU-based encoding also performs well. This result shows that although tree embeddings are superior at capturing the meaning of math expressions, their advantage over simple encoding methods such as GRU-based encoding decreases due to increased noise in the data; some math expressions submitted by students in incorrect steps are ill-posed and do not make sense. Using the TransE and TransR frameworks result in slightly worse performance than classifiers since these methods explicitly learn a representation for each math operation, which limits their performance on this task due to the shortage of training data. However, since they capture the structural difference in math expressions before and after the step, they can cancel out some of the noise in erroneous steps, resulting in acceptable performance.

5.5 Discussions

Overall, we find that the GRU-based and tree embedding-based math expression encoders in conjunction with a classifier perform almost equally well in most situations, while the CNN-based encoder performs worse. The tree embedding-based encoder has stronger generalizability across different data distributions. We believe that as the math expressions and operations get more complicated, methods that leverage the tree structure of math expressions would be more advantageous. We also observe that TransR outperforms TransE most of the time, although in some experiments using TransE and TransR to explicitly learn math operation embeddings lead to slightly worse performance than classifiers using implicit representations of math expressions. However, TransE and TransR are much more powerful and enable us to study more tasks such as clustering solution steps and identifying typical student errors and learning solution *strategies*; See Section 6 for a detailed discussion.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we developed a series of methods to learn representations of math operations by observing how math expressions change as a result of these operations in step-by-step solutions to open-ended math questions. Our methods leverage math expression encoding methods that map tree-structured math expressions into a math embedding vector space. We demonstrated the effectiveness of our methods on a dataset containing detailed student solution steps to equation solving questions in the Cognitive Tutor system on two tasks: i) classifying the math operation applied in each step and ii) classifying the feedback the system deploys for each incorrect step. Results show that our learned math operation representations are meaningful and can often effectively generalize across different data distributions such as questions with different difficulty levels.

However, the success of our methods heavily depends on the availability of diverse large-scale training data. The Cognitive Tutor dataset that we used in this work represents a heavily restricted solution process since the list of math operations a student can apply in a step is pre-defined. Therefore, additional work has to be done to extend our method to truly open-ended step-by-step solution processes that are less structured. Moreover, our methods are restricted to a single solution step only and do not consider the relationship across multiple steps, which is related to another important aspect of solving open-ended math questions: the overall solution strategy, i.e., which math operation to apply next. Furthermore, in both classification tasks, using tree embeddings to encode math expressions in conjunction with a classifier outperforms explicitly learning vectorized representations of math operations in the TransE and TransR frameworks. However, these explicit representations may enable us to perform other tasks such as Nevertheless, our work provides a series of tools to analyze the math expressions students write down in their solutions by bridging the gap between symbolic math representations with continuous representations in vector spaces, enabling the use of state-of-the-art neural network-based methods. We believe that this work can potentially open up a new line of research that studies how to automatically analyze student solutions for grading and feedback purposes.

There are many avenues of future work. First, since most real-world open-ended solutions contain a mixture of math expressions and text, there is a need to learn a joint representation of math expressions and text in a shared embedding space. Second, this joint representation will enable us to train automated feedback generation methods in an end-to-end manner, using sequence-to-sequence learning methods [41]. Third, using learned math expression representations as the states and learned math operation representations from the TransE and TransR frameworks as the state transition model, we can apply reinforcement learning and inverse reinforcement learning methods to learn solution strategies, i.e., which math operation to apply in the next step. We can also study solution strategies employed by real students [33] and diagnose their errors and design corresponding feedback mechanisms to improve their learning outcomes. These future work directions will enable us to tap into the full potential of explicit math operation representations, which is not fully demonstrated in this paper: on the CogTutor dataset, the only relevant real-world dataset we found, we could only evaluate these explicit representations on the math operation and feedback prediction tasks, where they may not outperform tree embedding-based classification-based methods.

7. REFERENCES

[1] D. M. Adams, B. M. McLaren, K. Durkin, R. E. Mayer, B. Rittle-Johnson, S. Isotani, and M. Van Velsen. Using erroneous examples to improve mathematics learning with a web-based tutoring system. *Computers in Human Behavior*, 36:401–411, 2014.

[2] Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3:1137–1155, 2003.

[3] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko. Translating embeddings for modeling multi-relational data. In *Advances in neural information processing systems*, pages 2787–2795, 2013.

[4] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proc. Conf. Empirical Methods Natural Language Process.*, pages 1724–1734, October 2014.

[5] K. Davila and R. Zanibbi. Layout and semantics: Combining representations for mathematical formula search. In *Prof. Intl. ACM SIGIR Conf. Res. Develop. Info. Retrieval*, page 1165–1168, 2017.

[6] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proc. Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, 2019.

[7] S. T. Dumais. Latent semantic analysis. *Annual review of information science and technology*, 38(1):188–230, 2004.

[8] J. A. Erickson, A. F. Botelho, S. McAteer, A. Varatharaj, and N. T. Heffernan. The automated grading of student open responses in mathematics. In *Proceedings of the Tenth International Conference on Learning Analytics & Knowledge*, pages 615–624, 2020.

[9] M. Q. Feldman, J. Y. Cho, M. Ong, S. Gulwani, Z. Popović, and E. Andersen. Automatic diagnosis of students' misconceptions in k-8 mathematics. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, pages 1–12, 2018.

[10] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning.* MIT Press, 2016.

[11] S. Grover and R. Pea. Computational thinking in k–12: A review of the state of the field. *Educational researcher*, 42(1):38–43, 2013.

[12] N. T. Heffernan and C. L. Heffernan. The assistments ecosystem: Building a platform that brings scientists and teachers together for minimally invasive research on human learning and teaching. *International Journal of Artificial Intelligence in Education*, 24(4):470–497, 2014.

[13] N. Houlsby, A. Giurgiu, S. Jastrzebski, B. Morrone, Q. De Laroussilhe, A. Gesmundo, M. Attariyan, and S. Gelly. Parameter-efficient transfer learning for NLP. In *International Conference on Machine Learning*, pages 2790–2799. PMLR, 2019.

[14] Y. Hu, Y. Zheng, H. Liu, D. Jiang, Y. Liu, and B. Ren. Accurate structured-text spotting for arithmetical exercise correction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 686–693, 2020.

[15] K. Kelly, N. Heffernan, S. D'Mello, J. Namais, and A. Strain. Adding teacher-created motivational video to an its. In *Proceedings of 26th Florida Artificial Intelligence Research Society Conference*, pages 503–508. Citeseer, 2013.

[16] A. Kembhavi, M. Seo, D. Schwenk, J. Choi, A. Farhadi, and H. Hajishirzi. Are you smarter than a sixth grader? textbook question answering for multimodal machine comprehension. In *Proceedings of the IEEE Conference on Computer Vision and Pattern recognition*, pages 4999–5007, 2017.

[17] Y. Kim. Convolutional neural networks for sentence classification. In *Proc. Conf. Empirical Methods Natural Language Process.)*, pages 1746–1751, Oct. 2014.

[18] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *Proc. Int. Conf. on Learn. Representations*, 2015.

[19] K. R. Koedinger, R. S. Baker, K. Cunningham, A. Skogsholm, B. Leber, and J. Stamper. A data repository for the edm community: The pslc datashop. *Handbook of educational data mining*, 43:43–56, 2010.

[20] K. R. Koedinger, A. Corbett, et al. *Cognitive tutors: Technology bringing learning sciences to the classroom.* na, 2006.

[21] J. Kolb, S. Farrar, and Z. A. Pardos. Generalizing expert misconception diagnoses through common wrong answer embedding. *International Educational Data Mining Society*, 2019.

[22] A. S. Lan, D. Vats, A. E. Waters, and R. G. Baraniuk. Mathematical language processing: Automatic grading and feedback for open response mathematical questions. In *Proceedings of the Second (2015) ACM Conference on Learning@ Scale*, pages 167–176, 2015.

[23] Y. Lin, Z. Liu, M. Sun, Y. Liu, and X. Zhu. Learning entity and relation embeddings for knowledge graph completion. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29, 2015.

[24] R. Liu, R. Patel, and K. R. Koedinger. Modeling common misconceptions in learning process data. In *Proceedings of the Sixth International Conference on Learning Analytics & Knowledge*, pages 369–377, 2016.

[25] B. Mansouri, S. Rohatgi, D. W. Oard, J. Wu, C. L. Giles, and R. Zanibbi. Tangent-cft: An embedding model for mathematical formulas. In *Proc. Intl. ACM SIGIR Conf. Res. Develop. Info. Retrieval*, page 11–18, 2019.

[26] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Proc. Conference on Advances in Neural Information Processing Systems*, pages 3111–3119, Dec. 2013.

[27] E. O'Rourke, E. Butler, A. D. Tolentino, and Z. Popović. Automatic generation of problems and explanations for an intelligent algebra tutor. In *International Conference on Artificial Intelligence in Education*, pages 383–395. Springer, 2019.

[28] T. Patikorn and N. T. Heffernan. Effectiveness of

crowd-sourcing on-demand assistance from teachers in online learning platforms. In *Proceedings of the Seventh ACM Conference on Learning@ Scale*, pages 115–124, 2020.

[29] L. Pfahler, J. Schill, and K. Morik. The search for equations–learning to identify similarities between mathematical expressions. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 704–718. Springer, 2019.

[30] C. Piech, J. Huang, A. Nguyen, M. Phulsuksombati, M. Sahami, and L. Guibas. Learning program embeddings to propagate feedback on student code. In *International conference on machine Learning*, pages 1093–1102. PMLR, 2015.

[31] T. W. Price, Y. Dong, and T. Barnes. Generating data-driven hints for open-ended programming. *International Educational Data Mining Society*, 2016.

[32] J. Qin, L. Lin, X. Liang, R. Zhang, and L. Lin. Semantically-aligned universal tree-structured solver for math word problems. In *Proc. Conf. Empirical Methods Natural Lang. Process.*, pages 3780–3789, Nov. 2020.

[33] A. N. Rafferty, R. A. Jansen, and T. L. Griffiths. Assessing mathematics misunderstandings via bayesian inverse planning. *Cognitive science*, 44(10):e12900, 2020.

[34] S. Ritter, J. R. Anderson, K. R. Koedinger, and A. Corbett. Cognitive tutor: Applied research in mathematics education. *Psychonomic bulletin & review*, 14(2):249–255, 2007.

[35] R. D. Roscoe, E. L. Snow, L. K. Allen, and D. S. McNamara. Automated detection of essay revising patterns: Applications for intelligent feedback in a writing tutor. *Grantee Submission*, 10(1):59–79, 2015.

[36] D. Saxton, E. Grefenstette, F. Hill, and P. Kohli. Analysing mathematical reasoning abilities of neural models, 2019.

[37] D. Selent and N. Heffernan. Reducing student hint use by creating buggy messages from machine learned incorrect processes. In *International conference on intelligent tutoring systems*, pages 674–675. Springer, 2014.

[38] V. Shiv and C. Quirk. Novel positional encodings to enable tree-based transformers. In *Proc. Intl. Conf. Neural Info. Process. Syst.*, pages 12081–12091, 2019.

[39] V. J. Shute. Focus on formative feedback. *Review of educational research*, 78(1):153–189, 2008.

[40] R. Singh, S. Gulwani, and A. Solar-Lezama. Automated feedback generation for introductory programming assignments. In *Proc. 34th ACM SIGPLAN Conf. on Programming Language Design and Implementation*, volume 48, pages 15–26, June 2013.

[41] I. Sutskever, O. Vinyals, and Q. Le. Sequence to sequence learning with neural networks. In *Proc. Advances in Neural Information Processing Systems*, pages 3104–3112, 2014.

[42] L. Van der Maaten and G. Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

[43] K. VanLehn. Bugs are not enough: Empirical studies of bugs, impasses and repairs in procedural skills. *The Journal of Mathematical Behavior*, 1982.

[44] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin. Attention is all you need. In *Proc. Intl. Conf. Neural Info. Process. Syst.*, volume 30, 2017.

[45] 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.

[46] Z. Wang, A. Lan, and R. Baraniuk. Mathematical formula representation via tree embeddings. Online: `https://people.umass.edu/~andrewlan/papers/preprint-forte.pdf`, 2021.

[47] J.-W. Wu, F. Yin, Y.-M. Zhang, X.-Y. Zhang, and C.-L. Liu. Handwritten mathematical expression recognition via paired adversarial learning. *International Journal of Computer Vision*, pages 1–16, 2020.

[48] Z. Xie and S. Sun. A goal-driven tree-structured neural model for math word problems. In *Proc. Int. Joint Conf. Artificial Intell.*, pages 5299–5305, 7 2019.

[49] W. Zhong, S. Rohatgi, J. Wu, C. Giles, and R. Zanibbi. Accelerating substructure similarity search for formula retrieval. In *Proc. European Conf. Info. Retrieval*, pages 714–727, 2020.

[50] W. Zhong and R. Zanibbi. Structural similarity search for formulas using leaf-root paths in operator subtrees. In L. Azzopardi, B. Stein, N. Fuhr, P. Mayr, C. Hauff, and D. Hiemstra, editors, *Proc. Intl. Conf. Neural Info. Process. Syst.*, pages 116–129, 2019.

[51] Y. Zou and W. Lu. Text2Math: End-to-end parsing text into math expressions. In *Proc. Conf. Empirical Methods Natural Lang. Process. and Intl. Joint Conf. Natural Lang. Process.*, pages 5327–5337, Nov. 2019.