The Chinese Remainder Theorem for Compact, Task-Precise, Efficient and Secure Word Embeddings

Patricia Thaine
Department of Computer Science
University of Toronto
pthaine@cs.toronto.edu

Gerald Penn
Department of Computer Science
University of Toronto
gpenn@cs.toronto.edu

Abstract

The growing availability of powerful mobile devices and other edge devices, together with increasing regulatory and security concerns about the exchange of personal information across networks of these devices has challenged the Computational Linguistics community to develop methods that are at once fast, space-efficient, accurate and amenable to secure encoding schemes such as homomorphic encryption. Inspired by recent work that restricts floating point precision to speed up neural network training in hardware-based SIMD, we have developed a method for compressing word vector embeddings into integers using the Chinese Reminder Theorem that speeds up addition by up to 48.27% and at the same time compresses GloVe word embedding libraries by up to 25.86%. We explore the practicality of this simple approach by investigating the trade-off between precision and performance in two NLP tasks: compositional semantic relatedness and opinion target sentiment classification. We find that in both tasks, lowering floating point number precision results in negligible changes to performance.

1 Introduction

In recent years, NLP models, particularly language models, have come under increasing scrutiny for their potential privacy leaks (e.g., Carlini et al. (2018)). One answer has been to push NLP models onto edge devices, such as mobile phones and browsers, for on-device inference using differentially private federated learning (e.g., Yang et al. (2018)). Edge computing, in turn, requires smaller and faster models, such as DistillBERT (Sanh et al., 2019) or mobileBERT (Sun et al., 2020), which theoretically improves the viability of using BERT embeddings on these devices.

What has yet to take place, however, is careful numerical analysis of word embeddings at varying precisions for different NLP tasks. Numerical analysis can inform potential alternatives to defining the representations of word embeddings themselves. We must carefully orchestrate a balance among the size of these representations, their security, the algebra of operations that they enable, and, because many edge devices are resource-restricted relative to the GPU clusters that we do research on, the efficiency and power consumption of those operations. In particular, feature/basic phones are growing in popularity in developing nations[1][2], often with the possibility of expanding their memory to about 16GB with a MicroSD. Many privacy-enabled language models, in particular, are simply out of reach for these low-resource devices.

No work to date has proposed using the Chinese Remainder Theorem (CRT) to create software-based compressed word embeddings, adapted to Single Instruction Multiple Data (SIMD). We propose here a CRT-based method which speeds up addition by up to 48.27% and compresses GloVe (Pennington et al., 2014) word embedding libraries by up to 25.86% (up to 68.68% of the original full-precision library), depending on the precision selected. We also explore different levels of numerical precisions for a representative task, as well as more abstractly by analysing the absolute error resulting from adding and multiplying truncated and rounded values.

Related Work. A number of recent papers have explored limiting input precision in order to speed up neural network training and inference. Zhang et al. (2018) and Ling et al. (2016) specifically

[1] https://qz.com/africa/1206462/smartphones-lost-market-share-to-feature-phones-in-africa-last-year/

[2] https://techpoint.africa/2019/03/18/drive-for-feature-phone-penetration-in-africa/

Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics, pages 3512–3521
April 19 - 23, 2021. ©2021 Association for Computational Linguistics

looked at the effects of limiting word embedding precision. Zhang et al. (2018) evaluate embeddings of different sizes combined with hardware-implemented SIMD for an adaptation of stochastic gradient descent that achieves a memory improvement of 2X and 1.2X faster training speed. Ling et al. (2016) demonstrate that using word embeddings made up of 8-bit fixed-point values performs just as well as other embeddings on word similarity, phrase similarity, and dependency parsing tasks. Tissier et al. (2019), on the other hand, has shown how to generate binary word embeddings that fit within a CPU's cache to increase computing speed. With limited effect on task accuracy, they manage to generate 256-bit embeddings which are 37.5× smaller than traditional 300-dimensional word embeddings.

While we have demonstrated that CRT can be used to increase the efficiency of word vector addition by transforming a vector into a single large number, it is usually used for the opposite purpose: to transform larger numbers into many smaller numbers to improve multiplication efficiency. For this purpose, CRT is integrated in certain homomorphic encryption libraries.

CRT has also been proposed as a method of encrypting multiple entries in a database (e.g., a_i) as one number, each decryptable by a secret modulus (e.g., m_i) (Yan, 2002). The security of such schemes are shaky and more complexity is usually added to guarantee security (Liu et al., 2014; Lin et al., 1992).

2 Background

2.1 Single Instruction Multiple Data

Single Instruction Multiple Data is a computer architecture classification described in Flynn (1966) as a single instruction acting simultaneously on multiple operands. Flynn (1966) points out that performance increases with the number of units. Prior to Flynn (1966), SIMD was discussed in Unger (1958); Slotnick et al. (1963); Crane and Githens (1965); Hellerman (1966). More recently, the Chinese Remainder Theorem was proposed as a method for attaining SIMD in order to optimize arithmetic for homomorphically encrypted values (Gentry et al., 2012; Smart and Vercauteren, 2014). We do not know of work that takes advantage of CRT-based SIMD for NLP, despite the repetitive tasks performed within certain algorithms.

2.2 Chinese Remainder Theorem

Theorem 2.1 (The Chinese Remainder Theorem)**.** *Let $m_1, m_2, ..., m_r$ be pairwise relatively prime positive integers. Then the system of congruences*

$$X \equiv a_1 (\text{mod } m_1)$$

$$X \equiv a_2 (\text{mod } m_2)$$

$$\dots$$

$$X \equiv a_r (\text{mod } m_r)$$

has a unique solution modulo $M = m_1 m_2 ... m_3$. (Rosen, 2000)

We use $\mathbf{a}$ to denote the vector of values $a_1, a_2, ..., a_r$ and $\mathbf{m}$ to denote the vector $m_1, m_2, ..., m_r$.

The Chinese Remainder Theorem can also be used to separate very large numbers into smaller chunks on which, in certain cases, arithmetic operations can be performed more efficiently (Rosen, 2000).

3 Algorithm: Vec2int

First, we must determine a minimum floating point precision (ϕ) the values within our word embeddings must have. Since the CRT only works with positive integers (and polynomials), we will multiply each value in the word embeddings (a_i) by 10^ϕ and truncate (or round) the result, then add the lowest possible integer value that can be found within our word embeddings (s). Before running the CRT algorithm, we must pre-compute a vector $\mathbf{m}$ of moduli, each of which are coprime and strictly greater than any possible value of a_i. $\mathbf{a}$ and $\mathbf{m}$ can then be input into the CRT algorithm in order to produce an integer X. Since we know the values of $\mathbf{m}$, we can easily perform a "reverse CRT" by calculating $a_i \leftarrow X \pmod{m_i}$.

Algorithm 1 Vector To Integer

```
procedure VEC2INT(a, φ, m, s)
    for i ← 1 to r do
        a_i ← truncate(a_i * 10^φ) + s
    end for
    X ← CRT(a, m)
    return X
end procedure
```

When two CRT-encoded vectors (say X_1 and X_2) are added together, their sum at index i can still be decoded by calculating $(X_1 + X_2) \pmod$

m_i), provided that either $X_1 + X_2 < m_i$, or $(X_1 + X_2) \pmod{2m_i}$, when $X_1 + X_2 \geq m_i$. The same property holds for multiplication; i.e., $(X_1 * X_2) \pmod{(m_i)^2}$, when $X_1 * X_2 \geq m_i$. In other words, this representation supports vector addition, multiplication and, in a more restricted range, subtraction, through executing those operations directly on integers.

Addition and multiplication are fundamental to most compositional semantics approaches, beginning with Mitchell and Lapata (2008, 2010). Addition is also central to measuring the relationship between word pairs, particularly when using the L1-norm as a metric, which is less sensitive to outliers than the L2-norm (Stratos, 2017). Word analogy is an example of such a task. Of course, the these operations can also be applied to measuring the relationship between sentence pairs and document pairs.

4 Reasoning about Precision using Numerical Analysis

One might assume that we have to perform extensive empirical analysis to determine the minimum floating point precision necessary for the inputs of a particular NLP algorithm. However, there are generalizations we can make based on concepts as simple as the absolute error and relative error of components of the operations, described in Heath (2018) as:

$$\text{Absolute error} = \text{approx. value} - \text{true value} \quad (1)$$

$$\text{Relative error} = \frac{\text{absolute error}}{\text{true value}} \quad (2)$$

For our operations, the vectors truncated or rounded at precision Φ within a dataset $|D_\Phi|$ are denoted $\mathbf{v_{\Phi i}}$, while the vectors of the original dataset are denoted $\mathbf{v_i}$. If we want to refer to index k of vector $\mathbf{v_i}$, we write it as $\mathbf{v_i}[k]$, where $|\mathbf{v_1}|$ is the size of the embeddings we are working with. The average absolute error between the vectors of the two datasets of size $|D|$ is computed as follows:

$$\frac{\sum_i \sum_k |\mathbf{v_{\Phi i}}[k] - \mathbf{v_i}[k]|}{|D| * |\mathbf{v_1}|} \quad (3)$$

We calculate the average absolute error of addition as follows:

$$\frac{\sum_i \sum_j \sum_k |(\mathbf{v_{\Phi i}}[k] + \mathbf{v_{\Phi j}}[k]) - (\mathbf{v_i}[k] + \mathbf{v_j}[k])|}{\binom{|D|}{2} * |\mathbf{v_1}|} \quad (4)$$

5 Experimentation & Analysis

5.1 Arithmetic

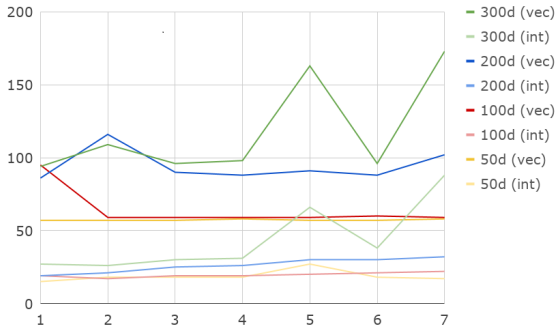

Figure 1: Milliseconds (y-axis) it takes to run 100000 element-wise additions over vectors of (*d) dimensions at ϕ precision (x-axis).

We have tested the efficiency of adding and multiplying integer representations of word embeddings compared to adding and multiplying their original vectors. On a 1.9GHz Intel i7-8650U with a 2.11 GHz burst rate, 1024MB L2 cache and 8192MB L3 cache, 100K additions (Figure 1) generally take 3x longer to perform on vectors than their integer encodings, regardless of precision and dimensionality, while the results for multiplication (Figure 2) are profound and negative at larger precisions and dimensionalities.

We may then theoretically expect the CRT-encoding of vectors as integers to result in a significant performance gain in the case of addition, but for the most part a significant performance loss in the case of multiplication. Nevertheless, in comparing the various pairs of rows in Tables 1 and 2, we notice that the computational gains made from adding CRT representations rather than word vec-

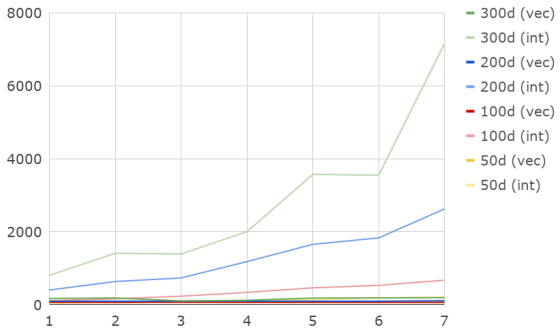

Figure 2: Milliseconds (y-axis) it takes to run 100000 element-wise multiplications over vectors of (*d) dimensions as ϕ precision (x-axis).

	100000	1000000	10000000
50d1(int)	**15**	**156**	**1548**
50d1(vec)	57	572	5764
50d2(int)	**18**	**172**	**1716**
50d2(vec)	57	572	5711
50d3(int)	**18**	**174**	**1746**
50d3(vec)	57	570	5712
50d4(int)	**18**	**184**	**1835**
50d4(vec)	58	570	5718
50d5(int)	**27**	**174**	**1719**
50d5(vec)	57	572	5712

Table 1: Milliseconds it takes to run 100000, 1000000, and 10000000 many additions over vectors of (*d) dimensions at (d*) precision.

	100000	1000000	10000000
50d1(int)	**37**	**371**	**3711**
50d1(vec)	57	574	5701
50d2(int)	**54**	**539**	**5385**
50d2(vec)	57	571	5690
50d3(int)	72	719	7203
50d3(vec)	**60**	**571**	**5698**
50d4(int)	102	1026	10261
50d4(vec)	**57**	**570**	**5698**
50d5(int)	131	1310	13107
50d5(vec)	**57**	**571**	**5700**

Table 2: Milliseconds it takes to run 100000, 1000000, and 10000000 many element-wise multiplication over vectors of (*d) dimensions at (d*) precision.

tors can considerably outweigh the losses from multiplying integer representations when the number of additions is greater or equal to the number of multiplications. In practice, as illustrated in several use cases below, the number of additions can be far greater in many important tasks. Recall also the findings of Ling et al. (2016), who did not observe a significant decrease in performance when using an 8-bit fixed-point value for word embeddings in word and phrase similarity and dependency parsing tasks. This implies that there is some room for compromise on precision practical NLP tasks as well.

5.2 Use Case 1: Compositional Semantics

A prime example of the significant performance improvements brought on by integer addition of word embeddings in the area of compositional semantics can be found in Salton and McGill (1986). It was demonstrated that vector addition is more effective than other proposed unsupervised compositional models (multiplication (Mitchell and Lapata, 2010), tensor product with convolution (Widdows and Ferraro, 2008), and dilation (Mitchell and Lapata, 2010)) for determining semantic relatedness between bigrams and other bigrams or unigrams (Asaadi et al., 2019).

Dataset and previous results. In Asaadi et al. (2019), the authors introduce BIRD, a bigram relatedness dataset created using the Best-Worst Scaling annotation method. To accomplish this task, annotators are provided with n sample, where n is often 4, and are asked which of the samples best represent the a given property and which one represents it the worst (Kiritchenko and Mohammad, 2016, 2017). Asaadi et al. (2019) compute bigram semantic relatedness using three different kinds of word embeddings; namely, pre-trained GloVe vectors[3], pre-trained fastText word embeddings[4], and word-context co-occurrence vectors extracted from a corpus of university websites (Turney et al., 2011)[5]. Specifically, the authors compute the relatedness score for the vectors representing the term pair AB-X, where AB is a bigram and X can be either a bigram or a unigram. The results of four unsupervised compositional models were compared:

- Weighted addition (Salton and McGill, 1986);
- Multiplication (Mitchell and Lapata, 2010);
- Tensor product with convolution (Widdows and Ferraro, 2008);
- Dilation (Mitchell and Lapata, 2010).

They then used Pearson correlation to compare the semantic relatedness scores, computed using these unsupervised compositional models, with the gold-standard in the BiRD. Addition turns out to be the composition method that results in the highest Pearson correlation scores.

Integrating vec2int The task introduced by Asaadi et al. (2019) is a prime candidate for using vec2int to speed up computations and reduce space: vast numbers of additions need to be performed in order to compute bigram relatedness in large datasets. These datasets often occupy a large amount of space and would also benefit from a compression method that can support addition in the compressed domain. The catch is that, in

[3]Originally obtained at: https://nlp.stanford.edu/projects/glove

[4]Originally obtained at: https://fasttext.cc/docs/en/crawl-vectors.html

[5]Code and data can be found at: https://github.com/sasaadi/BiRD

order to convert from a vector to an integer, we must limit the precision of each of the entries of the vector. Clearly, this can lead to concerns regarding the levels of accuracy that can be guaranteed by functions taking these precision-limited vectors as inputs. For our calculations, we forego any comparisons with floating point precision below 8 digits (other than the original number of digits[6]), since we want to maximize computational efficiency and minimize storage size. Recall that ϕ is the number of digits we keep after the decimal point.

Figure 3 and Figure 4 display the effects of truncating and rounding floating point numbers at various precisions then converting them to integers for determining bigram relatedness through the composition of fastText and GloVe vectors, respectively[7]. Interestingly, even sticking to a floating point precision of 1 or 2 can lead to results which are just as good as with the original precision.

Figure 5 shows the average absolute error (see Equation 3) and Figure 6, the average relative error (see Equation 4), for each possible sum of two vectors within the term context, fastText, and GloVe datasets. We can see that an increase in average absolute and relative errors lines up with the decrease in correlation score, and so we can determine at which point bigram relatedness would start to falter without computing Pearson or Spearman correlations. In fact, it appears to suffice only to calculate the average (Figure 9) and relative (Figure 10) errors of the truncated and rounded word vectors themselves (see Equations 1 and 2, respectively).

For interest, we show the average absolute error (Figure 7) and average relative error (Figure 8) for composition through multiplying fastText vectors at various precisions. fastText exhibits the most dramatic difference in performance between composition by addition and composition by multiplication at lower precisions.

[6]The largest precision within the term context vector is $\phi = 20$, within fastText is $\phi = 12$, and within GloVe is $\phi = 8$.

[7]We would like to thank Shima Asaadi, Saif M. Mohammad, and Svetlana Kiritchenko for providing the code and data they used for the their Big BiRD paper. The correlation scores for term context vectors are nearly the same as those for GloVe at untruncated precision, but remain stable down to $\phi = 2$.

5.3 Compression

We test the utility of the CRT as a method of compressing word embeddings. The results are shown in Table 3, reaching up to a 25.85% space reduction for GloVe vectors. We also calculate an upper bound of how large a dataset of word embeddings can get when compressed using CRT representations using the same relative primes $\mathbf{m}$ as the ones used to calculate the results displayed on Tables 3. Calculating the upper bound essentially comes down to associating each word in the dataset to the largest possible number N given $\mathbf{m}$:

$$N = \prod_i m_i \tag{5}$$

To demonstrate that, no matter what the vectors were the compression would work, we show the upper bounds compared to the size of the original dataset of 50-dimensional GloVe embeddings and the true CRT-compressed dataset of those embeddings on Table 4.

5.4 Use Case 2: Data Preprocessing for Private and Secure Computing

Another potential use case is if one wants to convert a text to its corresponding word embeddings directly on device, before being sent for cloud processing through homomorphic encryption or secure multiparty computation algorithms, which compute on precision-limited obfuscated data.

Arora et al. (2020) shows that pre-trained embeddings perform within 5-10% accuracy of benchmark tasks (NER, Sentiment analysis, GLUE) compared to contextual embeddings (e.g., BERT) and can even regularly match their performance when using industrial-level data.

Directly converting the words to their respective word embeddings on device is particularly useful for those algorithms, where table lookup over large amounts of data can be a fairly expensive operation. Not only can CRT representations compress the word embeddings to save valuable space on edge devices, but they can be selectively decompressed, unlike zip files, as needed. If you want to only convert the words in a user query to embeddings, there is no need to decompress the entire dataset!

We compare the results of CNN-based sentence-level sentiment analysis using GloVe vectors as input (Kim, 2014)[8] with those using

[8]https://github.com/yoonkim/CNN_

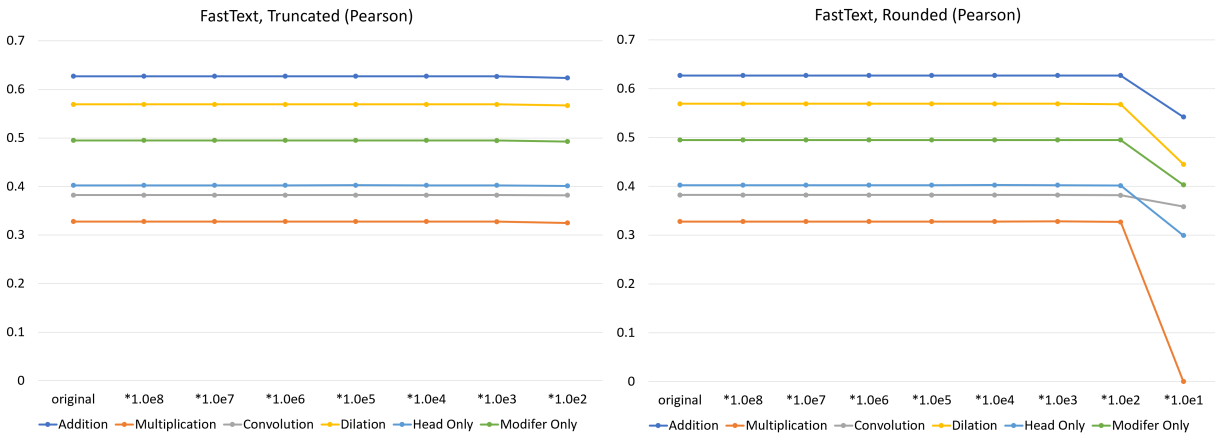

Figure 3: Pearson correlation results for determining bigram relatedness using addition of FastText vectors. $*1.0e\phi$ means we are approximating at precision ϕ. The trends are analogus with Spearman correlations.

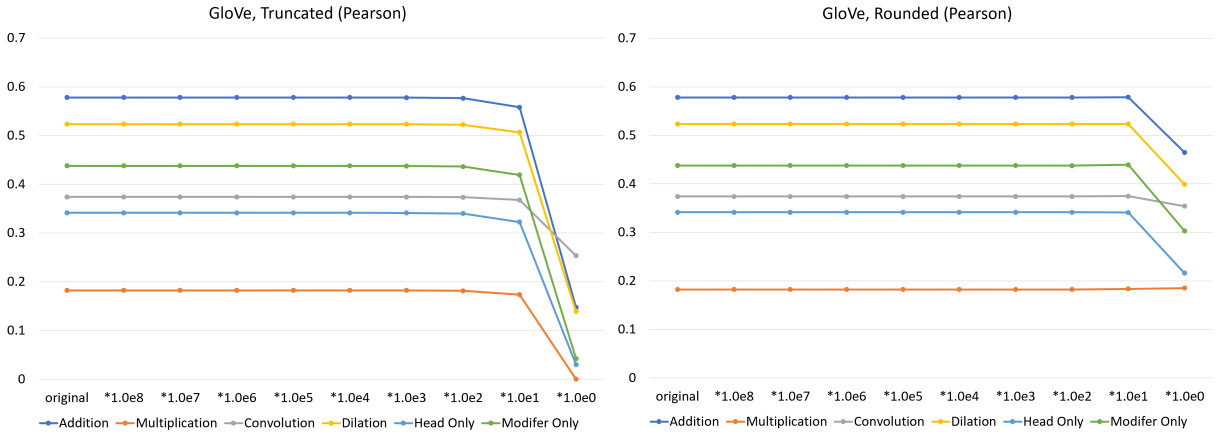

Figure 4: Pearson correlation results for determining bigram relatedness using addition of GloVe vectors. $*1.0e\phi$ means we are approximating at precision ϕ. The trends are analogus with Spearman correlations.

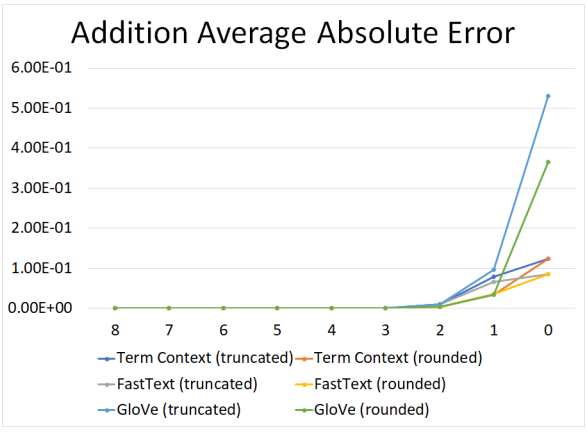

Figure 5: Average Absolute Error (y-axis) of Addition Composition at various precisions ϕ (x-axis).

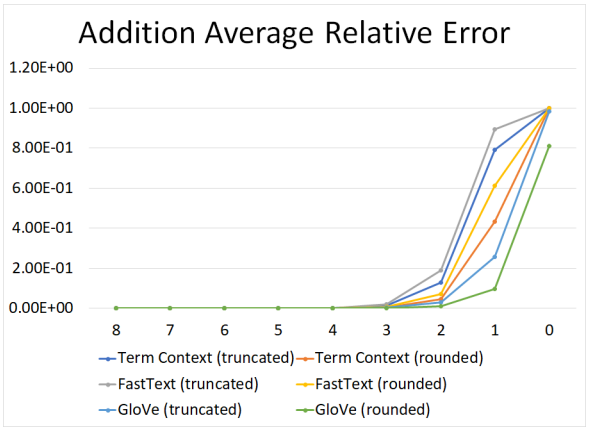

Figure 6: Average Relative Error (y-axis) of Addition Composition at various precisions ϕ (x-axis).

Φ	50d (vec)	50d (int)	% comp.	300d (vec)	300d (int)	% comp.
1	72370294	53660860	25.85	409761382	362331318	11.58
2	93553333	70375007	24.78	541234251	421515395	22.12
3	113672109	89863047	20.95	662404899	522909838	21.06
4	133684216	109772868	17.89	782523349	640203015	18.19
5	153685462	129763285	15.57	902535343	759975428	15.80
6	173685641	149762224	13.77	1022536755	879944768	13.94
7	193685703	169762728	12.35	1142537169	999941457	12.48
original	171350515	-	-	1037965801	-	-
zipped	69182687	-	-	394362421	-	-

Table 3: Size (in bytes) of 50-dimensional (50d) and 300-dimensional (300d) GloVe dataset (400k terms) at various precisions, stored as integer vectors (vec) and their CRT representations (int). % comp. shows the level of compression obtained.

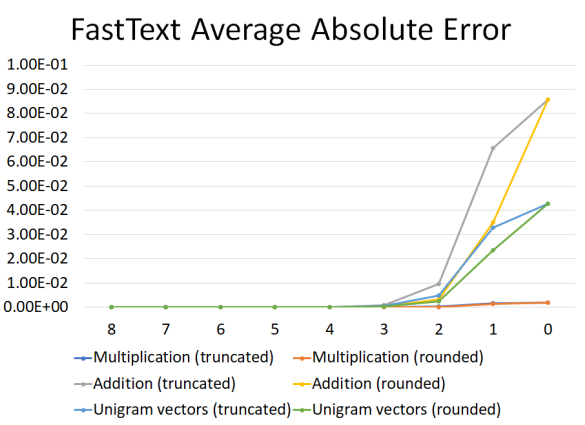

Figure 7: Average Absolute Error (y-axis) for fastText at various precisions ϕ (x-axis).

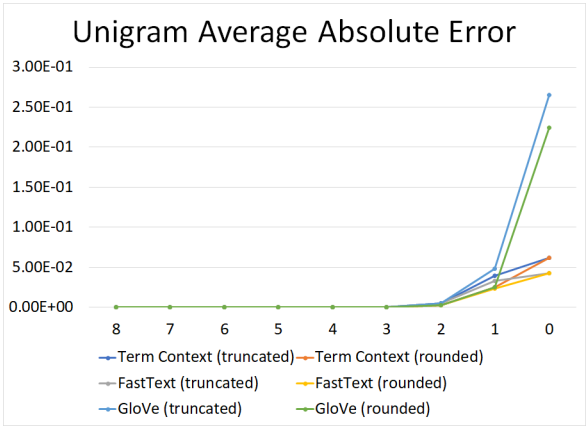

Figure 9: Average Absolute Error (y-axis) for unigram vectors at various precisions ϕ (x-axis).

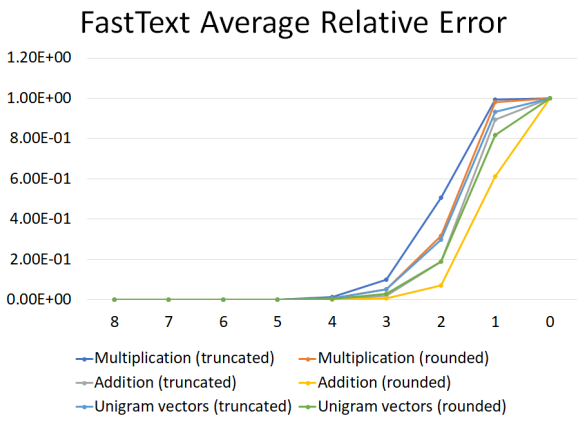

Figure 8: Average Relative Error (y-axis) for fastText at various precisions ϕ (x-axis).

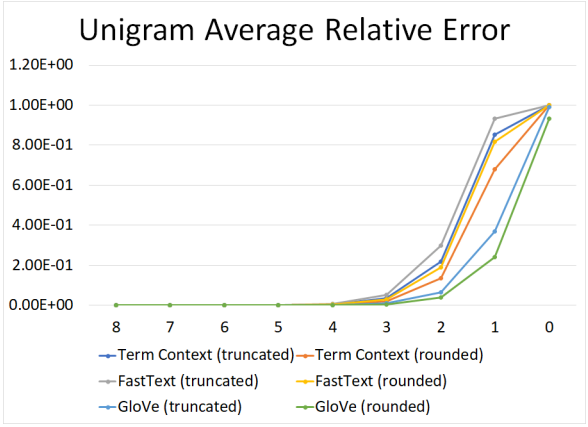

Figure 10: Average Relative Error (y-axis) for unigram vectors at various precisions ϕ (x-axis).

Φ	50d (vec)	50d (int)	50d Up. Bd.
1	72370294	53660860	53756606
2	93553333	70375007	138556818
3	113672109	89863047	90156697
4	133684216	109772868	110156747
5	153685462	129763285	130156797
6	173685641	149762224	150156847
7	193685703	169762728	170156897

Table 4: Size in bytes of 50-dimensional GloVe embeddings dataset, their CRT-compression and the upper bound size of that CRT-compression for any possible 50-dimensional vectors, at various precisions Φ.

BERT. Kim (2014) is the same work that Arora et al. (2020) used to compare with BERT over 6 different datasets. Of those datasets, we choose to analyze the variation in performance on the MR dataset (movie reviews – one sentence per review) Pang and Lee (2005) and on the opinion polarity detection subtask of the MPQA dataset (Wiebe et al., 2005).

In addition to comparing BERT-based sentiment analysis results with CNNs that take full-precision 300-dimensional GloVe vectors as inputs (which Arora et al. (2020) does), we verify the feasibility of this use case, by conducting experiments with varying precisions of 50-dimensional and 300-dimensional GloVe embeddings. Table 5 shows that sentence-based sentiment analysis with GloVe embeddings remains almost unchanged even at $\Phi = 1$ on the MR dataset, on which Arora et al. (2020) shows that BERT outperforms CNNs by 6.9. BERT therefore outperforms the CNN with 50d GloVe embeddings at precision $\phi = 1$ (which achieves a performance of 77.3) by 8.9. However, GloVe often matches BERT in this task when the BERT embeddings are trained on 16 times less data. Arora et al. (2020) also show that BERT outperforms GloVe on the MPQA dataset by a measly 0.9. We determine that BERT outperforms the CNN with 50d GloVe embeddings at precision $\phi = 1$ (which achieves a performance of 88.46) by only 1.14.

As is mentioned in Arora et al. (2020), it takes "440MB to store $\text{BERT}_{\text{BASE}}$ parameters, and on the order of 5-10 GB to store activations[, while p]retrained non-contextual embeddings (e.g., GloVe) require $O(nd)$ to store a n-by-d e-bedding matrix (e.g., 480 MB to store

`sentence`

Φ	50d	300d
-1	0.4908	0.5001
0	0.7571	0.7320
1	0.7730	0.7902
2	0.7745	0.7900
5	0.7730	-
full	0.769	0.7942

Table 5: Performance when varying the precision Φ of input GloVe embeddings to sentence-level sentiment analysis using a CNN.

a 400k by 300 GloVe embedding matrix)." Our method manages to reduce this embedding matrix size by 25%. MobileBERT uncased, which is made up of 23.21% of the number of parameters of $\text{BERT}_{\text{BASE}}$, ends up with a storage size of 139 MB (compare with 421 MB for $\text{BERT}_{\text{BASE}}$ uncased). That resource-constrained model is still 2.62x larger than our 53.66MB compressed GloVe embedding matrix, and without the benefit of our significant performance gain. Indeed it is difficult to imagine how a method that manipulates 512-bit vectors could compete with adding integers. MobileBERT's performance scores are within a couple of percentage points from those of $\text{BERT}_{\text{BASE}}$.

6 Discussion and Future Work

These preliminary results suggest that the vec2int algorithm would be an efficient way of encoding word vectors for specific NLP tasks, namely those which would benefit from arithmetic-supporting vector compression (which `tar` and `zip` are not). They also suggest that analysis of average relative and absolute error can be used to tune these representations. We have yet to test the effectiveness of using integer representations of word2vec (Mikolov et al., 2013) and BERT (Devlin et al., 2018), but expect the outcomes to be similar to the results for term context vectors, fastText, and GloVe. It would also be particularly interesting to see what kind of performance improvements can be obtained by applying the CRT to sentence- as well as document-level embeddings.

Relevant to privacy concerns, some constraints on homomorphic encryption schemes include limiting the number of possible multiplications and also limiting the precision of the values being computed upon. The same sort of numerical precision analysis and computational limitations that we have done in this paper can inform how we as

NLP scientists think of how our methods are integrable into certain privacy technology. Average relative and absolute error might be useful for creating better encrypted NLP algorithms using homomorphic encryption, since many homomorphic encryption schemes tend to require integer inputs or limited-precision inputs.

Acknowledgements

We are grateful to Professors Christina Christara and Sergey Gorbunov for their feedback, as well as to Shima Asaadi, Dr. Saif Mohammad, and Dr. Svetlana Kiritchenko for making their code available and for their responsiveness to our questions. This work was supported by an RBC Graduate Fellowship, the BRAIN Ontario Research Fund, and the Vector Institute.

References

Simran Arora, Avner May, Jian Zhang, and Christopher Ré. 2020. Contextual embeddings: When are they worth it? *arXiv preprint arXiv:2005.09117*.

Shima Asaadi, Saif Mohammad, and Svetlana Kiritchenko. 2019. Big bird: A large, fine-grained, bigram relatedness dataset for examining semantic composition. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 505–516.

Nicholas Carlini, Chang Liu, Jernej Kos, Úlfar Erlingsson, and Dawn Song. 2018. The secret sharer: Measuring unintended neural network memorization & extracting secrets. *arXiv preprint arXiv:1802.08232*.

Bently A Crane and JA Githens. 1965. Bulk processing in distributed logic memory. *IEEE Transactions on Electronic Computers*, (2):186–196.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Michael J Flynn. 1966. Very high-speed computing systems. *Proceedings of the IEEE*, 54(12):1901–1909.

Craig Gentry, Shai Halevi, and Nigel P Smart. 2012. Fully homomorphic encryption with polylog overhead. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 465–482. Springer.

Michael T Heath. 2018. *Scientific computing: an introductory survey*, volume 80. SIAM.

Herbert Hellerman. 1966. Parallel processing of algebraic expressions. *IEEE Transactions on Electronic Computers*, (1):82–91.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*.

Svetlana Kiritchenko and Saif M. Mohammad. 2016. Capturing reliable fine-grained sentiment associations by crowdsourcing and best–worst scaling. In *Proceedings of The 15th Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL)*, San Diego, California.

Svetlana Kiritchenko and Saif M Mohammad. 2017. Best-worst scaling more reliable than rating scales: A case study on sentiment intensity annotation. *arXiv preprint arXiv:1712.01765*.

Chu-Hsing Lin, Chin-Chen Chang, and Richard C. T. Lee. 1992. A record-oriented cryptosystem for database sharing. *The Computer Journal*, 35(6):658–660.

Shaoshi Ling, Yangqiu Song, and Dan Roth. 2016. Word embeddings with limited memory. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 387–392.

Yanjun Liu, Chin-Chen Chang, and Shih-Chang Chang. 2014. An access control mechanism based on the generalized aryabhata remainder theorem. *IJ Network Security*, 16(1):58–64.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

Jeff Mitchell and Mirella Lapata. 2008. Vector-based models of semantic composition. *proceedings of ACL-08: HLT*, pages 236–244.

Jeff Mitchell and Mirella Lapata. 2010. Composition in distributional models of semantics. *Cognitive science*, 34(8):1388–1429.

Bo Pang and Lillian Lee. 2005. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. *arXiv preprint cs/0506075*.

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.

Kenneth H Rosen. 2000. *Elementary number theory and its applications*. Addison Wesley Longman, Inc.

Gerard Salton and Michael J McGill. 1986. Introduction to modern information retrieval.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.

DL Slotnick, WC Borck, and RC McReynolds. 1963. The solomon computer–a preliminary report. In *Proc. 1962 Workshop on Computer Organization*, page 66. Washington, DC, Spartan.

Nigel P Smart and Frederik Vercauteren. 2014. Fully homomorphic simd operations. *Designs, codes and cryptography*, 71(1):57–81.

Karl Stratos. 2017. Reconstruction of word embeddings from sub-word parameters. *arXiv preprint arXiv:1707.06957*.

Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. 2020. Mobilebert: a compact task-agnostic bert for resource-limited devices. *arXiv preprint arXiv:2004.02984*.

Julien Tissier, Christophe Gravier, and Amaury Habrard. 2019. Near-lossless binarization of word embeddings. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7104–7111.

Peter D Turney, Yair Neuman, Dan Assaf, and Yohai Cohen. 2011. Literal and metaphorical sense identification through concrete and abstract context. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 680–690. Association for Computational Linguistics.

Stephen H Unger. 1958. A computer oriented toward spatial problems. *Proceedings of the IRE*, 46(10):1744–1750.

Dominic Widdows and Kathleen Ferraro. 2008. Semantic vectors: a scalable open source package and online technology management application. In *LREC*.

Janyce Wiebe, Theresa Wilson, and Claire Cardie. 2005. Annotating expressions of opinions and emotions in language. *Language resources and evaluation*, 39(2-3):165–210.

Song Y Yan. 2002. *Number theory for computing*. Springer Science & Business Media.

Timothy Yang, Galen Andrew, Hubert Eichner, Haicheng Sun, Wei Li, Nicholas Kong, Daniel Ramage, and Françoise Beaufays. 2018. Applied federated learning: Improving google keyboard query suggestions. *arXiv preprint arXiv:1812.02903*.

Jian Zhang, Jiyan Yang, and Hector Yuen. 2018. Training with low-precision embedding tables.