Learning Mathematical Properties of Integers

Maria Ryskina*
Language Technologies Institute
Carnegie Mellon University
mryskina@cs.cmu.edu

Kevin Knight
DiDi Labs
kevinknight@didiglobal.com

Abstract

Embedding words in high-dimensional vector spaces has proven valuable in many natural language applications. In this work, we investigate whether similarly-trained embeddings of integers can capture concepts that are useful for mathematical applications. We probe the integer embeddings for mathematical knowledge, apply them to a set of numerical reasoning tasks, and show that by learning the representations from mathematical sequence data, we can substantially improve over number embeddings learned from English text corpora.

1 Introduction

Word vector representations learned by neural models have been shown to capture linguistic knowledge that can be useful for downstream NLP tasks (Belinkov and Glass, 2019). In this work, we look at whether similarly-trained integer embeddings can represent useful mathematical knowledge. As a simple example, a mathematician may look at a few values of the function $2^n - 1$:

n	1	2	3	4	5	6	7	...
$2^n - 1$	1	3	7	15	31	63	127	...

and notice patterns such as *'if n is even, then $2^n - 1$ is divisible by 3'*. This type of reasoning is frequent in the early stages of mathematical work. In order for software to assist people in identifying such patterns, it needs to have an internal representation of the basic properties of integers. Just as learned word representations can capture attributes like the word's part of speech or syntactic dependency label (Köhn, 2015; Belinkov et al., 2017), we would like to develop integer embeddings that capture primality, divisibility by 3, etc.

In this paper, we learn integer embeddings from mathematical resources and probe them for mathematical knowledge. Unlike most natural-language features, many number-theoretic properties are highly regular and often may be easily determined from the integer value itself (e.g. converting an integer to its binary representation instantly reveals whether it is a power of 2). However, our focus is not on building classifiers to predict these properties, but rather on exploring whether this information can be learned from the integer co-occurrence statistics in mathematical data and how it is distributed across the vector's dimensions.

```
[A000040]: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, ...   (primes)
[A000045]: 0, 1, 1, 2, 3, 5, 8, 13, 21, ...          (Fibonacci)
[A000108]: 1, 1, 2, 5, 14, 42, 132, 429, ...         (Catalan)
[A005132]: 0, 1, 3, 6, 2, 7, 13, 20, 12, 21, ...     (Recamán)
```

Figure 1: Example sequences and their IDs in the Online Encyclopedia of Integer Sequences (OEIS).

Besides that, we are also interested in whether the integer representations learned from context can capture meaningful mathematical properties without us having to define them explicitly. It is not clear how one would probe for such unspecified features, but we can instead estimate the usefulness of the learned embeddings for downstream numerical reasoning tasks. In this work, we apply several types of integer embeddings to a set of tasks built around mathematical regularities, and show that embeddings learned from integer sequences yield better performance than the ones trained on text.

2 Data: OEIS

Our source of mathematical knowledge is the Online Encyclopedia of Integer Sequences (OEIS), a well-known database of number sequences representing properties that are of interest to mathematicians (Sloane, 2003). Several samples from OEIS are shown in Figure 1. While some OEIS sequences may be recognized by non-mathematicians (e.g. the Fibonacci sequence `[A000045]`), others are based on complex mathematical regularities and would be extremely difficult for a layperson to interpret (e.g. the greatest possible number of diag-

*Research performed at DiDi Labs.

Proceedings of the Fourth BlackboxNLP Workshop on Analyzing and Interpreting Neural Networks for NLP, pages 389–395

Method	Corpus	Tokens	Types
Ours	OEIS	14M	133,004
GloVe–840B–300D	Common Crawl	840B	64,729
SkipGram–BoW–5	Wikipedia	∼2.5B	2,272
FastText–Wiki	Wikipedia, UMBC, `statmt.org`	16B	12,037

Table 1: Training corpus details for the integer embeddings used in this work. OEIS types here only include integers that occur 3 or more times; all others are replaced by UNK.

	Train	Dev	Test
Sequences	302,281	16,793	16,793
Integer tokens	12,979,924	719,808	719,237
Mean sequence length	43	43	43
Integer types	1,531,064	128,488	127,976
Singleton types	1,268,771	–	–
Token OOV rate	0%	10.0%	9.8%

Table 2: Summary of the OEIS data splits used in this work. The development split is used for model selection and the test split is used in the sequence completion experiments. Type statistics are reported without minimum count filtering. Singleton types refer to the integers that occur only once in the training set.

onals of a polyhedron with n faces [A279015]).

The sequential structure of OEIS allows us to use it for training NLP models developed for textual data, such as recurrent neural network (RNN) language models and co-occurrence-based word embeddings. Of the 336K sequences in OEIS, we allocate 90% for training and divide the rest into development (used for hyperparameter tuning) and test splits; split statistics are listed in Table 2. Of the 1.5M integer types in the vocabulary, 83% appear in training only once. The sequences are represented by their n first elements ($n = 43$ on average), so the coverage of larger integers in OEIS is sparser, and they make up most of the out-of-vocabulary items in the development and test sets.[1]

3 Integer Embeddings

In our experiments, we compare the integer embeddings learned from OEIS with the representations of integer tokens in the vocabulary of word embedding models trained on English text. Table 1 details the training corpus statistics for all embeddings.

3.1 Embeddings Learned from OEIS

Motivated by the distributional semantics paradigm, we learn multi-dimensional embedding vectors from the contexts in which integers occur in OEIS. We use three training methods, well-known in NLP:

LSTM embeddings: we train a two-layer LSTM language model on the 300K OEIS training sequences and extract the 100-dimensional vectors from the weight matrix of its embedding layer.[2]

LSA embeddings: Latent Semantic Analysis (Hofmann, 1999) performs dimensionality reduction on a "document-term" matrix using truncated singular value decomposition. The rows of the matrix represent the integer sequences (as available in OEIS), the columns stand for the integer types, and the values in the cells are equal to the number of occurrences of each integer in each sequence. The resulting matrix is sparse, and its low rank yields vectors with a maximum of 65 dimensions. This method takes OEIS co-occurrences into account, but not the ordering within the sequences.

FastText: we learn 100-dimensional FastText embeddings (Bojanowski et al., 2017), both with and without the additional subword-level information.

3.2 Pre-trained Embeddings

Prior work has successfully employed word representations that are pre-trained on large text corpora for NLP applications. Naturally, these text corpora include numerical data as well, and some properties of a number can be inferred from the textual context (e.g., if an integer follows the phrase *the year*, it most likely has four digits). While the training data for these embeddings is not designed to encode integer properties, it may contain billions of tokens, compared to only ∼13M in OEIS. It has been shown that word embeddings retain some knowledge of the integer properties (Naik et al., 2019), so we exploit these resources as a baseline.

We use three sets of pre-trained embeddings: GloVe (Pennington et al., 2014), SkipGram bag-of-words (Levy and Goldberg, 2014), and FastText (Bojanowski et al., 2017). For all three models, we select the versions that performed best in the numerical knowledge tests of Naik et al. (2019).[3]

Thawani et al. (2021) provide a comprehensive description of prior work on number representations learned from text. Prior work concentrates on basic numeracy (counting, paraphrasing, rela-

[1] For the OEIS data download details, see Appendix A.

[2] For implementation details, see Appendix C.

[3] Specific model versions and links to download them are listed in Appendix B.

Embeddings	Evenness		Divisibility by 3		Divisibility by 4		Primality	
	Single	All	Single	All	Single	All	Single	All
Random baseline	0.50	0.50	0.67	0.67	0.75	0.75	0.87	0.87
GloVe–840B–300D	0.51	0.76	0.67	0.47	0.75	0.71	0.87	0.84
SkipGram–BoW–5	0.50	0.52	0.67	0.67	0.75	0.75	0.87	0.87
FastText–Wiki	0.51	0.61	0.67	0.66	0.75	0.76	0.87	0.62
OEIS–LSTM	0.69	0.93	0.67	0.71	0.76	0.81	0.86	0.95
OEIS–LSA	**0.80**	0.62	0.67	0.67	0.75	0.75	0.87	0.87
OEIS–FastText (with subword units)	0.78	**1.00**	**0.69**	0.94	**0.80**	**1.00**	0.82	**1.00**
OEIS–FastText (no subword units)	0.59	**1.00**	0.68	**0.98**	0.77	0.98	0.86	**1.00**
Concatenate(OEIS–FastText, FastText–Wiki)	0.78	**1.00**	**0.69**	0.94	**0.80**	**1.00**	0.82	**1.00**

Table 3: Accuracies of the logistic regression probing classifiers for three binary properties. The classifiers are trained on integers 1–1000, and results are reported on 1001–2000. "All" uses the entire integer embedding as input, while "Single" uses only the most relevant component of the embedding, chosen by train (1–1000) accuracy. Highest accuracy in each experiment is shown in **bold**, if it is exceeds the random choice baseline.

Embeddings	Value (R^2)		Magnitude (R^2)	
	Single	All	Single	All
GloVe–840B–300D	0.82	**1.00**	0.65	**0.99**
SkipGram–BoW–5	0.79	0.99	0.70	0.97
FastText–Wiki	**0.90**	0.99	**0.76**	0.98
OEIS–LSTM	0.49	0.86	0.31	0.78
OEIS–LSA	0.08	0.55	0.13	0.53
OEIS–FastText (with subword units)	0.38	0.99	0.33	0.96
OEIS–FastText (no subword units)	0.49	0.99	0.39	0.95
Concatenate(OEIS–FastText, FastText–Wiki)	**0.90**	**1.00**	**0.76**	0.98

Table 4: Performance of the linear regression models trained to predict an integer's value and order of magnitude (number of digits). We fit the regression models to integers 1–2000 and evaluate the coefficient of determination R^2 of the fit on the same set (between 0 and 1, higher is better). Best results in each experiment shown in **bold**.

tive magnitude, word problems), while we focus on integer properties relevant to mathematical fields such as number theory.

4 What's in an Integer Embedding?

For the learned vectors to be useful for mathematical applications, they need to contain the information about the important integer properties, and this information can be distributed over one or several neurons. For example, in the LSTM embeddings trained on OEIS, we quickly find an "evenness neuron": the 156th element of the embedding vector v is generally positive for even numbers and negative for odd numbers. This holds for integers from 1 to 50 with only a few exceptions, e.g.:

$$v_{156}(1) = 0.15 \qquad v_{156}(6) = 0.38$$
$$v_{156}(2) = 0.29 \qquad v_{156}(7) = -0.31$$
$$v_{156}(3) = -0.04 \qquad v_{156}(8) = 0.39$$
$$v_{156}(4) = 0.26 \qquad v_{156}(9) = -0.02$$
$$v_{156}(5) = -0.08 \qquad v_{156}(10) = 0.43$$

Probing classifiers. We use probing classifiers to test whether the embeddings learn mathematical properties. To avoid spurious correlations, we keep our classifiers very simple. We start by probing for three binary properties: divisibility by 2, 3, and 4, and primality. We train a logistic regression classifier on integers from 1 to 1000 and evaluate its predictions on integers from 1001 to 2000.

Table 3 shows the prediction accuracies for classifiers trained on each embedding type, using either the entire vector or the single most predictive dimension. Generally speaking, the classifiers trained on LSA vectors and pre-trained word embeddings perform on par with the random baseline, providing no evidence of containing the relevant information. However, the OEIS-trained LSTM and FastText vectors can be reliably used to predict mathematical properties of integers, strongly outperforming the baseline in all cases.

Subword information (in our case, digits) is also useful for inferring mathematical properties: e.g.,

Prompt	Answer	Explanation
0, 2, 4, 6,	8	$a_{n+1} = a_n + 2$ (arithmetic progression)
0, 1, 1, 2, 3, 5,	8	$a_{n+2} = a_n + a_{n+1}$ (Fibonacci numbers)
65536, 256, 16,	4	$a_{n+1} = \sqrt{a_n}$
6, 28, 12, 14, 24, 7,	48	$a_{2n+1} = 2 \cdot a_{2n-1}, a_{2n+2} = a_{2n}/2$ (alternating geometric progressions)
32, 35, 39, 44,	50	$a_{n+1} = a_n + n + 2$ (increments forming an arithmetic progression)

Table 5: Example numerical sequence completion problems used in our experiments, along with the rationales behind the gold answers. All questions shown here are collected from the test preparation website Nibcode. In our experimental setup, the language model encodes the prompt and needs to predict the most likely continuation.

a :	b ::	c :	?	Incorrect answer options			Explanation
5 :	36 ::	6 :	49	48	50	56	$x : (x+1)^2$
42 :	56 ::	72 :	90	81	92	100	$x(x+1) : (x+1)(x+2)$
48 :	122 ::	168 :	290	215	225	292	$(x^2-1) : [(x+4)^2+1]$
210 :	380 ::	182 :	342	156	240	272	$(x^2-x) : [(x+5)^2-(x+5)]$
11529 :	7235 ::	152943 :	213549	62034	163044	203448	The sum of the digits in each pair is the same

Table 6: Sample mathematical analogy problems used in our experiments, along with the rationales behind the gold answers; the questions shown here are collected from the test preparation website LearnFrenzy. All analogy problems are of the form *a:b :: c:?*, and the task is to choose one of the 3–5 provided options that would most appropriately fill in the blank. The column labeled "?" shows the expected correct answer, and the numbers shaded in grey are the remaining answer options. In our experiments with integer embeddings, we use vector arithmetic to find the projected vector of the answer and then select its nearest neighbor among the presented options.

divisibility by 4 can be determined from the last two digits of a number. The subword-based version of FastText averages vectors for all digit 3-grams to 6-grams, plus the vector for the whole integer. We get further improvements by training classifiers on a concatenation of OEIS–FastText and FastText–Wiki vectors, both trained with subword units.

We also probe for the number's order of magnitude (number of digits) and for the value of the integer itself using a linear regression (Table 4): we fit it on integers 1–2000 and evaluate the coefficient of determination R^2 of the fit. In this case, vectors pre-trained on large text corpora perform well.

Completing integer sequences. The task of predicting the next integer in a given sequence has long been used in aptitude testing as a measure of inductive reasoning skills.[4] We collect 57 sequence completion problems from online test preparation websites offering practice questions. Table 5 shows five sample sequence completion problems from our dataset, annotated with answer explanations.[5]

Most sequences test the subject's ability to recognize common patterns like arithmetic progression (0, 2, 4, 6, ? → 8) or well-known sequences like Fibonacci numbers (0, 1, 1, 2, 3, 5, ? → 8), but some sequences feature additional distractor patterns such as multiple distinct sequences alternating or the increments themselves forming a sequence following another pattern (32, 35, 39, 44, ? → 50).

Table 7 compares the performance of the LSTM model trained on OEIS with two baselines:

Baseline 1: GPT-2. We predict the next token using the OpenAI GPT-2 (Radford et al., 2019) language model.[6] GPT-2 can memorize sequences very well, but it is not specifically trained for mathematical generalization. GPT-2 achieves slightly higher accuracy (Precision@1) than our LSTM model but substantially lower Precision@5.

Baseline 2: Search. We match the prompt directly against the OEIS database and return the most frequent continuation. This is a strong baseline: 56% of the test set questions occur somewhere in OEIS followed by the gold answer at least once.

While human aptitude questions focus on a specific range of patterns that are relatively easy for people to identify, we also want to test the predictive abilities on the OEIS test set sequences which showcase more sophisticated phenomena. The LSTM embeddings yield higher accuracy in that case (lower half of Table 7). The search baseline is not as effective for this task because the

[4]We note the non-uniqueness of solutions is a persistent criticism of these tests (Korossy, 1998).

[5]For the full list of sequence completion problems and their sources, see https://github.com/ryskina/integer-embedding-tests

[6]We use the AllenNLP prediction demo: https://demo.allennlp.org/next-token-lm

Test set	Method	P@1	P@5
Aptitude tests	OEIS–LSTM	0.05	0.37
	GPT-2	0.07	0.19
	OEIS search	**0.53**	**0.56**
OEIS test set	OEIS–LSTM	**0.14**	**0.26**
	OEIS search (full)	0.02	0.02
	OEIS search (last 5)	0.12	0.17

Table 7: Performance on the sequence completion task. Precision@k measures how often the reference answer is among the top k predicted continuations. For the OEIS search baseline, we sort the possible next tokens by frequency of occurring after the given sequence prefix (using either the full prefix or its last five elements).

prompts are now much longer (42 tokens on average compared to 5). Limiting the search to only the last 5 tokens of the prompt improves accuracy, but it still does not outperform the OEIS–LSTM.

Mathematical analogies. A traditional benchmark for evaluating word representations is word analogy, i.e. answering questions of the form "*a* is to *b* as *c* is to ..." (Mikolov et al., 2013a). We perform a similar mathematical analogy test, collecting 79 questions from numerical aptitude practice tests. All questions are multiple-choice (3 to 5 answer options), to mitigate non-uniqueness of the solution. Table 6 shows five sample analogy problems from our dataset, annotated with the intended analogy explanations.[7]

Following Mikolov et al. (2013b), we use vector arithmetic to solve analogies: out of the given options, for the task *a:b :: c:?* we choose the number whose embedding has the highest cosine similarity to the vector $v(c) - v(a) + v(b)$. Table 8 shows that only the FastText embeddings learned from OEIS outperform the random choice baseline on this task. We believe this is due to the linear regularity encoded in the embeddings by design (lacking in LSA and LSTM), and the training data that highlights mathematical properties useful for the task (as compared to FastText–Wiki).

Expanding integer seed sets. In mathematical exploration, we often have a small set S of integers that exhibit some behavior and aim to find other integers that behave similarly. We test the models' ability to expand a given set by finding the centroid of the embeddings of S's members and sorting candidates by cosine distance to the centroid.

[7]Full list of mathematical analogy problems and their sources can be found at https://github.com/ryskina/integer-embedding-tests

Method	Accuracy
Random choice baseline	0.28
OEIS–LSTM	0.25
OEIS–LSA	0.27
OEIS–FastText	**0.34**
FastText–Wiki	0.18

Table 8: Peformance on the multiple-choice numerical analogy questions. The answer to *a:b :: c:?* is chosen by highest cosine similarity to $v(c) - v(a) + v(b)$. OEIS–FastText is trained with subword information.

Seed set	Vectors	Candidate expansions
5 13 29	OEIS–FT	**19, 7, 17, 3,** 9, **23**
	FT–Wiki	26, 12, 28, 14, 27, 15
73 97 83	OEIS–FT	**79, 71, 67, 89,** 77, **103**
	FT–Wiki	82, 81, 78, 84, 76, **79**
729 1024 243	OEIS–FT	**2187, 81, 256, 64, 27, 512**
	FT–Wiki	768, **256**, 640, 840, 384, 216

Table 9: Given a small seed set of integers, we predict other candidate members of the set, ranked by the cosine similarity of their vectors to the centroid of the seed set. Methods include OEIS–FastText and FastText–Wiki, both trained with subword information.

Some qualitative examples are shown in Table 9. Given the seed set **5 13 29**, top results given by the OEIS-trained FastText include small prime numbers, while for **73 97 83** we mostly get prime numbers of similar magnitude. The seed **729 1024 243** returns other powers of 2 and 3. By contrast, embeddings trained on non-OEIS texts mainly just return additional numbers in the same range.

5 Conclusion

We introduce integer embeddings trained on the Online Encyclopedia of Integer Sequences (OEIS) and probe them for mathematical knowledge along with the integer representations found in the vocabulary of pre-trained English word embeddings. We find the OEIS embeddings promising for mathematical applications, as they are able to capture some meaningful mathematical regularities.

Acknowledgements

We thank the DiDi Labs NLP Group members for helpful discussion, and the anonymous reviewers for their valuable feedback.

References

Yonatan Belinkov, Nadir Durrani, Fahim Dalvi, Hassan Sajjad, and James Glass. 2017. What do neural machine translation models learn about morphology? In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 861–872, Vancouver, Canada. Association for Computational Linguistics.

Yonatan Belinkov and James Glass. 2019. Analysis methods in neural language processing: A survey. *Transactions of the Association for Computational Linguistics*, 7:49–72.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

Thomas Hofmann. 1999. Probabilistic latent semantic analysis. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*.

OEIS Foundation Inc. The on-line encyclopedia of integer sequences. Published electronically at `http://oeis.org`. Retrieved on 2020-07-22.

Arne Köhn. 2015. What's in an embedding? Analyzing word embeddings through multilingual evaluation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2067–2073, Lisbon, Portugal. Association for Computational Linguistics.

Klaus Korossy. 1998. Solvability and uniqueness of linear-recursive number sequence tasks. *Methods of Psychological Research Online*, 3(1):43–68.

Omer Levy and Yoav Goldberg. 2014. Dependency-based word embeddings. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 302–308, Baltimore, Maryland. Association for Computational Linguistics.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013a. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. 2013b. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 746–751, Atlanta, Georgia. Association for Computational Linguistics.

Aakanksha Naik, Abhilasha Ravichander, Carolyn Rose, and Eduard Hovy. 2019. Exploring numeracy in word embeddings. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3374–3380, Florence, Italy. Association for Computational Linguistics.

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.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

Neil J. A. Sloane. 2003. The on-line encyclopedia of integer sequences. *Notices of the American Mathematical Society*, 50(8).

Avijit Thawani, Jay Pujara, Filip Ilievski, and Pedro Szekely. 2021. Representing numbers in NLP: a survey and a vision. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 644–656, Online. Association for Computational Linguistics.

Ronald J Williams and Jing Peng. 1990. An efficient gradient-based algorithm for on-line training of recurrent network trajectories. *Neural computation*, 2(4):490–501.

A Data sources

OEIS sequence dump was downloaded from the encyclopedia website: `https://oeis.org/stripped.gz` (accessed on 2020-07-22).

Sequence completion problems were collected from five test preparation websites: `Nibcode`, `Syvum`, `12MinPrep`, `IQTestPrep`, `Hitbullseye`.

Mathematical analogy problems were collected from four test practice websites: `LearnFrenzy`, `Examsbook`, `toppr`, `AllIndiaExams`.

B Pre-trained embeddings

Based on the empirical analysis of numeracy in word embeddings conducted by Naik et al. (2019), we choose three embedding models pre-trained on English text corpora:

- **GloVe–840B–300D:** 300-dimensional GloVe vectors trained on 840B tokens from the Common Crawl corpus. Downloaded from `https://nlp.stanford.edu/projects/glove`
- **SkipGram–BoW–5:** 300-dimensional SkipGram bag-of-words embeddings with window size 5. Downloaded from `https://levyomer.wordpress.com/2014/04/25/dependency-based-word-embeddings`

- **FastText–Wiki:** 300-dimensional FastText vectors with subword information, trained on the 16B tokens from Wikipedia 2017, UMBC webbase corpus and `statmt.org` news dataset. Downloaded from https://fasttext.cc/docs/en/english-vectors.html

C LSTM implementation details

We implement our two-layer LSTM language model using the PyTorch toolkit. Our model uses truncated BPTT (Williams and Peng, 1990) and is trained for 40 epochs. Hyperparameters: hidden size 200, dynamically annealed learning rate starting at 20, gradients clipped at 0.25.