Semantic Representations of Mathematical Expressions in a Continuous Vector Space

Neeraj Gangwar
gangwar2@illinois.edu

Nickvash Kani
kani@illinois.edu

arXiv:2211.08142v1 [cs.CL] 8 Oct 2022

Abstract

Mathematical notation makes up a large portion of STEM literature, yet, finding semantic representations for formulae remains a challenging problem. Because mathematical notation is precise and its meaning changes significantly with small character shifts, the methods that work for natural text do not necessarily work well for mathematical expressions. In this work, we describe an approach for representing mathematical expressions in a continuous vector space. We use the encoder of a sequence-to-sequence architecture, trained on visually different but mathematically equivalent expressions, to generate vector representations (embeddings). We compare this approach with an autoencoder and show that the former is better at capturing mathematical semantics. Finally, to expedite future projects, we publish a corpus of equivalent transcendental and algebraic expression pairs.

1 Introduction

Despite there being well-established search technologies for most other modes of data, there exist no *major* algorithms for processing mathematical content (Larson et al., 2013). The first step toward processing new modes of data is to create embedding methods that can transform that information block into a machine-readable format. Most mathematical equation modeling attempts have focused on establishing a homomorphism between an equation and surrounding mathematical text (Zanibbi et al., 2016; Krstovski and Blei, 2018). While this approach can help find equations used in similar contexts, it overlooks the following key points: (1) there is a large chunk of data in which equations occur without any context (consider textbooks that contain equations with minimal explanation), and (2) in scientific literature, the same equations may be used in a variety of disciplines with different contexts; encoding equations according to textual context hampers cross-disciplinary retrieval.

We argue that context alone is not sufficient for finding representations of mathematical content, and the embedding models must consider the equation itself. To this end, we present a novel embedding technique that learns to generate representations of mathematical expressions based on semantics. In our proposed approach, *we train a sequence-to-sequence model on equivalent expression pairs* and use the trained encoder to generate vector representations. Figure 1 shows an example of our approach that embeds expressions according to their mathematical semantics producing better clustering, retrieval, and analogy results. The efficacy of our approach is highlighted when compared to an autoencoder. We also compare our embedding model with two existing methods: EQNET (Allamanis et al., 2017) and EQNET-L (Liu, 2022), further proving our model's ability to capture the semantic meaning of an expression.

The contributions of our work are threefold:

1. We show that a sequence-to-sequence model can learn to generate expressions that are mathematically equivalent to a given input.
2. We use the encoder of such a model to generate continuous vector representations of mathematical expressions. These representations capture semantics and not just the structure. These embeddings produce better clustering and retrieval of similar mathematical expressions.
3. We publish a corpus of equivalent transcendental and algebraic expressions pairs which can be used to develop more complex mathematical embedding approaches.

We end this manuscript with a comprehensive study of the embedding schema detailing its potential for general information processing and retrieval tasks and noting the limitations of our approach. All datasets and source code are available on our project page.[1]

2 Related Work

While information processing for mathematical expressions is still a relatively new field, other groups have attempted to create embedding methods for mathematical content.

Starting with embedding schemes for mathematical *tokens*, Gao et al. (2017) embedded mathematical symbols using word2vec (Mikolov et al., 2013) on the Wikipedia corpus. They created a mathematical token embedding (symbol2vec) in which tokens used in similar contexts ($\sin$ & $\cos$, $=$ & $\approx$) were grouped together. Using these symbol embeddings, they extended their approach to embed entire formulae using the paragraph-to-vector approach (formula2vec). More commonly, other approaches have attempted to extract textual descriptors of equations using surrounding text and use pre-trained embeddings of those textual keywords to create an embedding of the equation in question (Kristianto et al., 2014; Schubotz et al., 2017).

Expanding token descriptors to represent equations is more complex. Schubotz et al. (2016) created equation descriptors by combining the textual keyword descriptors of the mathematical tokens included within an expression. By organizing the equation descriptors into keyword pairs, clustering algorithms grouped equations according to Wikipedia categories. In a similar approach, Kristianto et al. (2017) viewed equations as dependency relationships between mathematical tokens. Nominal NLP methods were used to extract textual descriptors for these tokens, and the equation was transformed into a dependency graph. A combination of interdependent textual descriptors allowed the authors to derive better formula descriptors that were used by indexers for retrieval tasks.

Alternatively, multiple groups have attempted to represent equations as feature sets, sequences of symbols that partially describe an equation's visual layout (Zanibbi et al., 2015; Fraser et al., 2018). Krstovski and Blei (2018) used word2vec in two different manners to find equation embeddings. In one approach, they treated an equation as a single token. In their second approach, they treated variables, symbols, and operators as tokens. The equation embedding was computed as the average of the embeddings of these individual tokens. Mansouri et al. (2019) used a modified version of the latter method that extracted features from both the symbol layout tree and operator tree representations of the expression. Ahmed et al. (2021) used a complex schema where an equation was embedded using a combination of a message-passing network (to process its graph representation) and residual neural network (to process its visual representation).

Most of these approaches depend on embedding mathematical tokens and expressions using surrounding textual information, effectively establishing a homomorphism between mathematical and textual information. While these approaches have produced crucial initial results, there are still two important limitations that need to be addressed. Firstly, an ideal embedding scheme should be able to process equations without surrounding text, such as in the case of pure math texts like the Digital Library of Mathematical Functions (DLMF) (Lozier, 2003).

Secondly, expressions that have equivalent semantic meaning may be written a multitude of ways (consider $x^{-1} = \frac{1}{x}$ or $\sin(x) = \cos(x - \frac{\pi}{2})$). The embedding method should understand that the expressions are equivalent and produce similar embeddings. Allamanis et al. (2017) and Liu (2022) have previously proposed EqNet and EqNet-L, respectively, for finding semantic representations of simple symbolic expressions. However, these approaches only focus on ensuring that the embeddings of *equivalent* expressions are grouped together. They do not consider or explore semantically similar but non-equivalent expressions.

To this end, we look at mathematical expressions in isolation without any context and propose an approach to semantically represent these expressions in a continuous vector space. Our approach considers semantic similarity in addition to mathematical equivalence.

3 Proposed Approach

We frame the problem of embedding mathematical expressions as a sequence-to-sequence learning problem and utilize the encoder-decoder framework. While word2vec-based approaches assume that proximity to a word suggests similarity, we contend that **for mathematical expressions, equivalence implies similarity**. Furthermore, if we train a sequence-to-sequence model to generate expressions that are mathematically equivalent

[1]https://github.com/mlpgroup/expemb

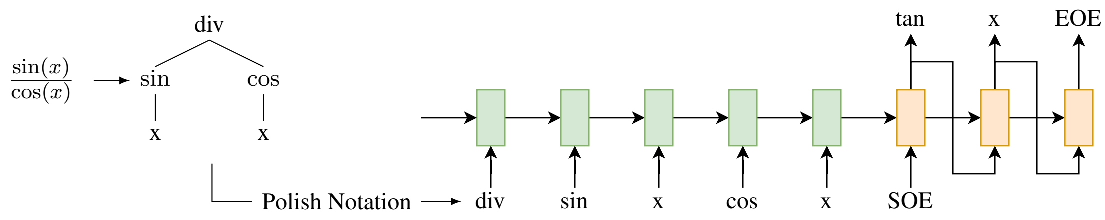


Figure 1: Example our model trained on equivalent expression pairs. The encoder and decoder components are shown in green and orange, respectively. Given an input expression $\frac{\sin(x)}{\cos(x)}$, the model learns to generate $\tan(x)$. We use the last hidden state of the encoder as the *continuous vector representation* of the input expression. SOE and EOE are the start and end tokens, and "div" is the division operation.

to the input, the encoder should learn to produce embeddings that map semantically equivalent expressions closer together. Figure 1 shows an example of this approach. To accomplish this, we need a machine learning model capable of learning equivalent expressions and a dataset of equivalent expressions to train the model (Section 4).

Model. In encoder-decoder architectures, the encoder maps an input sequence to a vector. The decoder is conditioned on this vector to generate an output sequence. In our approach, the encoder maps an expression to a vector that is referred to as the *continuous vector representation* of this expression. The decoder uses this representation to generate an output expression. We train our model in two decoder settings: (1) *equivalent expression setting* (ExpEmb-E), in which the decoder generates an expression that is mathematically equivalent to the input expression, and (2) *autoencoder setting* (ExpEmb-A), in which the decoder generates the input expression exactly.

There are several choices for modeling encoders and decoders, for example, Recurrent Neural Networks (RNN), Gated Recurrent Unit (GRU), Long Short-Term Memory (LSTM), Convolutional Networks (ConvNet), Transformer, etc. In this work, we use an architecture similar to the one proposed by Kiros et al. (2015) for learning skip-thought vectors. **We use GRU to model our encoder and decoder and use the additive attention mechanism (Bahdanau et al., 2015) in the decoder.** The final hidden state of the encoder depends on the entire input sequence and interpreted as the *continuous vector representation* or *embedding* of the input expression. An overview of our model is shown in Figure 2. Refer to Appendix A for a mathematical description of the model and dimensions of different layers in our architecture.

Data Formatting. Mathematical expressions are typically modeled as trees with nodes describing a variable or an operator (Ion et al., 1998). Since we are using a sequence-to-sequence model, we use the Polish (prefix) notation to convert a tree into a sequence of tokens (Lample and Charton, 2019). As shown in Figure 1, this transforms the expression $\frac{\sin(x)}{\cos(x)}$ into the sequence $[\text{div}, \sin, x, \cos, x]$. Tokens are encoded as one-hot vectors and passed through an embedding layer before being fed to the encoder or decoder.

Other Details. Crucial to the success of our model is the adoption of the teacher forcing algorithm (Williams and Zipser, 1989) during training and the beam search algorithm (Koehn, 2004) during validation and testing. Because not all sequences produce valid expression trees, the beam search algorithm is essential in finding the output with minimum loss. As the training of our network progresses, the incidences of output expressions that produce invalid trees become increasingly rare, suggesting that the network is capable of learning mathematical tree structure.

4 Equivalent Expressions Dataset

Training our model requires a dataset of mathematically equivalent expression *pairs*. Working off a known collection of simple mathematical expressions, we use SymPy (Meurer et al., 2017) to create the *Equivalent Expressions Dataset* consisting of $\sim$4.6M equivalent expression pairs. This dataset is available publicly.[2]

Generation. To generate valid mathematical expressions, we refer to the work of Lample and Charton (2019). We take expressions from their publicly available datasets and perform pre-processing on the input expression to remove the parts that are

[2]https://mlpgroup.github.io/expemb

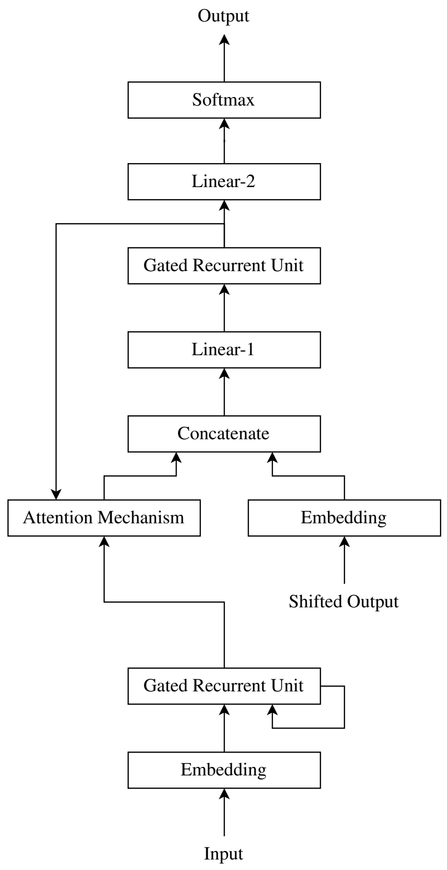


Figure 2: Our model architecture. An input expression (represented in the Polish notation) is fed into the encoder at the bottom as a sequence of tokens. The encoder processes the input and generates the embedding. The hidden states of the encoder are passed to the decoder to generate an output expression.

not necessary for our use case. This process gives us a set of valid mathematical expressions. From this group of expressions, we use SymPy to generate mathematically equivalent but visually different counterparts (see appendix B for details). For each pair of equivalent expressions $\mathbf{x}_1$ and $\mathbf{x}_2$, we add two examples $(\mathbf{x}_1, \mathbf{x}_2)$ and $(\mathbf{x}_2, \mathbf{x}_1)$ to the dataset. In our data generation process, we encounter expressions that result in NaN (Not a Number) when parsed using SymPy. We exclude examples that contain such expressions from our dataset.

Dataset. Our training dataset has 4,662,232 input-output *pairs* comprised of 2,744,809 unique expressions. Note that the number of equivalent expression pairs per unique expression is not straightforward to compute because some expressions result in more equivalent expressions than others. All expressions are univariate and consist of the following operators:

Operator Type	# Expressions
Arithmetic	1,423,972
Trigonometric	341,805
Hyperbolic	214,731
Logarithmic/Exponential	901,460

Table 1: Number of expressions containing a type of operator in the Equivalent Expressions Dataset.

- Arithmetic: $+$, $-$, $\times$, $/$, abs, pow, sqrt
- Trigonometric: $\sin$, $\cos$, $\tan$, $\cot$, $\sec$, $\csc$, $\sin^{-1}$, $\cos^{-1}$, $\tan^{-1}$
- Hyperbolic: $\sinh$, $\cosh$, $\tanh$, $\coth$, $\sinh^{-1}$, $\cosh^{-1}$, $\tanh^{-1}$
- Logarithmic/Exponential: $\ln$, $\exp$

For embedding analysis, this work only considers simple polynomial and transcendental mathematical expressions, and as such, we limit the input and output expressions to have a maximum of five operators. Table 1 shows the number of expressions containing a particular type of operator. Note that one expression can contain multiple types of operators. The sequence length of the expressions in the training dataset is 16.18 ± 4.29. Our validation and test datasets contain 2,000 and 5,000 expressions with sequence lengths of 15.03 ± 4.31 and 16.20 ± 4.13, respectively.

5 Experiments

We consider two models for our experiments:

- ExpEmb-E is the model trained on disparate but mathematically equivalent expression pairs.
- ExpEmb-A refers to an autoencoder that is trained to generate the same expression as the input.

While the focus of this work is to demonstrate the efficacy of ExpEmb-E, the autoencoder approach serves as an important contrast, demonstrating that ExpEmb-E yields representations that better describe *semantics* and are superior for clustering, retrieval, and analogy tasks. We use the model shown in Figure 2 for our experiments.

5.1 Equivalent Expression Generation

The first objective is to show that ExpEmb-E can learn to generate equivalent expressions for a given input. To evaluate if two expressions $\mathbf{x}_1$ and $\mathbf{x}_2$ are mathematically equivalent, we simplify their difference $\mathbf{x}_1 - \mathbf{x}_2$ using SymPy and compare it to 0. In this setting, if the model produces an expression that is the same as the input, we do not count it as a model success. There are instances in which SymPy takes significant time to simplify an expression and eventually fails with out-of-memory errors. To handle these cases, we put a threshold on its execution time. If the simplification operation takes more time than the threshold, we count it as a model failure. We also evaluate ExpEmb-A and verify that it learns to generate the input exactly. This serves as an important contrast to ExpEmb-E and is useful in understanding how well a sequence-to-sequence model understands the mathematical structure. During the evaluation, we use the beam search algorithm to generate output expressions. As an output can be verified programmatically, we consider all the outputs in the beam. If any of the outputs are correct, we count it as a model success.

We train ExpEmb-A with $H = 128$ and ExpEmb-E with $H = 1024$ for this experiment where H is the model dimension (see Appendix A). The accuracy of these models is shown in Table 2. First, we note that ExpEmb-A is easily able to encode and generate the input expression and achieves a near-perfect accuracy with $H = 128$ and greedy decoding (beam size = 1). Second, the ExpEmb-E results demonstrate that generating equivalent expressions is a significantly harder task. For this setting, the greedy decoding does not perform well. We observe an improvement of 35% for a beam size of 50. However, we also see a jump in the number of invalid expressions being assigned high log probabilities with this beam size. This experiment demonstrates that both ExpEmb-A and ExpEmb-E are capable of learning the mathematical structure, and ExpEmb-E can learn to generate expressions that are mathematically equivalent to the input expression. While ExpEmb-E achieves a lower accuracy than ExpEmb-A, we see in Section 5.2 that the former exhibits some interesting properties and is more useful for retrieval tasks.

5.2 Embedding Evaluation

Next, we evaluate the usefulness of the representations generated by the ExpEmb-E model. Unlike the natural language textual embedding problem (Wang et al., 2018), there do not exist standard tests to quantitatively measure the embedding performance of methods built for mathematical expressions. Hence, our analysis in this section must be more qualitative in nature. These experiments show some interesting properties of the representations generated by the ExpEmb-A and ExpEmb-E models and demonstrate the efficacy of the proposed approach.

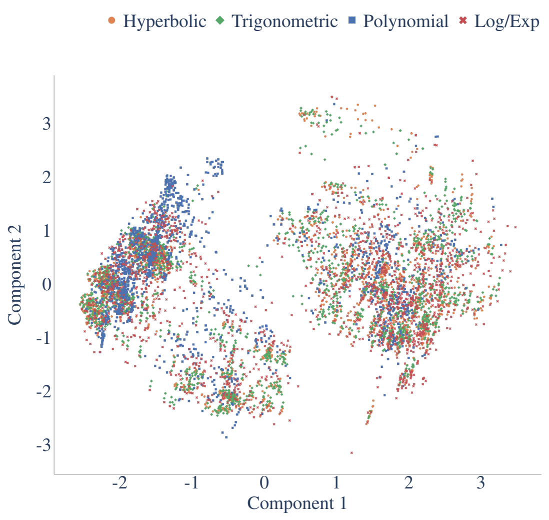
Beam Size	ExpEmb-A	ExpEmb-E
1	0.9994	0.5188
10	1.0000	0.7736
50	1.0000	0.8666

Table 2: Accuracy of ExpEmb-A and ExpEmb-E on the datasets with expressions containing a maximum of 5 operators (sequence length 16.18 ± 4.29 for the training dataset).

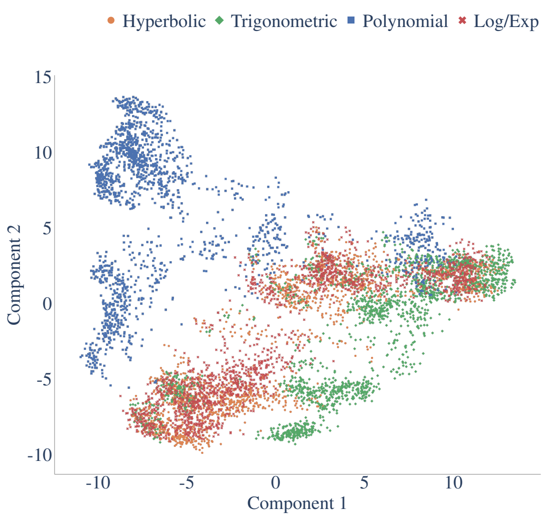
Embedding Plots. To gauge if similar expressions are clustered in the embedding vector space, we plot and compare the representations generated by the ExpEmb-E and ExpEmb-A models. For this experiment, we use 8,000 simple expressions that belong to one of the following categories: hyperbolic, trigonometric, polynomial, and logarithmic/exponential. Each expression is either polynomial or contains hyperbolic, trigonometric, or logarithmic/exponential operators. Below are a few examples of expressions belonging to each of these classes:

- Polynomial: $x^2 + 2x + 5$, $2x + 2$
- Trigonometric: $\sin(x)\tan(x)$, $\cos^5(4x)$
- Hyperbolic: $\cosh(x-4)$, $\sinh(x\cos(2))$
- Log/Exp: e^{-2x-4}, $\log(x+3)^3$

We use Principal Component Analysis (PCA) for dimensionality reduction. Figure 3 shows the plots for ExpEmb-A and ExpEmb-E. We observe from these plots that the clusters in the ExpEmb-E plot are more distinguishable compared to the ExpEmb-A plot and ExpEmb-E does a better job at grouping similar expressions together. For ExpEmb-E, there is an overlap between expressions belonging to hyperbolic and logarithmic/exponential classes. This is expected because hyperbolic operators can be written in terms of the exponential operator. Furthermore, ExpEmb-A focuses on just the structure of expressions and groups together expressions that follow the same structure. For example, representations generated by ExpEmb-A for $x^2 -$



(a) ExpEmb-A



(b) ExpEmb-E

Figure 3: Plots for the representations generated by ExpEmb-A and ExpEmb-E. PCA is used to reduce the dimensionality from 128 (ExpEmb-A) and 1024 (ExpEmb-E) to 2. The proposed approach, ExpEmb-E, groups expressions with similar operations together, whereas ExpEmb-A groups expressions mainly based on their structure. Interactive version of these plots are available on our project page.

$2x$, $\log(2x+5)$, $\tanh(3x+4)$, $\tanh^{-1}(3x+4)$, $x^2(x+5)$, $2x-5$, $\log(8-4x)$, $4x+8$, and $\cos(5x+15)$ are grouped together. On the other hand, representations generated by ExpEmb-E capture semantics in addition to the overall structure.

Distance Analysis. To understand the applicability of the embeddings for the information retrieval task, we perform distance analysis on a sample of 10,000 expressions. The similarity between two expressions is defined as the inverse of the cosine distance between their representations. We find the five closest expressions to a given query. Table 3 shows the results of this experiment. We observe that the closest expressions computed using ExpEmb-E are more similar to the query in terms of the overall structure and operators. On the other hand, ExpEmb-A focuses on a part of the query and finds other expressions that share that particular feature. For example, the first query in Table 3 consists of polynomial and trigonometric expressions. The closest expressions computed using ExpEmb-E follow the same structure, whereas the expressions computed using ExpEmb-A seem to focus on the polynomial part of the query. This behavior is also apparent in the second example. We believe that the ability of ExpEmb-E to group similar expressions together can prove useful in information retrieval problems where the aim is to find similar expressions to a given query. Refer to Appendix D for more examples.

Embedding Algebra. Word representations generated by methods like word2vec (Mikolov et al., 2013) and GloVe (Pennington et al., 2014) exhibit an interesting property that simple algebraic operations on the representations can be used to solve analogies of the form "$\mathbf{x}_1$ is to $\mathbf{y}_1$ as $\mathbf{x}_2$ is to $\mathbf{y}_2$". Following along the same lines, we perform simple algebraic operations on the representations generated by our model. For a given triplet of expressions $\mathbf{x}_1$, $\mathbf{y}_1$, and $\mathbf{y}_2$, we compute $\mathbf{z} = \mathrm{emb}(\mathbf{x}_1) - \mathrm{emb}(\mathbf{y}_1) + \mathrm{emb}(\mathbf{y}_2)$ and find the expression with representation closest to $\mathbf{z}$ in terms of cosine similarity. Here, emb represents a function that returns the vector representation for an input expression. For this experiment, we use the entire training set and ensure that all the expressions required for an analogy are present in this set. Table 4 shows the results for ExpEmb-A and ExpEmb-E. It is interesting that ExpEmb-E works for the first four examples and returns the expected expressions. This demonstrates a degree of semantic learning. Unsurprisingly, ExpEmb-A performs very poorly on this task, demonstrating the value of the ExpEmb-E approach. There are cases for which neither ExpEmb-E nor ExpEmb-A generates the expected output, but even in these cases, the ExpEmb-A results are very poor compared to ExpEmb-E. *Overall, the results of this analysis further bolster the efficacy of the* ExpEmb-E

Query	ExpEmb-A	ExpEmb-E
$21x - 3\sin(x)$	1. $11x - 2e^{x}$	1. $52x + 4\sin(x)$
	2. $2x + \operatorname{acosh}(11x)$	2. $75x + 25\sin(x)$
	3. $-x + e^{11x}$	3. $257x + 256\cos(x)$
	4. xe^{-21x}	4. $7x - 7\sin(x)$
	5. $\sin(\operatorname{asinh}(x + 21))$	5. $7x - \tan(4)$
$\frac{3x}{4} + \cos(x) + 9$	1. $2x\left(\frac{5x}{4} + \cos(x)\right)$	1. $\frac{x}{25} + \cos(x) + 2$
	2. $\left(\frac{x}{4} - 1\right)\cos(x)$	2. $3x + \cos(x) + \frac{5}{4}$
	3. $24x + \cos(x) + 3$	3. $-x + \cos(x) + 7$
	4. $\cos(x) + 3\tan\left(\frac{x}{3}\right)$	4. $6x + \cos(x) + 17$
	5. $x(2x + 7)\cos(x)$	5. $3x + \cos(x) + 12$

Table 3: Expressions closest to a given query computed using representations generated by ExpEmb-A and ExpEmb-E models. It is evident that ExpEmb-E does a better job at learning the semantics and overall structure of the expressions than ExpEmb-A.

$\mathbf{x}_1$	$\mathbf{y}_1$	$\mathbf{y}_2$	$\mathbf{x}_2$ (expected)	ExpEmb-A	ExpEmb-E
$\cos(x)$	$\sin(x)$	$\csc(x)$	$\sec(x)$	$\cosh^{-1}(x)$	$\sec(x)$
$\sin(x)$	$\cos(x)$	$\cosh(x)$	$\sinh(x)$	ex	$\sinh(x)$
$\sin(x)$	$\csc(x)$	$\sec(x)$	$\cos(x)$	$\cos(x)$	$\cos(x)$
$x^2 - 1$	$x + 1$	$x + 2$	$x^2 - 4$	$\frac{x}{\log(x)^2}$	$x^2 - 4$
$x^2 - 1$	$x + 1$	$2x + 2$	$4x^2 - 4$	$\log(x^{\frac{3}{22}})$	$x^2 - 4$
$\sin(x)$	$\sinh(x)$	$\cosh(x)$	$\cos(x)$	πx	$\cosh(\sinh(x))$
x^2	x	$\sin(x)$	$\sin^2(x)$	$10x^2$	$x^2\sin(x)$

Table 4: Examples of embedding algebra on the representations generated by ExpEmb-A and ExpEmb-E. ExpEmb-E gets the first four analogies correct and generates reasonably close outputs for the remaining compared to ExpEmb-A.

embedding approach.

5.3 Comparison with Existing Methods

As discussed in Section 2, prior works have examined if machine learning models can generate semantic representations for equivalent expressions. This historical perspective serves as an established evaluation of if ExpEmb-A and ExpEmb-E encoders learn to generate representations *that capture semantics and not just the syntactic structure*. This property is measured as the proportion of k nearest neighbors of each test expression that belong to the same class (Allamanis et al., 2017). For a test expression q belonging to a class c, the score is defined as

$$score_k(q) = \frac{|\mathbb{N}_k(q) \cap c|}{\min(k, |c|)} \tag{1}$$

where $\mathbb{N}_k(q)$ represents k nearest neighbors of q based on cosine similarity.

We use the datasets published by Allamanis et al. (2017) for this evaluation. These datasets contain equivalent expressions from the Boolean (Bool) and polynomial (Poly) domains. In these datasets, a class is defined by an expression, and all the equivalent expressions belong to the same class. The datasets are split into training, validation, and test sets and contain two test sets: (1) SeenEqClass containing classes that are present in the training set, and (2) UnseenEqClass containing classes that are not present in the training set. To transform the training set into the input-output format used by ExpEmb-E, we generate all possible pairs for the expressions belonging to the same class. To limit the size of the generated training set, we select a maximum of 100,000 random pairs from each class. For ExpEmb-A, we use all the ex-

Dataset	EqNet	EqNet-L	ExpEmb-A		ExpEmb-E	
	$score_5(\%)$	$score_5(\%)$	$score_5(\%)$	H	$score_5(\%)$	H
SimpBool8	97.4	-	27.1	1024	99.5	1024
SimpBool10	99.1	-	16.3	128	80.9	256
Bool5	65.8	73.7	25.1	256	57.1	64
Bool8	58.1	-	22.5	512	100.0	1024
Bool10	71.4	-	4.6	1024	77.5	1024
SimpBoolL5	85.0	72.1	28.5	64	79.7	128
BoolL5	75.2	-	15.5	1024	70.4	256
SimpPoly5	65.6	56.3	12.5	256	31.2	1024
SimpPoly8	98.9	98.0	31.3	64	97.2	256
SimpPoly10	99.3	-	28.7	1024	100.0	256
oneV-Poly10	81.3	80.0	55.8	1024	75.5	1024
oneV-Poly13	90.4	-	28.2	128	99.7	512
Poly5	55.3	-	22.0	1024	48.1	128
Poly8	86.2	87.1	19.8	512	76.6	512

Table 5: $score_5(\%)$ for UnseenEqClass of the SemVec datasets. The scores for EqNet and EqNet-L are from their published work (Allamanis et al., 2017; Liu, 2022). For ExpEmb-A and ExpEmb-E, the best scores and the corresponding model dimensions H are shown.

pressions present in the training set. See Appendix C for the dataset details.

Both ExpEmb-A and ExpEmb-E are trained with $H = 64, 128, 256, 512, 1024$. For evaluating our models, we use the UnseenEqClass test set. Table 5 shows the scores achieved by our approach. This table shows the best scores and corresponding model dimensions H for ExpEmb-A and ExpEmb-E. Refer to Appendix C for the scores achieved by our model with different values of H. We observe that (1) *the representations learned by* ExpEmb-E *capture semantics and not just the syntactic structure* (2) ExpEmb-A does not perform as well as ExpEmb-E. We further observe that ExpEmb-E achieves a similar performance as EqNet and EqNet-L on most of the datasets. Also, ExpEmb-E performs better than EqNet and EqNet-L on the datasets with sufficiently large training sets. Though the representation sizes (synonymous with model dimension H in our approach) are higher for ExpEmb-E than the one used in EqNet and EqNet-L, our encoder is very simple compared to both of these approaches. Our encoder consists of a GRU layer, whereas EqNet and EqNet-L use TreeNN-based encoders. Additionally, the training for EqNet and EqNet-L explicitly pushes the representations of expressions belonging to the same class closer, whereas our approach leaves it to the model to infer from the dataset.

6 Conclusion

In this work, we develop a framework to represent mathematical expressions in a continuous vector space. We show that a sequence-to-sequence model can be trained to generate expressions that are mathematically equivalent to the input, and the encoder of this model can be used to generate vector representations of mathematical expressions. We perform quantitative and qualitative experiments and demonstrate that these representations encode the semantics and not just the syntactic structure. Our experiments also show that these representations perform better at grouping similar expressions together and analogy tasks compared to the representations generated by an autoencoder. We discuss limitations and future work of our approach in Section 7.

7 Limitations

We try our proposed approach on a dataset of longer expressions. The sequence length for this dataset is 31.95 ± 30.27 for the training set and 36.40 ± 40.49 for the test set. ExpEmb-A with $H = 128$ performs well for this dataset. However, ExpEmb-E does not work well for this dataset even with $H = 2048$. Table 7 shows the accuracy of our approach for both of these settings. We observe that simply increasing the model dimension from 1024 to 2048 does not result in the desired gain.

From → To	ExpEmb-E
Bool5 → Bool10	3.7
Bool5 → Bool8	16.4
BoolL5 → Bool8	32.5
oneV-Poly10 → oneV-Poly13	60.0
Poly5 → Poly8	28.1
Poly5 → SimpPoly10	39.5
Poly8 → oneV-Poly13	52.3
Poly8 → SimpPoly10	95.6
Poly8 → SimpPoly8	97.8
SimpPoly5 → SimpPoly10	34.6
SimpPoly8 → SimpPoly10	93.4

Table 6: Compositionality test of ExpEmb-E. $score_5(\%)$ achieved on UnseenEqClass of a dataset represented by "To" on a model trained on the dataset represented by "From". For scores achieved by EqNet, refer to Figure 7 in Allamanis et al. (2017).

Beam Size	ExpEmb-A	ExpEmb-E
1	0.9969	0.4377
10	0.9993	0.6977
50	0.9995	0.7709

Table 7: Accuracy of ExpEmb-A and ExpEmb-E on the datasets with longer expressions (sequence length 31.95 ± 30.27 for the training dataset).

We believe that other results of this work should generalize for longer expressions, given a model that learns to generate equivalent expressions.

We also perform evaluation of compositionality on the SemVec datasets (Allamanis et al., 2017). For this evaluation, we train our model on a simpler dataset and evaluate on a complex dataset. For example, a model trained on Bool5 is evaluated on Bool10. We use UnseenEqClass for the evaluation. The results of this experiment are shown in Table 6. Our model does not perform as well as EqNet on this task, even for the datasets for which our model achieves a better $score_k(q)$.

Furthermore, we require downstream tasks or better evaluation metrics to evaluate the quality of the learned representations. Though we evaluate our approach on the SemVec datasets, we do not perform a similar evaluation on the Equivalent Expressions Dataset. This is because the Equivalent Expressions Dataset contains a small number of equivalent expressions for a given expression, that in turn results from limiting the number of operators in an expression to 5. With a dataset of longer expressions, this evaluation might become possible. We leave these avenues to be explored in the future.

References

Saleem Ahmed, Kenny Davila, Srirangaraj Setlur, and Venu Govindaraju. 2021. Equation Attention Relationship Network (EARN) : A Geometric Deep Metric Framework for Learning Similar Math Expression Embedding. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 6282–6289.

Miltiadis Allamanis, Pankajan Chanthirasegaran, Pushmeet Kohli, and Charles Sutton. 2017. Learning continuous semantic representations of symbolic expressions. In *International Conference on Machine Learning*, pages 80–88. PMLR.

Dzmitry Bahdanau, Kyung Hyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations, ICLR 2015*.

Dallas Fraser, Andrew Kane, and Frank Wm. Tompa. 2018. Choosing Math Features for BM25 Ranking with Tangent-L. In *Proceedings of the ACM Symposium on Document Engineering 2018*, 17, pages 1–10. Association for Computing Machinery, New York, NY, USA.

Liangcai Gao, Zhuoren Jiang, Yue Yin, Ke Yuan, Zuoyu Yan, and Zhi Tang. 2017. Preliminary Exploration of Formula Embedding for Mathematical Information Retrieval: Can mathematical formulae be embedded like a natural language? *arXiv:1707.05154 [cs]*.

Patrick Ion, Robert Miner, Ron Ausbrooks, et al. 1998. Mathematical markup language (mathml) version 3.0.

Ryan Kiros, Yukun Zhu, Russ R Salakhutdinov, Richard Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Skip-thought vectors. *Advances in neural information processing systems*, 28.

Philipp Koehn. 2004. Pharaoh: a beam search decoder for phrase-based statistical machine translation models. In *Conference of the Association for Machine Translation in the Americas*, pages 115–124. Springer.

Giovanni Yoko Kristianto, Akiko Aizawa, et al. 2014. Extracting textual descriptions of mathematical expressions in scientific papers. *D-Lib Magazine*, 20(11):9.

Giovanni Yoko Kristianto, Goran Topić, and Akiko Aizawa. 2017. Utilizing dependency relationships between math expressions in math IR. *Information Retrieval Journal*, 20(2):132–167.

Kriste Krstovski and David M. Blei. 2018. Equation Embeddings. *arXiv:1803.09123 [cs, stat]*.

Guillaume Lample and François Charton. 2019. Deep learning for symbolic mathematics. In *International Conference on Learning Representations*.

Ray R Larson, Chloe Reynolds, and Fredric C Gey. 2013. The abject failure of keyword ir for mathematics search: Berkeley at ntcir-10 math. In *NTCIR*.

Jiaxin Liu. 2022. Eqnet-l: A new representation learning method for mathematical expression. In *2022 International Conference on Big Data, Information and Computer Network (BDICN)*, pages 547–551. IEEE.

Ilya Loshchilov and Frank Hutter. 2018. Decoupled weight decay regularization. In *International Conference on Learning Representations*.

Daniel W. Lozier. 2003. NIST Digital Library of Mathematical Functions. *Annals of Mathematics and Artificial Intelligence*, 38(1):105–119.

Behrooz Mansouri, Shaurya Rohatgi, Douglas W. Oard, Jian Wu, C. Lee Giles, and Richard Zanibbi. 2019. Tangent-CFT: An Embedding Model for Mathematical Formulas. In *Proceedings of the 2019 ACM SIGIR International Conference on Theory of Information Retrieval*, ICTIR '19, pages 11–18, Santa Clara, CA, USA. Association for Computing Machinery.

Aaron Meurer, Christopher P. Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B. Kirpichev, Matthew Rocklin, Amit Kumar, Sergiu Ivanov, Jason K. Moore, Sartaj Singh, Thilina Rathnayake, Sean Vig, Brian E. Granger, Richard P. Muller, Francesco Bonazzi, Harsh Gupta, Shivam Vats, Fredrik Johansson, Fabian Pedregosa, Matthew J. Curry, Andy R. Terrel, Štěpán Roučka, Ashutosh Saboo, Isuru Fernando, Sumith Kulal, Robert Cimrman, and Anthony Scopatz. 2017. Sympy: symbolic computing in python. *PeerJ Computer Science*, 3:e103.

Tomas Mikolov, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. In *ICLR*.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

Moritz Schubotz, Alexey Grigorev, Marcus Leich, Howard S. Cohl, Norman Meuschke, Bela Gipp, Abdou S. Youssef, and Volker Markl. 2016. Semantification of Identifiers in Mathematics for Better Math Information Retrieval. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '16, pages 135–144, New York, NY, USA. Association for Computing Machinery.

Moritz Schubotz, Leonard Krämer, Norman Meuschke, Felix Hamborg, and Bela Gipp. 2017. Evaluating and Improving the Extraction of Mathematical Identifier Definitions. In *Experimental IR Meets Multilinguality, Multimodality, and Interaction*, Lecture Notes in Computer Science, pages 82–94, Cham. Springer International Publishing.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, Brussels, Belgium. Association for Computational Linguistics.

Ronald J Williams and David Zipser. 1989. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1(2):270–280.

Richard Zanibbi, Akiko Aizawa, and Michael Kohlhase. 2016. NTCIR-12 MathIR Task Overview. page 10.

Richard Zanibbi, Kenny Davila, Andrew Kane, and Frank Tompa. 2015. The Tangent Search Engine: Improved Similarity Metrics and Scalability for Math Formula Search. *arXiv:1507.06235 [cs]*.

A Model Details

Model. Let $\mathbf{x} = (x_1, x_2 \dots x_N)$ and $\mathbf{y} = (y_1, y_2 \dots y_M)$ represent an input and output expression pair. Each x_i is first mapped to a static vector $\mathbf{e}_i$. This sequence of vectors is passed to a GRU which produces a hidden state $\mathbf{h}_i$ for each time step $1 \leq i \leq N$ according to the following equations:

$$\begin{aligned}
\mathbf{r}_i &= \sigma(\mathbf{W}_r \mathbf{e}_i + \mathbf{U}_r \mathbf{h}_{i-1}) \\
\mathbf{z}_i &= \sigma(\mathbf{W}_z \mathbf{e}_i + \mathbf{U}_z \mathbf{h}_{i-1}) \\
\tilde{\mathbf{h}}_i &= \tanh(\mathbf{W}\mathbf{e}_i + \mathbf{U}(\mathbf{r_t} \odot \mathbf{h_{i-1}})) \\
\mathbf{h}_i &= (1 - \mathbf{z}_i) \odot \mathbf{h}_{i-1} + \mathbf{z}_i \odot \tilde{\mathbf{h}}_i
\end{aligned} \tag{2}$$

where $\mathbf{r}_i$, $\mathbf{z}_i$, and $\tilde{\mathbf{h}}_i$ are the reset gate, update gate, and proposed state at time step i, respectively. σ and $\odot$ denote the sigmoid function and element-wise product, respectively. The final hidden state, $\mathbf{h}_N$, depends on the entire input sequence and interpreted as the *continuous vector representation* of the input expression. The decoder generates an output expression conditioned on the encoder hidden states $\mathbf{h}_i$ for $1 \leq i \leq N$. It first computes a context vector $\mathbf{c}_t$ at time step t according to the following equations:

$$\begin{aligned}
a_{i,t} &= \mathbf{v}_a^T \tanh(\mathbf{W}_a \mathbf{s}_{t-1} + \mathbf{U}_a \mathbf{h}_i) \\
\mathbf{c}_t &= \sum_{i=1}^{N} \frac{\exp(a_{i,t})}{\sum_{j=1}^{N} \exp(a_{j,t})} \mathbf{h}_i
\end{aligned} \tag{3}$$

where $\mathbf{s}_t$ represents the decoder hidden state at time step t. The context vector $\mathbf{c}_t$ is combined with the static vector representation of the output token predicted at the previous time step as:

$$\mathbf{d}_t = \mathbf{W}_c[\mathbf{c}_t, \mathbf{o}_{t-1}] \tag{4}$$

where $\mathbf{o}_t$ represents the static vector representation of y_t, the output token at time step t. $\mathbf{d}_t$ follows (2) to generate the decoder hidden state $\mathbf{s}_t$. The probability of the output token y_t is defined as:

$$P(y_t \mid y_1 \dots y_{t-1}, \mathbf{x}) = \text{softmax}(\mathbf{W}_o \mathbf{s}_t) \tag{5}$$

Loss. For each time step, the decoder produces a probability distribution of y_t. The probability of the output sequence $\mathbf{y}$ is defined as:

$$P(\mathbf{y} \mid \mathbf{x}) = \prod_{t=1}^{M} P(y_t \mid y_1 \dots y_{t-1}, \mathbf{x}) \tag{6}$$

Dataset	ExpEmb-A	ExpEmb-E
SimpBool8	21,604	4,440,450
SimpBool10	13,081	1,448,804
Bool5	712	17,934
Bool8	146,488	16,143,072
Bool10	25,560	3,041,640
SimpBoolL5	6,009	66,876
BoolL5	23,219	552,642
SimpPoly5	156	882
SimpPoly8	1,934	113,660
SimpPoly10	31,143	6,731,858
oneV-Poly10	767	25,590
oneV-Poly13	60,128	9,958,406
Poly5	352	1,350
Poly8	6,785	257,190

Table 8: Number of training examples in the transformed training sets of the SemVec datasets.

We maximize the log-likelihood of the output sequence. For an example pair $(\mathbf{x}, \mathbf{y})$, the loss is defined as:

$$l(\mathbf{x}, \mathbf{y}) = -\sum_{t=1}^{M} \log P(y_t \mid y_1 \dots y_{t-1}, \mathbf{x}) \tag{7}$$

Architecture. Table 10 shows the dimensions of different layers in our architecture shown in Figure 2.

B Equivalent Expression Generation

We use SymPy to generate mathematically equivalent expressions for a given expression. We apply the operations shown in Table 9 to a given expression to get mathematically equivalent expressions. The examples shown in the table are intended to give an idea about each operation. The actual behavior of these operations is much more complex. Refer to the SymPy documentation for more details.[3] We use the `rewrite` function for expressions containing trigonometric or hyperbolic operations. Specifically, we use this function to rewrite an expression containing trigonometric (or hyperbolic) operators in terms of other trigonometric (or hyperbolic) operators.

C SemVec Datasets

Table 8 shows the number of examples in our training set of the SemVec datasets. We use the validation and test sets in their original form. To generate

[3] https://docs.sympy.org/latest/tutorial/simplification.html

Function	Input(s)	Output
`simplify`	$\sin(x)^2 + \cos(x)^2$	1
`expand`	$(x+1)^2$	$x^2 + 2x + 1$
`factor`	$x^2 + 5x + 6$	$(x+2)(x+3)$
`cancel`	$(x^3 + 2x)/x$	$x^2 + 2$
`trigsimp`	$\sin(x)\cot(x)$	$\cos(x)$
`expand_log`	$\ln(x^2)$	$2\ln(x)$
`logcombine`	$\ln(x) + \ln(2)$	$\ln(2x)$
`rewrite`	$\sin(x)$, cos	$\cos(x - \pi/2)$
	$\sinh(x)$, tanh	$2\tanh\left(\frac{x}{2}\right)(1 - \tanh^2\left(\frac{x}{2}\right))^{-1}$

Table 9: List of SymPy functions with examples that are used to generate equivalent expressions.

	Layer	Input	Output
Encoder	Embedding	$B \times S$	$B \times S \times H$
	GRU	$B \times S \times H, B \times H$	$B \times S \times H$
Decoder	Embedding	$B \times T$	$B \times T \times H$
	Attention	$B \times S \times H, B \times T \times H$	$B \times T \times H$
	Linear-1	$B \times T \times 2H$	$B \times T \times H$
	GRU	$B \times T \times H, B \times H$	$B \times T \times H$
	Linear-2	$B \times T \times H$	$B \times T \times V$
	Softmax	$B \times T \times V$	$B \times T \times V$

Table 10: Input and output dimensions of the layers in our model architecture (B: batch size, S: input sequence length, T: output sequence length, V: vocabulary size, H: model dimension).

$score_k(q)$ for our model, we use the source code provided by Allamanis et al. (2017).

Table 11 shows the results of our approach on the SEMVEC datasets. It can be seen that EXPEMB-A performs much worse than EXPEMB-E for all values of H. This is not surprising as the autoencoder training looks at each expression in isolation and does not utilize the fact that each expression belongs to a particular class. Hence, the representations learned by this model do not capture the semantics of mathematical expressions and only capture the structure.

D Distance Analysis

Table 12 shows more examples of the distance analysis as discussed in Section 5.2.

E Training Details

For training our models, we consider expressions with a maximum of 512 tokens. We use the AdamW optimizer (Loshchilov and Hutter, 2018) for training our models. For the Equivalent Expressions Dataset, a learning rate of 10^{-4} and a batch size of 64 are used. The same learning rate and batch size are used for the majority of the SEMVEC datasets, barring the ones mentioned in Table 13. For the dataset with longer equivalent expressions (discussed in Section 7), we use a batch size of 32 with a learning rate of 10^{-4}.

We use a single Quadro RTX 6000 GPU for training EXPEMB-E on the dataset with longer expressions and a single V100 GPU for all other models. For the Equivalent Expressions Dataset, EXPEMB-A ($H = 128$) takes ~1 hour and EXPEMB-E ($H = 1024$) takes ~2 hours to train for an epoch. We train both of these models for 20 epochs. For the dataset with longer expressions, the training times for an epoch are ~4 hours and ~20 hours for EXPEMB-A ($H = 128$) and EXPEMB-E ($H = 2048$) respectively. Table 14 shows the approximate training time per epoch (in seconds) for the SEMVEC datasets.

F Performance of Validation Datasets

For the Equivalent Expressions Dataset, the validation is performed using the model accuracy

Dataset	ExpEmb-A					ExpEmb-E				
	$H = 64$	128	256	512	1024	64	128	256	512	1024
SimpBool8	22.7	23.4	20.7	26.6	27.1	93.2	95.5	97.2	98.8	99.5
SimpBool10	10.0	16.3	10.0	9.3	11.7	59.4	62.1	80.9	75.7	78.2
Bool5	16.3	9.1	25.1	21.3	21.3	57.1	49.5	51.9	52.3	56.9
Bool8	11.8	10.6	15.0	22.5	19.6	98.4	99.9	100.0	100.0	100.0
Bool10	3.0	4.0	3.5	4.3	4.6	29.5	39.1	47.8	71.5	77.5
SimpBoolL5	28.5	16.8	16.9	23.3	22.4	46.1	79.7	63.2	76.3	68.1
BoolL5	10.9	15.2	3.8	15.1	15.5	46.4	46.8	70.4	52.0	50.7
SimpPoly5	8.3	4.2	12.5	4.2	4.2	14.6	15.6	27.1	25.0	31.2
SimpPoly8	31.3	28.3	26.7	28.6	29.8	82.7	95.4	97.2	96.7	91.8
SimpPoly10	24.6	25.5	25.1	26.0	28.7	99.8	99.9	100.0	99.9	100.0
oneV-Poly10	43.8	36.6	44.3	44.8	55.8	48.5	58.2	70.9	74.6	75.5
oneV-Poly13	20.9	28.2	25.7	26.2	26.6	94.0	99.1	99.5	99.7	99.7
Poly5	7.8	18.8	17.9	20.5	22.0	28.9	48.1	41.9	36.8	30.6
Poly8	8.7	18.3	18.7	19.8	18.8	41.6	64.5	69.3	76.6	76.3

Table 11: $score_5(\%)$ achieved by our model on UnseenEqClass of the SemVec datasets.

(as described in Section 5.1) with a beam size of 1. Table 15 shows the accuracy of ExpEmb-A and ExpEmb-E on the Equivalent Expressions Dataset and the dataset with longer expressions. Table 16 shows the scores, $score_k(q)$, achieved by our model on the validation sets of the SemVec datasets.

Query	ExpEmb-A	ExpEmb-E
$x^2 - 8\sin(x)$	1. $e(-x + \sin(x))$	1. $20x^2 + 20\sin(x)$
	2. $(4x + 36)\cos(x)$	2. $-2x^2 + 3\sin(x)$
	3. $x + \sqrt{7}\cos(x)$	3. $9x + 12\sin(x)$
	4. $-6x + 6\sin(x)$	4. $x^2 + \frac{\sin(x)}{20}$
	5. $-8x + 9\sin(x)$	5. $80x^2 + 160\sin(x)$
$\cos(x+1) + \frac{1}{2}$	1. $32\tan(x+1)$	1. $\frac{\sin(x+1)}{2} + \frac{3}{2}$
	2. $6\cos(x+1)$	2. $4x + \cos(x+1) + 3$
	3. $\tan(x+1) + 125$	3. $6\cos(x+1)$
	4. $\sinh(x+1) + 6$	4. $\frac{4x\cos(x+1)}{5}$
	5. $\sqrt{x+1} - 3$	5. $\cos(x+4) + \operatorname{acos}(5)$
$x(x\cos(5) + \operatorname{asinh}(x))$	1. $x + (x + \operatorname{asinh}(5))\operatorname{asinh}(x)$	1. $x(-x\cos(x) + x + \operatorname{acos}(x))$
	2. $x\left(x + e^{\operatorname{asinh}(x)+5}\right)$	2. $x(x + \cos(2)) + \operatorname{asin}(x)$
	3. $\sqrt{x + \tan(5)} + \operatorname{asinh}(x)$	3. $x^2\cos(x) + x + \operatorname{asin}(x) + 2$
	4. $x\sin(5) + x + \log(x)$	4. $x\sin(1) + 3x + \operatorname{acosh}(x)$
	5. $\left(\sqrt{2}x + 3\right)\sin(x)$	5. $x(x\operatorname{asinh}(3) + \cos(x))$
$2x^2(x + \operatorname{acos}(2x) + 5)$	1. $4x^2(\sqrt{x} + 2x + 2)$	1. $2x(6x + \operatorname{atanh}(2x) + 3)$
	2. $2x - \sin(\operatorname{asinh}(x)) + 5$	2. $(x+3)(x + \operatorname{asin}(2x) + 1)$
	3. $2x + \cos(x) + \tan(2x) + 3$	3. $x^2(\operatorname{asinh}(2x) + 2)$
	4. $x\operatorname{asin}(2x) + 2x + 3$	4. $x(2x + \operatorname{atanh}(2x) - 1)$
	5. $\frac{\sqrt{x} + 2x - 5}{x^4}$	5. $x(x + \sinh(2x) + 4) + x$

Table 12: Expressions closest to a given query computed using representations generated by ExpEmb-A and ExpEmb-E.

Dataset	Decoder Setting	H	Learning Rate	Batch Size
Bool5	ExpEmb-A	All	10^{-4}	8
Bool5	ExpEmb-E	64	2×10^{-4}	64
SimpPoly5	ExpEmb-A	All	5×10^{-5}	8
SimpPoly5	ExpEmb-E	All	5×10^{-5}	16
SimpPoly8	ExpEmb-A	All	10^{-4}	16
oneV-Poly10	ExpEmb-E	64	2×10^{-4}	64
oneV-Poly10	ExpEmb-A	All except 64	5×10^{-5}	8
oneV-Poly10	ExpEmb-A	64	10^{-4}	8
oneV-Poly13	ExpEmb-A	All	5×10^{-5}	32
Poly5	ExpEmb-E	All	10^{-4}	32
Poly5	ExpEmb-A	All	5×10^{-5}	8

Table 13: Datasets with hyperparameters for which the learning rate and batch sizes are different than 10^{-4} and 64 respectively.

Dataset	ExpEmb-A					ExpEmb-E				
	$H = 64$	128	256	512	1024	64	128	256	512	1024
SimpBool8	17	19	18	19	21	3405	1921	2164	3313	2245
SimpBool10	12	12	14	12	13	1141	714	823	1391	850
Bool5	3	3	3	3	3	11	6	7	12	7
Bool8	111	127	129	135	119	11088	9706	7087	9737	12618
Bool10	22	26	25	24	26	2574	1504	1706	2719	1782
SimpBoolL5	4	4	4	4	4	37	23	23	38	27
BoolL5	15	15	15	15	15	298	186	187	354	214
SimpPoly5	1	1	1	1	1	2	2	2	2	2
SimpPoly8	5	5	6	5	6	79	45	47	81	53
SimpPoly10	27	26	30	27	27	7932	3089	3140	3153	3634
oneV-Poly10	4	4	4	5	5	22	18	13	23	14
oneV-Poly13	112	112	118	117	118	5958	5869	6041	6177	6994
Poly5	2	2	2	2	2	2	2	2	2	2
Poly8	5	5	6	6	6	183	102	103	187	120

Table 14: Approximate training time in seconds per epoch for the SemVec datasets.

Beam Size	Equivalent Expressions Dataset		Dataset with Longer Expressions	
	ExpEmb-A	ExpEmb-E	ExpEmb-A	ExpEmb-E
1	1.0000	0.6015	0.9994	0.4341

Table 15: Accuracy of ExpEmb-A and ExpEmb-E on the validation datasets of the Equivalent Expressions Dataset and the dataset with longer expressions.

Dataset	ExpEmb-A					ExpEmb-E				
	$H = 64$	128	256	512	1024	64	128	256	512	1024
SimpBool8	23.8	22.5	21.4	26.0	24.1	95.4	96.9	98.0	99.4	99.6
SimpBool10	6.4	8.8	5.5	6.6	6.8	54.0	56.2	80.4	67.1	71.5
Bool5	17.9	15.6	19.2	20.8	21.5	57.4	45.7	52.8	52.4	53.2
Bool8	11.3	11.8	13.2	16.3	15.0	98.1	99.9	100.0	100.0	100.0
Bool10	1.8	2.3	2.0	1.8	2.1	26.0	34.4	37.9	67.9	74.6
SimpBoolL5	23.4	22.5	24.6	25.2	26.0	51.7	75.3	73.1	74.3	75.3
BoolL5	11.7	15.2	11.0	14.3	14.6	66.0	69.7	73.7	67.4	64.3
SimpPoly5	25.9	25.9	29.6	18.5	22.2	33.3	40.7	37.0	40.7	33.3
SimpPoly8	15.4	15.2	14.3	15.2	12.6	50.0	74.5	91.6	80.6	78.0
SimpPoly10	11.7	12.7	11.0	10.5	11.0	99.3	99.3	99.4	99.4	99.4
oneV-Poly10	24.5	29.7	28.9	24.6	26.8	27.3	32.6	53.7	58.9	61.8
oneV-Poly13	19.8	22.1	20.2	20.5	20.0	91.8	98.2	98.7	98.7	98.7
Poly5	27.8	22.2	16.7	16.7	16.7	44.4	55.6	50.0	41.7	33.3
Poly8	10.3	14.6	15.0	15.5	12.1	61.6	82.4	81.5	85.1	84.3

Table 16: $score_5(\%)$ achieved by our model on the validation sets of the SemVec datasets.