Semantic Representations of Mathematical Expressions in a Continuous Vector Space

Anonymous authors
Paper under double-blind review

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 (or embeddings). We compare this approach with an autoencoder and show that the former is better at capturing mathematical semantics. Finally, to expedite future research, 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 equation embedding methods have focused on establishing a homomorphism between an equation and its surrounding mathematical text (Zanibbi et al., 2016; Krstovski & 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 math textbooks that contain equations with minimal explanation), and (2) in scientific literature, an equation may be used in a variety of disciplines with different contexts, and encoding equations based on textual context hampers cross-disciplinary retrieval.

We argue that context *alone* is not sufficient for finding representations of mathematical content, and embedding methods must understand equations in addition to the surrounding context. To this end, we consider mathematical expressions without context and present a novel embedding method to generate semantically-rich representations of these formulae. In our proposed approach, *we train a sequence-to-sequence model on equivalent expression pairs* and use the trained encoder to generate vector representations or embeddings. Figure 1 shows an example of our approach that embeds expressions according to their 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 method with two prior works proposed for an analogous problem of clustering equivalent expressions: EqNet (Allamanis et al., 2017) and EqNet-L (Liu, 2022), further proving our model's ability to capture the semantics.

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 the 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 visual structure. They are better at clustering and retrieving similar mathematical expressions.

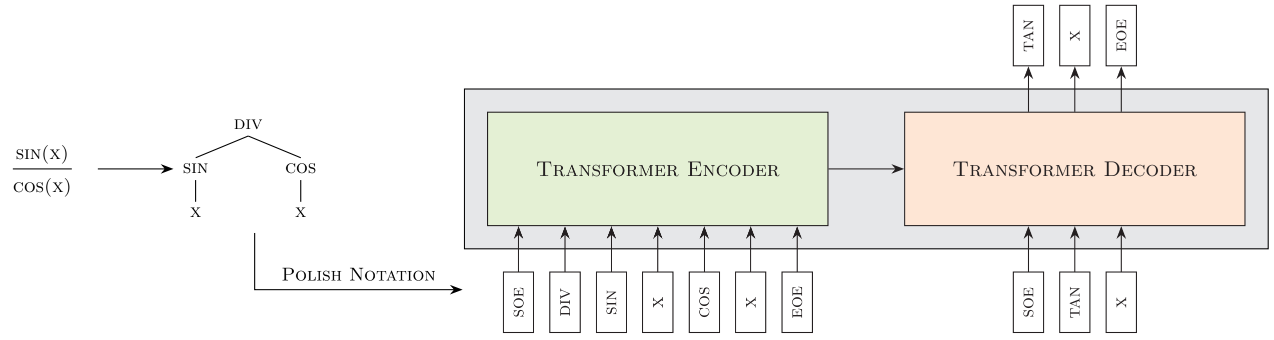


Figure 1: An overview of our approach. The model is trained to generate an expression that is mathematically equivalent to the input. For example, given an input expression $\frac{\sin(x)}{\cos(x)}$, the model learns to generate $\tan(x)$. We use max pooling on the hidden states of the last encoder layer corresponding to the input tokens and use the result as the *continuous vector representation* of the input expression. Here, SOE, EOE, and DIV represent the start-token, the end-token, and the division-operation, respectively.

3. We publish a corpus of equivalent transcendental and algebraic expressions pairs that can be used to develop more complex mathematical embedding approaches.

We end this manuscript with a comprehensive study of our proposed approach, detailing its potential for general information processing and retrieval tasks and noting its limitations. The 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, = & ≈) 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 natural language processing (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 & 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 and the equation as a sequence of 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.

[1] Anonymous link.

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 a residual neural network (to process its visual representation). Peng et al. (2021) trained BERT (Devlin et al., 2019) on mathematical datasets to create a pre-trained model for mathematical formula understanding.

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 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 may be written in a multitude of mathematically equivalent ways (consider $x^{-1} = \frac{1}{x}$ or $\sin(x) = \cos(x - \frac{\pi}{2})$). Embedding methods should recognize that such expressions are mathematically equivalent and should 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 (Seq2Seq) learning problem and utilize the encoder-decoder framework. While natural language embedding approaches, like word2vec, assume that proximity in the text suggests similarity, we contend that for mathematical expressions, mathematical equivalence implies similarity. Furthermore, we hypothesize that if we train a sequence-to-sequence model to generate expressions that are mathematically equivalent to the input, the encoder would learn to generate embeddings that map semantically equivalent expressions together in the embedding vector space. Figure 1 shows an example of our approach. To accomplish this, we need a machine learning model capable of generating equivalent expressions and a dataset of equivalent expression pairs (Section 4).

Model. In encoder-decoder architectures, the encoder encodes the input sequence, and the decoder is conditioned on the encoded vector to generate an output sequence. In our approach, the encoder maps the input expression to a vector that is referred to as the *continuous vector representation* of the expression. The decoder uses this representation to generate an output expression.

There are several choices for modeling encoders and decoders. We use the Transformer architecture (Vaswani et al., 2017) in this work. It has been successfully used in various NLP applications, like machine translation, language modeling, and summarization to name a few, and has also been shown to work with mathematical expressions (Lample & Charton, 2019).

Embedding Vector. Every encoder layer in the Transformer architecture generates hidden states corresponding to each token in the input sequence. We use max pooling on the hidden states of the last encoder layer to generate the continuous vector representation or the embedding vector of the input expression. The special tokens, like state-of-expression and end-of-expression tokens, are not considered for max pooling. Furthermore, we considered the following other candidates to generate the embedding vectors but chose max pooling based on its performance in our initial experiments - (1) average pooling of the hidden states of the last encoder layer and (2) the hidden states of the last encoder layer corresponding to the first and the last tokens.

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. For example, $\frac{\sin(x)}{\cos(x)}$ is converted to the sequence $[\text{div}, \sin, x, \cos, x]$ (Figure 1). The primary reason for using the Polish notation is that we refer to the datasets created by Lample & Charton (2019) which are in Polish notation (details in Section 4). Furthermore, their work showed that the Transformer model is capable of generating valid prefix expressions without requiring constraints during decoding. In our experiments, we also found that invalid output prefix expressions become increasingly rare as the training progresses. Hence, we use the Polish notation for simplicity and leave the exploration of tree-based representations of mathematical expressions for future work. The tokens are encoded as one-hot vectors and passed through an embedding layer and positional encoding before being fed to the encoder or decoder.

Decoding. We use the beam search (Koehn, 2004) to generate expressions at inference. We do not use any other constraint during decoding. The beam search algorithm is essential in finding an output with as high a probability as computationally possible.

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 ∼4.6M equivalent expression pairs. This dataset is available publicly on our project page.

Generation. To generate valid mathematical expressions, we refer to the work of Lample & Charton (2019). We take expressions from their publicly available datasets and perform pre-processing to remove the parts that are not necessary for our use case. This process gives us a set of valid mathematical expressions. For these expressions, we use SymPy to generate mathematically equivalent but visually different counterparts (see Appendix A 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.

Dataset. Our training set has 4,662,300 input-output *pairs* comprised of 2,744,824 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:

- 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

The variable in the expressions is considered to be positive and real. 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. Table 2 shows the number of operators in the training, validation, and test sets and corresponding sequence lengths.

Our validation and test sets contain a single expression per example instead of a pair. This is because there are multiple possible correct equivalent expressions for a given input expression. At inference, we use SymPy to determine if an output expression is equivalent to the input, instead of fixing one or multiple outputs for an input. The validation and test sets contain 2,000 and 5,000 expressions, respectively.

Note that an expression will only be present in one set but an equivalent expression may be present in another set. For example, if $\sin(x)$ is present in the training set as either input or output, it will not be

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

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

Table 2: The number of operators and sequence lengths of training, validation, and test sets of the Equivalent Expressions Dataset.

Set	# Operators	Sequence Length
Training	5.68 ± 1.32	16.19 ± 6.28
Validation	5.60 ± 1.29	15.03 ± 4.31
Test	5.98 ± 1.22	16.20 ± 4.13

present in the validation and test sets. But $\cos(x - \frac{\pi}{2})$ may be present in either validation or test set, but not both.

5 Experiments

We consider two decoder settings for our experiments:

- Equivalent expression setting, or ExpEmb-E, in which the model is trained on disparate but mathematically equivalent expression pairs.
- Autoencoder setting, or ExpEmb-A, in which the model is trained as an autoencoder to output the same expression as the input.

While the focus of this work is to demonstrate the efficacy of ExpEmb-E, ExpEmb-A serves as an important contrast, demonstrating that ExpEmb-E yields representations that better describe *semantics* and are superior for clustering, retrieval, and analogy tasks.

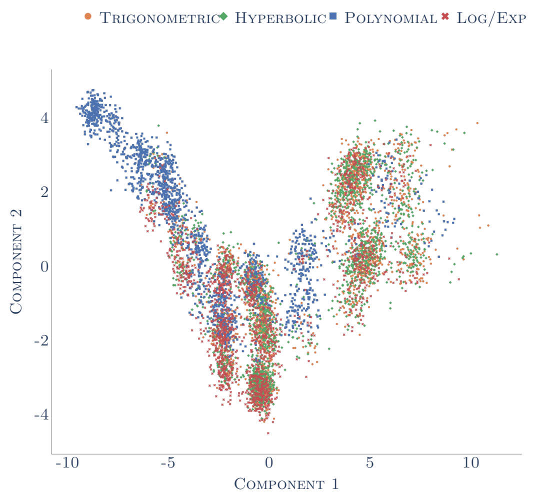
5.1 Training Details

We use the Transformer architecture with 6 encoder layers, 6 decoder layers, 8 attention heads, and the ReLU activation. The model dimension, indicated by D_{model}, is 512 for experiments in Sections 5.2 and 5.3 and is 64 for experiments in Section 5.4. The layer normalization is performed before other attention and feedforward operations. We use the Adam optimizer (Kingma & Ba, 2015) with a learning rate of 10^{-4} and do not consider expressions with more than 256 tokens. Refer to Appendix B for more details.

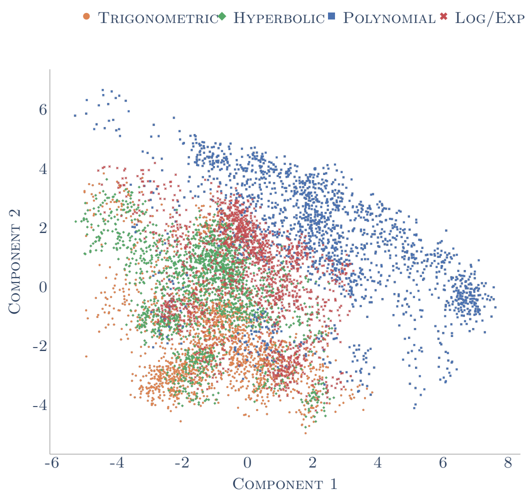
For experiments in Sections 5.2 and 5.3, we use the Equivalent Expressions Dataset (Section 4). For ExpEmb-A, we use unique expressions present in the training set. The experiments in Section 5.4 use the datasets created by Allamanis et al. (2017) (details in the same section).

5.2 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



(a) ExpEmb-A



(b) ExpEmb-E

Figure 2: Plots for the embedding vectors generated by ExpEmb-A (autoencoder) and ExpEmb-E (proposed equivalent-expression method). PCA is used to reduce the dimensionality from 512 to 2. ExpEmb-E groups expressions with operators of the same type together, indicating its ability to understand semantics, whereas ExpEmb-A groups expressions mainly based on their visual structure. The interactive versions of these plots are available on our project page.

Table 3: Accuracy of ExpEmb-A and ExpEmb-E on the Equivalent Expressions Dataset. ExpEmb-A has an easier task of generating the input at the decoder, while ExpEmb-E must generate a mathematically equivalent expression.

Beam Size	ExpEmb-A	ExpEmb-E
1	0.9996	0.7206
10	0.9998	0.8118
50	0.9998	0.8386

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. At inference, 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.

The accuracy of these models is shown in Table 3. First, we note that ExpEmb-A is easily able to encode and generate the input expression and achieves a near-perfect accuracy with greedy decoding (beam size = 1). Second, the ExpEmb-E results demonstrate that generating mathematically equivalent expressions is a significantly harder task. For this setting, greedy decoding does not perform well. We observe an improvement of 11% with a beam size of 50. However, we also see an increase in the number of invalid prefix 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 5.3 that the former exhibits some interesting properties and is more useful for retrieval tasks.

Table 4: Expressions closest to a given query based on the representations generated by ExpEmb-A and ExpEmb-E. It is evident that ExpEmb-E does a better job at learning semantics and the overall structure of the expressions. Refer to Appendix D for more examples.

Query	ExpEmb-A	ExpEmb-E
$4x^2\cos(3x-1)$	$4x^2e^{-3x-1}$	$-10x^2\cos(x-10)$
	$4x^2e^{2x-1}$	$-10x^2\cos(x+10)$
	$4x^2e^{-2x-1}$	$x^2\cos(x-10)$
	$4x^2e^{-3x-9}$	$x^2\cos(x+1)$
	$4x^2e^{-x-1}$	$x^2\cos(x-30)$
$x^2+\log(x)+1$	$\sqrt{x}+x^2+\log(x)$	$x^2+\log(x)+1$
	$\sqrt{x}+x^2+e^x$	$x^2+\log(x)+3$
	$\sqrt{x}+x^2+\operatorname{acos}(x)$	$x^2+\log(x)+2$
	$\log\left(x^{x^2}\right)+\sinh(x)$	$x^2+x+\log(x)+1$
	$\log\left(x^{x^2}\right)+\tan(x)$	$x^2+x+\log(x)$

5.3 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 ExpEmb-E and ExpEmb-A and demonstrate the efficacy of the proposed approach.

For these experiments, we use the *trained* ExpEmb-A and ExpEmb-E models from Section 5.2, and no further training is performed for these experiments. The data used for the experiments are described in their respective sections.

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 a set of 7,000 simple expressions belonging to four categories: hyperbolic, trigonometric, polynomial, and logarithmic/exponential. Each expression is either a polynomial or contains hyperbolic, trigonometric, logarithmic, or exponential operators. Each expression contains only one category of operators. 35% of these expressions are from the training set and the remaining 65% are unseen expressions. Below are a few examples of expressions belonging to each class:

- 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 2 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. ExpEmb-E does a better job at grouping similar expressions together, suggesting its ability to understand semantics. 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 $\tan\left(x\frac{\sqrt{2}}{2}\right)$, $x^2(x^2-x)$, $\sinh^{-1}\left(x\frac{\sqrt{2}}{2}\right)$, and $\log\left(x\frac{\sqrt{2}}{2}\right)$

Table 5: Examples of embedding algebra on the representations generated by ExpEmb-A and ExpEmb-E. The correct predictions are shown in **bold**.

x_1	y_1	y_2	x_2 (expected)	x_2 (predicted) ExpEmb-A	x_2 (predicted) ExpEmb-E
$\cos(x)$	$\sin(x)$	$\csc(x)$	$\sec(x)$	$\boldsymbol{\sec(x)}$	$\boldsymbol{\sec(x)}$
$\cos(x)$	$\sin(x)$	$\cot(x)$	$\tan(x)$	$\sqrt{x}$	$x+\cot(x)$
$\sin(x)$	$\cos(x)$	$\cosh(x)$	$\sinh(x)$	$\tanh(x)$	$\boldsymbol{\sinh(x)}$
$\sin(x)$	$\csc(x)$	$\sec(x)$	$\cos(x)$	$\boldsymbol{\cos(x)}$	$\boldsymbol{\cos(x)}$
$\sin(x)$	$\csc(x)$	$\cot(x)$	$\tan(x)$	$\cos(x)$	$\boldsymbol{\tan(x)}$
$\sin(-x)$	$-\sin(x)$	$\cos(x)$	$\cos(-x)$	$\sin(x)$	$\boldsymbol{\cos(-x)}$
$\tan(-x)$	$-\tan(x)$	$-\cot(x)$	$\cot(-x)$	$\sinh(-x)$	$\boldsymbol{\cot(-x)}$
$\sin(-x)$	$-\sin(x)$	$-\cot(x)$	$\cot(-x)$	$\tan(-x)$	$\boldsymbol{\cot(-x)}$
$\sinh(-x)$	$-\sinh(x)$	$\cosh(x)$	$\cosh(-x)$	$\operatorname{acosh}(x)$	$\boldsymbol{\cosh(-x)}$
$\sinh(-x)$	$-\sinh(x)$	$-\tanh(x)$	$\tanh(-x)$	$\boldsymbol{\tanh(-x)}$	$\boldsymbol{\tanh(-x)}$
$\sinh(-x)$	$-\sinh(x)$	$-\cosh(x)$	$-\cosh(-x)$	$\tan(-x)$	$\cosh(-x)$
$\sinh(-x)$	$-\sinh(x)$	$\tanh(x)$	$-\tanh(-x)$	$\operatorname{acosh}(x)$	$\tanh(-x)$
x^2	x	$\sin(x)$	$\sin^2(x)$	$\boldsymbol{\sin^2(x)}$	$x^2+\sin(x)$
x^2	x	$\cos(x)$	$\cos^2(x)$	$\boldsymbol{\cos^2(x)}$	$\boldsymbol{\cos^2(x)}$
x^2	x	$\log(x)$	$\log(x)^2$	$\log\left(\sqrt{2}\right)$	$\log\left(x^2\right)$
x^2	x	e^x	$(e^x)^2$	x^4	x^2e^x
e^x	x	$2x$	e^{2x}	$\boldsymbol{e^{2x}}$	$2e^x$
x^2	x	$x+2$	$(x+2)^2$	$\tan^2(x)$	x^2+2
$\log(x)$	x	$\sin(x)$	$\log(\sin(x))$	$\boldsymbol{\log(\sin(x))}$	$\sin(\log(x))$
$\sin(x)$	x	$\log(x)$	$\sin(\log(x))$	$\log(\sin(x))$	$\boldsymbol{\sin(\log(x))}$

are grouped together. On the other hand, representations generated by ExpEmb-E capture semantics in addition to the syntactic structure.

Distance Analysis. To understand the applicability of the embeddings for the information retrieval task, we perform distance analysis on embeddings generated by ExpEmb-A and ExpEmb-E. We use all expressions from the training set of the Equivalent Expressions Dataset as the pool of expressions for this experiment. The similarity between two expressions is defined as the inverse of the cosine distance between their embedding vectors. We find the five closest expressions to a given query expression. Table 4 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 syntactic structure and operators. On the other hand, ExpEmb-A focuses on the visual features, like operators, and finds other expressions containing these features. For example, the first query in Table 4 consists of polynomial and trigonometric expressions. The closest expressions computed using ExpEmb-E follow the same structure, whereas ExpEmb-A seems to put more emphasis on the polynomial multiplier. This behavior is also apparent in the second example. We believe that the ability of ExpEmb-E to group similar expressions, in terms of operators and the syntactic structure, 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 embeddings 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 embedding vectors can be used to solve analogies of the form "x_1 is to y_1 as x_2 is to y_2". Following along the same lines, we perform simple algebraic operations on the embeddings generated by ExpEmb-A and ExpEmb-E. For a given triplet of expressions x_1, y_1, and y_2, we compute

$$z = \text{emb}(x_1) - \text{emb}(y_1) + \text{emb}(y_2) \tag{1}$$

where emb represents a function that returns the embedding vector of an input expression. We then find an expression with the embedding vector closest to z in terms of cosine similarity, excluding x_1, y_1, and y_2.

Table 6: $score_5$(%) on UNSEENEQCLASS of the SEMVEC datasets. The scores for EQNET and EQNET-L are from their published work (Allamanis et al., 2017; Liu, 2022).

DATASET	EQNET	EQNET-L	EXPEMB-A		EXPEMB-E	
	$score_5$(%)	$score_5$(%)	$score_5$(%)	TRAINING SET SIZE	$score_5$(%)	TRAINING SET SIZE
BOOL8	58.1	-	30.6	146,488	100.0	16,143,072
ONEV-POLY13	90.4	-	38.7	60,128	99.6	9,958,406
SIMPPOLY10	99.3	-	40.1	31,143	99.8	6,731,858
SIMPBOOL8	97.4	-	36.9	21,604	99.4	4,440,450
BOOL10	71.4	-	10.8	25,560	91.3	3,041,640
SIMPBOOL10	99.1	-	24.4	13,081	95.5	1,448,804
BOOLL5	75.2	-	38.3	23,219	36.0	552,642
POLY8	86.2	87.1	32.7	6,785	87.3	257,190
SIMPPOLY8	98.9	98.0	47.6	1,934	98.9	113,660
SIMPBOOLL5	85.0	72.1	55.1	6,009	71.1	66,876
ONEV-POLY10	81.3	80.0	59.8	767	74.1	25,590
BOOL5	65.8	73.7	36.4	712	57.7	17,934
POLY5	55.3	-	5.7	352	45.2	1,350
SIMPPOLY5	65.6	56.3	28.1	156	20.8	882

To create a pool of expressions for this experiment, we use all expressions from the training set and add any missing expressions that are required for an analogy. We create 20 analogy examples, 12 of which are based on mathematical identities, and the remaining are based on simple substitutions. Table 5 shows the results for EXPEMB-A and EXPEMB-E. It is interesting to observe that EXPEMB-E works for nine examples that are based on mathematical identities. It demonstrates a degree of semantic learning. EXPEMB-A performs poorly for the identity-based examples and gets four substitution-based examples right, demonstrating that EXPEMB-A understands the visual structure to a certain extent. The results of this analysis further bolster the efficacy of EXPEMB-E for learning semantically-rich embeddings.

5.4 Comparison with Existing Methods

As discussed in Section 2, prior works (Allamanis et al., 2017; Liu, 2022) have examined if machine learning models can generate similar representations of mathematically equivalent expressions. It should be noted that these prior works only focus on generating similar representations for mathematically equivalent expressions and do not explore the applicability of their methods to the similarity of non-equivalent expressions. But this historical perspective serves as an established evaluation of the representations generated by EXPEMB-E and EXPEMB-A. If we refer to all mathematically equivalent expressions as belonging to a class, 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

$$score_k(q) = \frac{|\mathbb{N}_k(q) \cap c|}{\min(k, |c|)} \tag{2}$$

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. The expressions in the training and validation sets come from the same classes. The dataset contains 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 measure the generalization to the unseen classes).

Our approach requires data to be in the input-output format. For EXPEMB-E, the input and output are mathematically equivalent expressions. To transform the training set into the input-output format for

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, the input and output are the same expressions. For this setting, we use all the expressions present in the training set. Table 6 shows the training set sizes obtained by these transformations for ExpEmb-E and ExpEmb-A. We use the validation and test sets in their original form.

As mentioned in Section 5.1, both ExpEmb-A and ExpEmb-E are trained with a model dimensionality of 64. To enable a fair comparison with the existing approaches, we use the same model configuration and hyperparameter values for all 14 SemVec datasets. Refer to Appendix B for a detailed description of the model configuration and hyperparameter values.

For evaluating our models, we use the UnseenEqClass test set. Table 6 shows the scores achieved by our approach. We observe that the representations learned by ExpEmb-E capture semantics and not just the syntactic structure. It should be noted that our approach does not use the distance between representations at the time of training and only trains the model to generate equivalent expressions, whereas the training for EqNet and EqNet-L explicitly pushes the representations of equivalent expressions closer in the embedding vector space. We further observe that ExpEmb-E performs better than the existing approaches on the datasets with a sufficiently high number of training examples. For other datasets, the model does not perform well on UnseenEqClass but does well on SeenEqClass and the validation set (See Appendix C for details). We believe that this is because of not having enough examples in the training set for generalization to the unseen classes, and these datasets may perform better with simpler Seq2Seq models and different hyperparameters. We leave this exploration for future work.

In our experiments, ExpEmb-A does not perform as well as ExpEmb-E. One possible reason for this behavior may be the less number of examples for training the model in the autoencoder setting. To rule out this possibility, we train models with three model dimensions of 32, 64, and 128 with 179K, 703K, and 2.8M parameters, respectively. We do not observe any significant difference in the performance across these model dimensions. The model also does not achieve good scores on the training and validation sets (See Appendix C for details). Furthermore, ExpEmb-E performs better than ExpEmb-A on the datasets with comparable training set sizes, for example, consider ExpEmb-A on Bool8 vs ExpEmb-E on SimpPoly8. With these observations, we conclude with good certainty that the representations learned by ExpEmb-A are inferior to those learned by ExpEmb-E in terms of capturing semantics.

6 Conclusion

In this work, we developed a framework to represent mathematical expressions in a continuous vector space. We showed that a Seq2Seq model could be trained to generate expressions that are mathematically equivalent to the input. The encoder of this model could be used to generate vector representations of mathematical expressions. We performed quantitative and qualitative experiments and demonstrated that these representations were semantically rich. Our experiments also showed that these representations are better at grouping similar expressions and perform better on the analogy task than the representations generated by an autoencoder.

We also note certain limitations. There is a need for downstream tasks or better evaluation metrics to evaluate the quality of the learned representations of mathematical expressions. Though we evaluated our approach on the SemVec datasets, we could not perform a similar evaluation on the Equivalent Expressions Dataset due to a limited number of equivalent expressions per expression. Furthermore, the dataset presented in this work consists of univariate expressions that contain simple algebraic and transcendental operators. It may be extended to include expressions with multiple variables and more complex operators. We leave these avenues for future research.

References

Saleem Ahmed, Kenny Davila, Srirangaraj Setlur, and Venu Govindaraju. 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)*, pp. 6282–6289, January 2021. doi: 10.1109/ICPR48806.2021.9412619.

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

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv:1810.04805 [cs]*, May 2019.

Dallas Fraser, Andrew Kane, and Frank Wm. Tompa. Choosing Math Features for BM25 Ranking with Tangent-L. In *Proceedings of the ACM Symposium on Document Engineering 2018*, number 17, pp. 1–10. Association for Computing Machinery, New York, NY, USA, August 2018. ISBN 978-1-4503-5769-2.

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

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

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015.

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

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

Giovanni Yoko Kristianto, Goran Topić, and Akiko Aizawa. Utilizing dependency relationships between math expressions in math IR. *Information Retrieval Journal*, 20(2):132–167, April 2017. ISSN 1386-4564, 1573-7659. doi: 10.1007/s10791-017-9296-8.

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

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

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

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

Daniel W. Lozier. NIST Digital Library of Mathematical Functions. *Annals of Mathematics and Artificial Intelligence*, 38(1):105–119, May 2003. ISSN 1573-7470. doi: 10.1023/A:1022915830921.

Behrooz Mansouri, Shaurya Rohatgi, Douglas W. Oard, Jian Wu, C. Lee Giles, and Richard Zanibbi. Tangent-CFT: An Embedding Model for Mathematical Formulas. In *Proceedings of the 2019 ACM SIGIR International Conference on Theory of Information Retrieval*, ICTIR '19, pp. 11–18, Santa Clara, CA, USA, September 2019. Association for Computing Machinery. ISBN 978-1-4503-6881-0. doi: 10.1145/3341981.3344235.

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. Sympy: symbolic computing in python. *PeerJ*

Computer Science, 3:e103, January 2017. ISSN 2376-5992. doi: 10.7717/peerj-cs.103. URL `https://doi.org/10.7717/peerj-cs.103`.

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

Shuai Peng, Ke Yuan, Liangcai Gao, and Zhi Tang. Mathbert: A pre-trained model for mathematical formula understanding. *arXiv preprint arXiv:2105.00377*, 2021.

Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532–1543, Doha, Qatar, October 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1162.

Moritz Schubotz, Alexey Grigorev, Marcus Leich, Howard S. Cohl, Norman Meuschke, Bela Gipp, Abdou S. Youssef, and Volker Markl. 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, pp. 135–144, New York, NY, USA, July 2016. Association for Computing Machinery. ISBN 978-1-4503-4069-4. doi: 10.1145/2911451.2911503.

Moritz Schubotz, Leonard Krämer, Norman Meuschke, Felix Hamborg, and Bela Gipp. Evaluating and Improving the Extraction of Mathematical Identifier Definitions. In Gareth J.F. Jones, Séamus Lawless, Julio Gonzalo, Liadh Kelly, Lorraine Goeuriot, Thomas Mandl, Linda Cappellato, and Nicola Ferro (eds.), *Experimental IR Meets Multilinguality, Multimodality, and Interaction*, Lecture Notes in Computer Science, pp. 82–94, Cham, 2017. Springer International Publishing. ISBN 978-3-319-65813-1. doi: 10.1007/978-3-319-65813-1_7.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 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*, pp. 353–355, Brussels, Belgium, November 2018. Association for Computational Linguistics. doi: 10.18653/v1/W18-5446. URL `https://aclanthology.org/W18-5446`.

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

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

A Equivalent Expression Generation

We use SymPy to generate mathematically equivalent expressions for a given formula. We apply the operations shown in Table 7 to get mathematically equivalent expressions. The examples shown in the table are intended to give an idea about each function. The actual behavior of these functions is much more complex. Refer to the SymPy documentation for more details.[2] We use the `REWRITE` function for expressions containing trigonometric or hyperbolic operators. Specifically, we use this function to rewrite an expression containing trigonometric (or hyperbolic) operators in terms of other trigonometric (or hyperbolic) operators.

Table 7: List of SymPy functions with examples that are used to generate equivalent expressions.

Function	Input	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	$2\tanh\left(\frac{x}{2}\right)(1 - \tanh^2\left(\frac{x}{2}\right))^{-1}$

B Training Details

We use the PyTorch implementation of the Transformer architecture for our experiments with a modified version of the decoder to enable caching to speed up the generation process at inference.[3] We use 6 encoder layers, 6 decoder layers, 8 attention heads, 0.1 dropout probability, and the ReLU activation for our experiments. The layer normalization is performed in the encoder and decoder layers before other attention and feedforward operations. We use the Adam optimizer with a learning rate of 10^{-4}. We use a fixed seed of 42 for reproducibility and label smoothing while computing the loss.

For the Equivalent Expressions Dataset, we use a model dimension of 512, a feedforward dimension of 2048, and a batch size of 256. These experiments were run on two 32GB V100 GPUs. We use early stopping with 300K minimum steps, 1M maximum steps, and patience of 30K steps. In terms of epochs, this translates to 17 minimum epochs, 55 maximum epochs, and patience of 2 epochs for ExpEmb-E and 28 minimum epochs, 94 maximum epochs, and patience of 3 epochs for ExpEmb-A. We evaluate the model at the end of every epoch.

For the SemVec datasets, we use a model dimension of 64, a feedforward dimension of 256, and a batch size of 512. These experiments were run on one 16GB V100 GPU. We use early stopping with 50K minimum steps, 1M maximum steps, and patience of 20K steps or 2 epochs (whichever is smaller). Table 8 shows these values in terms of epochs for different datasets. We evaluate the model at the end of every epoch.

C Results for the SemVec Datasets

To compute $score_k(q)$ for our model, we use the source code provided by Allamanis et al. (2017).[4]

Tables 9 and 10 show the scores achieved by ExpEmb-A and ExpEmb-E on the validation and SeenEq-Class test sets of the SemVec datasets, respectively. For ExpEmb-A, we try three model dimensions of 32, 64, and 128 to rule out the effect of underfitting and overfitting while training the model. Table 11

[2]docs.sympy.org/latest/tutorial/simplification.html
[3]pytorch.org/docs/stable/generated/torch.nn.Transformer.html
[4]github.com/mast-group/eqnet

Table 8: The minimum, maximum, and patience epochs for early stopping used in the SemVec training.

Dataset	ExpEmb-A			ExpEmb-E		
	Minimum	Maximum	Patience	Minimum	Maximum	Patience
Bool8	175	3,485	70	2	32	2
oneV-Poly13	424	8,475	170	3	52	2
SimpPoly10	820	16,394	328	4	77	2
SimpBool8	1,163	23,256	466	6	116	3
Bool10	1,000	20,000	400	9	169	4
SimpBool10	1,924	38,462	770	18	354	8
BoolL5	1,087	21,740	435	47	926	19
Poly8	3,572	71,429	1,429	100	1,989	40
SimpPoly8	12,500	250,000	5,000	226	4,505	91
SimpBoolL5	4,167	83,334	1,667	382	7,634	153
oneV-Poly10	25,000	500,000	10,000	1,000	20,000	400
Bool5	25,000	500,000	10,000	1,389	27,778	556
Poly5	50,000	1,000,000	20,000	16,667	333,334	6,667
SimpPoly5	50,000	1,000,000	20,000	25,000	500,000	10,000

Table 9: $score_5(\%)$ achieved by ExpEmb-A and ExpEmb-E on the validation sets of the SemVec datasets.

Dataset	ExpEmb-A			ExpEmb-E
	$D_{\text{MODEL}} = 32$	$D_{\text{MODEL}} = 64$	$D_{\text{MODEL}} = 128$	
Bool8	36.2	36.3	35.5	100.0
oneV-Poly13	38.3	39.0	38.5	99.9
SimpPoly10	37.5	31.5	34.0	99.9
SimpBool8	42.3	46.7	44.5	99.8
Bool10	9.6	9.9	9.1	93.9
SimpBool10	23.2	26.6	27.2	96.3
BoolL5	47.4	47.6	47.7	98.2
Poly8	36.4	36.5	36.0	98.6
SimpPoly8	36.5	35.4	34.6	99.8
SimpBoolL5	75.0	76.1	74.4	99.5
oneV-Poly10	42.1	46.3	42.5	83.0
Bool5	42.4	42.1	35.4	81.0
Poly5	11.6	9.5	7.4	43.2
SimpPoly5	20.0	28.8	32.5	43.8

shows the scores achieved by these models on UnseenEqClass. It can be seen that changing the number of parameters in the model does not have a significant effect on the scores.

We also perform an evaluation of compositionality on the SemVec datasets (Allamanis et al., 2017). For this evaluation, we train our model on a simpler dataset and evaluate it 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 12.

D Distance Analysis

Table 14 shows more examples of the distance analysis as discussed in Section 5.3.

Table 10: $score_5$(%) achieved by ExpEmb-A and ExpEmb-E on SeenEqClass of the SemVec datasets.

Dataset	ExpEmb-A $D_{\text{model}} = 32$	ExpEmb-A $D_{\text{model}} = 64$	ExpEmb-A $D_{\text{model}} = 128$	ExpEmb-E
Bool8	36.1	36.1	35.5	100.0
oneV-Poly13	38.0	38.5	38.1	99.9
SimpPoly10	37.7	32.2	34.5	99.9
SimpBool8	41.6	46.1	43.5	99.8
Bool10	9.6	9.8	8.9	94.0
SimpBool10	23.0	26.8	27.3	96.5
BoolL5	40.5	42.0	42.2	68.7
Poly8	34.9	35.0	34.5	95.2
SimpPoly8	38.5	36.7	36.1	99.6
SimpBoolL5	58.2	59.3	58.3	80.9
oneV-Poly10	36.1	39.6	37.9	77.3
Bool5	41.1	43.2	34.6	80.2
Poly5	14.7	8.2	4.2	46.2
SimpPoly5	30.9	32.6	42.2	47.9

Table 11: $score_5$(%) achieved by ExpEmb-A on UnseenEqClass of the SemVec datasets for different model dimensions.

Dataset	$D_{\text{model}} = 32$	$D_{\text{model}} = 64$	$D_{\text{model}} = 128$
Bool8	31.1	30.6	25.9
oneV-Poly13	37.7	38.7	38.4
SimpPoly10	47.4	40.1	44.9
SimpBool8	33.1	36.9	34.1
Bool10	10.7	10.8	8.6
SimpBool10	20.5	24.4	25.4
BoolL5	35.6	38.3	38.2
Poly8	32.2	32.7	32.1
SimpPoly8	50.4	47.6	47.3
SimpBoolL5	55.2	55.1	54.9
oneV-Poly10	58.3	59.8	59.7
Bool5	36.7	36.4	28.1
Poly5	14.9	5.7	6.6
SimpPoly5	18.8	28.1	17.7

E Equivalent Expression Generation

For the Equivalent Expressions Dataset, Table 13 shows the accuracy of ExpEmb-A and ExpEmb-E on the validation set of the Equivalent Expressions Dataset with beam sizes of 1, 10, and 50.

Table 12: Compositionality test of ExpEmb-E. $score_5$(%) achieved on UnseenEqClass of a SemVec dataset represented by To by a model trained on the dataset represented by From. For scores achieved by EqNet, refer to Figure 7 in Allamanis et al. (2017).

From → To	$score_5$(%)
Bool5 → Bool10	12.5
Bool5 → Bool8	32.5
BoolL5 → Bool8	29.3
oneV-Poly10 → oneV-Poly13	29.8
Poly5 → Poly8	32.9
Poly5 → SimpPoly10	45.1
Poly8 → oneV-Poly13	29.9
Poly8 → SimpPoly10	58.9
Poly8 → SimpPoly8	98.9
SimpPoly5 → SimpPoly10	45.5
SimpPoly8 → SimpPoly10	65.9

Table 13: Accuracy of ExpEmb-A and ExpEmb-E on the validation set of the Equivalent Expressions Dataset.

Beam Size	ExpEmb-A	ExpEmb-E
1	0.9995	0.7615
10	0.9995	0.8365
50	0.9995	0.8665

Table 14: Expressions closest to a given query computed based on embeddings generated by ExpEmb-A and ExpEmb-E.

Query	ExpEmb-A	ExpEmb-E
$21x - 3\sin(x)$	$-x\log(x) + 21x$	$210x + 210\sin(x)$
	$-3x\operatorname{acosh}(x) + 21x$	$-60x + 20\sin(x)$
	$2x - \frac{16\log(x)}{3}$	$11x + 11\sin(x)$
	$-10x\cos(x) + 21x$	$21x + 21\cos(x)$
	$-xe^x + 21x$	$-70x + 10\sin(x)$
$\sin(x) + \cosh^2(x)$	$\cosh^2(x) + \cosh(x)$	$\cosh^2(x) + \cosh(x)$
	$\sin(x) + \cos^2(x)$	$\sin(x) + \cos^2(x)$
	$\sin(x) + \tan^2(x)$	$\sin^2(x) + \sin(x)$
	$\sin^2(x) + \sin(x)$	$\cos(x) + \frac{1}{\cosh^{10}(x)}$
	$\operatorname{acos}^2(x) + \operatorname{acos}(x)$	$\log(x)^{20} + \sin(x)$
$x^2 + 5$	$x^2 + e^e$	$x^5 + x^2$
	$x^2 + \operatorname{atanh}(x)$	$x^2 + 5$
	$x^2 + \tanh(x)$	$x^6 + x^2$
	$x^2 + \sinh(x)$	$x^4 + x^2$
	$x^2 + \cosh(x)$	$\left(x^2 + 1\right)^5$
$\sinh\left(x^2 + 1\right)$	$\sqrt{x^2 + \log(9)}$	$\operatorname{atanh}\left(x^2 + 1\right)$
	$\log\left(x^2 + \log(9)\right)$	$-\operatorname{atanh}\left(x^2 + 1\right)$
	$\log\left(x^2 + \log(8)\right)$	$\operatorname{atanh}\left(x^2 + 5\right)$
	$\log\left(x^2 + ex\right)$	$\operatorname{atanh}\left(x^2 + 2\right)$
	$e^{x^2 + e^4}$	$\operatorname{atanh}\left(x^2 + 4\right)$
$2x + 1 + e^{-x}$	$\log(x) + \log(2) + e^{-x}$	$x^2 + 2x + e^{-x}$
	$\log(2x) + e^{-x}$	$x^2 + 2x + 1 + e^{-x}$
	$e^{-x} + e^{-2x}$	$2x + e^x + e^{-x}$
	$(-x + \log(2x))^5$	$x^2 + 3x + e^{-x}$
	$\left(-x + e^{2x}\right)^4$	$3x + e^x + e^{-x}$
$\sin^2(x) + e^{-x}$	$\log\left(x^2\right) + e^{-x}$	$\sin(x) + e^{-10x}$
	$\log\left(x^5\right) + e^{-x}$	$x + \sin(x) + e^{-10x}$
	$\log\left(x^3\right) + e^{-x}$	$e^{20x} + \sin(x)$
	$\log\left(x^4\right) + e^{-x}$	$e^{10x} + \sin(x)$
	$\log(2x) + e^{-x}$	$x + e^{10x} + \sin(x)$
$x^2\log(x)$	$x^2\operatorname{acosh}(x)$	$x^4\log(x)$
	$x^3\log(x)$	$x^3\log(x)$
	$x^2\operatorname{atanh}(x)$	$2x^2\log(x)$
	$x^2\operatorname{acos}(x)$	$3x^2\log(x)$
	$x^2\operatorname{asinh}(x)$	$4x^2\log(x)$
$\log\left(x + \sqrt{x^2 + 1}\right)$	$\frac{1}{x^2 + x + 2\log(2) + 2}$	$\log\left(x^{10} + x - 10 + \frac{1}{\csc(10)}\right)$
	$\frac{4}{\frac{2x^2}{3} + 1}$	$\log\left(\frac{1}{\csc(10)} + \frac{1}{x^{10}}\right)$
	$\frac{5}{5x^2 + x + 10}$	$\log\left(x - \frac{10}{\csc(10)}\right)$
	$\frac{1}{x^2 + x + 1 + 2\log(2)}$	$\log\left(x - \frac{10}{\csc(20)}\right)$
	$\log\left(x + \frac{1}{\sec(250)}\right)$	$\log\left(\frac{x}{2\log(2)} + x + 1\right)$