Hash2Vec: Feature Hashing for Word Embeddings

Luis Argerich, Matias J. Cano, and Joaquin Torre Zaffaroni

Facultad de Ingeniería, Universidad de Buenos Aires,
Ciudad Autónoma de Buenos Aires, Argentina.
lrargerich@gmail.com

Abstract. In this paper we propose the application of feature hashing to create word embeddings for natural language processing. Feature hashing has been used successfully to create document vectors in related tasks like document classification. In this work we show that feature hashing can be applied to obtain word embeddings in linear time with the size of the data. The results show that this algorithm, that does not need training, is able to capture the semantic meaning of words. We compare the results against GloVe showing that they are similar. As far as we know this is the first application of feature hashing to the word embeddings problem and the results indicate this is a scalable technique with practical results for NLP applications.

Keywords: feature hashing, natural language processing, word embedding

1 Introduction

The main objective of word embeddings is to form a vector space of words that makes "sense". Usually, this means that semantically similar words are together. One use for these language models is to estimate the probability of an n-gram being correct. [1] suggested that word embeddings can also be used to create reverse-dictionaries, in which one writes the definition of a word and the algorithm suggests a concept that fits the definition. Even more, it is possible to create bilingual reverse dictionaries which can be quite useful in translation tasks. In recent works, several interesting properties of the resulting vector spaces were found [2]. There are many other applications of word embeddings [3] which make the topic a very active area of research in NLP.

One way to create word embeddings is by using the bag of words (BOW) model where the word co-occurrence matrix is calculated. In this matrix, each row represents a unique word so that the i,j-element is the amount of times word j has co-occurred with word i. This matrix can become huge in the order of millions by millions, making its use difficult in any application.

As a result, modern models (GloVe [3], *word2vec* [4]) learn to represent words with a fixed reduced dimensionality.

A simple technique for dimensionality reduction is Feature Hashing [5], also known as the *hashing trick*. The idea is to apply a hashing function to each feature of a high dimensional vector to determine a new dimension for the feature in a reduced space. Feature hashing has been used successfully to reduce the dimensionality of the BOW model for texts [5]. [6] used feature hashing to classify mail as *spam* or *ham*.

To mitigate the effect of hash collisions [7] propose the use of a second hash function ξ that determines the sign of a feature.

It has been shown that Feature Hashing preserves the inner product between vectors and the error can be bounded. This is explained using the Johnson-Lindenstrauss lemma [8] [9] and showing that feature hashing is a particular case of a J-L projection where the projection matrix has exactly one $+1$ or -1 in each row.

Therefore if we apply the hashing trick to the word co-occurrence matrix we are able to obtain an embedding where the inner products between the embedded vectors accurately represent the inner products between the original vectors in the co-occurrence matrix. Our experiments confirm that the distortion between using the full vectors and this embedding is minimal. This means that vectors that are close in the original matrix will also be close in the embedded space.

Interestingly, embeddings can be constructed without the full-size matrix. Memory consumption is then reduced from $O(n^2)$ to $O(n \times k)$, where n is the size of the corpus in words and k is a fixed dimensionality that can be small (in the order of hundreds.)

2 Algorithm

The main formula for the algorithm can be seen in (1). It is a variation of the feature hashing equation shown in [5] with the addition of the second hashing function proposed by [7], as well as the domain-specific part which is the weight function for each co-occurrence.

$$\bar{w}_j^{(k)} = \sum_{w^{(c)} \in C_k; h(w^{(c)})=j} \xi(w^{(c)}) \sum_{i=1}^{n^{(k)}(w^{(c)})} f_i^{(k,c)} \tag{1}$$

$w(k)$ represents the k-th word, and $\bar{w}(k)$ represents the reduced vector for such word and therefore $\bar{w}_j(k)$ is the j-th component for the k-th word vector. C_k represents all the contexts of the k-th word and h is a hashing function as [5] shows. ξ is the additional hashing function such that $\xi : \text{String} \rightarrow \{-1, 1\}$ proposed by [7]. $n^{(k)}(x)$ represents the amount of times word x has appeared with word k, and finally, $f_i(k, c)$ is the aging (or weighting) for the word c according to word k in the i-th time they have appeared together.

Words are processed from the text in linear fashion and for each new word an embedding vector is created. A window of size k is defined to determine which words co-occur in the context of another. We iterate through the context

applying Feature Hashing to construct the embeddings. A simplified version of the algorithm can be seen in Algorithm 1.

Algorithm 1 Hash2Vec

Parameters: n the embedding size, k the context size, h hash function, ξ hash sign-function and f aging function.

```
words ← Dictionary()
for every word w in text do
    if w ∉ keys(words) then words[w] ← Array(n)
    for every context word cw with distance d do
        weight ← f(d)
        sign ← ξ(cw)
        words[w][h(cw)] ← words[w][h(cw)] + sign × weight
```

In practice we only need to keep track of the k past words and update both the embedding of the current one and the previous ones. This scheme allows the embeddings to be computed in a streamlined way with $O(n)$ complexity. For simplicity we assume that for each word the embedding is updated based on the words within the k-sized window.

The weight function f is a parameter and a deciding factor on the performance of the algorithm. For example, if $f(d) = 1\ \forall d$, then we are simply calculating a reduced version of the co-occurrence matrix. We obtained better results using f similar to a Gaussian distribution, i.e., $f(d) = e^{-\left(\frac{d}{\sigma}\right)^2}$.

One interesting property of Hash2Vec is that it always constructs the same embeddings for the same starting corpus, while *word2vec* and GloVe do not, as they either depend on the starting seed or use stochastic optimization.

2.1 Variations on the algorithm

In order to improve the quality of embeddings, we decided to try some variations on the basic idea and performed various preprocessing tasks before running Hash2Vec.

We preprocessed the corpus to remove stopwords. We used two criteria to select the words to filter: calculating a certain percentile, avoiding the words above it or using a stoplist of words to be removed. Both methods resulted in an improvement on the similarity tests.

We applied the algorithm proposed on [4] to adjoin phrases. This is very important because otherwise “New York”, “San Francisco”, etc. would not exist as a single token.

We also obtained modestly better results using *homogeneous sentence selection*, in which each sentence in an original text moves to a final text according to a uniform probability distribution, instead of simply truncating the text. Our hypothesis is that with better sampling, the context words are less biased to local articles (if using a source like Wikipedia), making the final vector better in

the sense that it was trained with many different articles and usages of the word. Moreover, our experiments revealed that this algorithm is sensible to polysemy. [10] explore this idea to construct different representations for the same word.

2.2 Applications to speed up other embedding algorithms

The algorithm that we propose constructs a lower-dimensional approximation of the word co-occurrence matrix, which can then be used directly or as the starting point for another algorithm, like GloVe. In the case of *word2vec*, [11] have shown that the SGNS model learns an explicit matrix factorization (EMF) and that very similar results can be obtained by using applying the EMF on the original matrix. The reduced-form matrix that Hash2Vec creates can be used instead of the original to obtain again very similar results without the memory footprint of an $O(n^2)$ matrix.

2.3 Streaming applications and parallelization

Since the embeddings are refined as text is processed, the algorithm is practical for streaming applications. The model can also adapt to changes in language, e.g. new words being used. Thus the embeddings can constantly be updated while being always usable, which is something GloVe cannot do.

By nature Hash2Vec is easily parallelizable since the embeddings can be updated from different portions of text in parallel and then just added up to construct the final vectors. This associative property makes it trivial for the algorithm to be ported to a model like MapReduce. Even more, if no regularization function is applied on the final vectors, embeddings trained in different contexts can be combined by adding the individual vectors of each word in both corpora.

3 Results

For benchmarking we compared our results with two commonly used datasets. The first is wordsim353 [12] and the second is from Amazon's Mechanical Turk, as used in [13]. Both datasets hold word pairs with similarity scores, representing human assigned similarity judgements. We then calculate the Spearman correlation between our results and the dataset's to obtain a metric on the quality of the embeddings. The similarity measure we used is cosine similarity, like the one used in [3].

Both graphs in Figure 1 show that Hash2Vec approximates the full vectors when the vector dimension increases. A very good approximation can be obtained with dimensions in the order of the hundreds, orders of magnitude smaller than full vectors. As expected, the co-ocurrence matrix appears to be a theoretical limit on the performance of Hash2Vec.

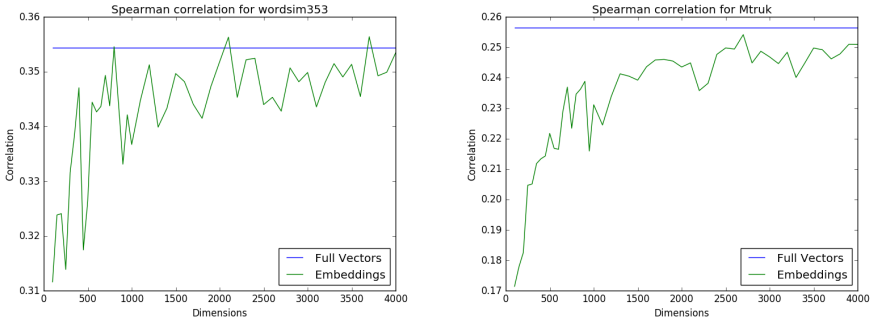

Fig. 1. Algorithm's base performance (by correlating it to the Wordsim353 and Amazon Mechanical Turk datasets) as the vector size increases.

3.1 Comparison with GloVe

We trained GloVe using the same corpus with $k = 15$ and $n = 600$ in both cases. Table 1 shows the 5 most similar words returned by GloVe and Hash2Vec for some queries.

Table 1. Most similar words comparison with GloVe

	computer	king	physics	italy	wounded	anglican
GloVe	computers	son	chemistry	germany	killed	churches
	software	i	mechanics	france	mortally	church
	systems	emperor	quantum	italian	soldiers	catholic
	hardware	ii	mathematics	greece	injured	communion
	game	kings	particle	spain	dead	orthodox
Hash2Vec	program	england	mathematics	france	killed	lutheran
	computers	iii	study	germany	injured	episcopal
	hardware	henry	chemistry	spain	captured	orthodox
	programs	charles	theory	switzerland	after	presbyterian
	game	james	astronomy	russia	defeated	communion

We observe the results of GloVe and Hash2Vec to be very similar in Table 1. Both models capture the semantic meaning of words and in some cases Hash2Vec seems to outperform Glove.

In Figure 2 we can compare both models by correlating the embeddings to the MTurk dataset. We can see that Hash2Vec performs worse than GloVe across all dimensions. This is expected as the original co-ocurrence matrix also performs worse than GloVe.

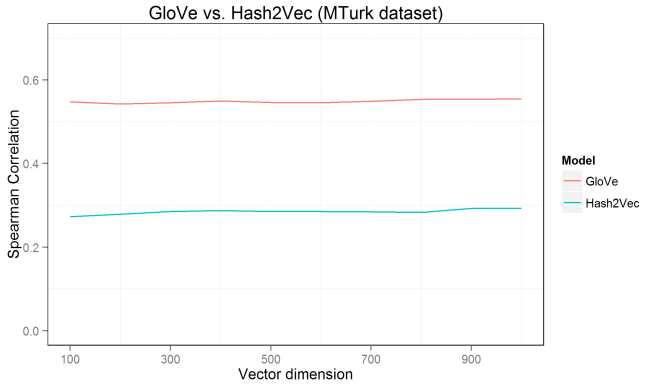

Fig. 2. Comparison of Hash2Vec and GloVe by correlating the resulting embeddings with MTurk data through increasing vector sizes. $k = 15$ and $f(d) = e^{-2(\frac{d}{k})^2}$.

3.2 Embeddings visualization

To understand how the algorithm behaves and to further observe the resulting vector space, we applied t-SNE [14], a widely used dimensionality reduction algorithm.

After training Hash2Vec on a 110-million-word corpus ($k = 5$ and $n = 600$), we applied t-SNE. Given that the algorithm groups together similar words, we should be able to see it in the graph. Since t-SNE is about $O(n^2)$ in memory, we only used the $15,000$ most common words. Zipf's law gives some guarantees to only keeping the most common words.

In Figure 3 we show an interesting sector extracted from the reduced matrix cartesian plot.

3.3 The myth behind word analogies and Hash2Vec for word analogies

Word analogies have been used to show the expressiveness power of a word embedding model [2]. GloVe and *word2vec* produce embeddings where a linear relationship exists between analog words. The most popular example is the analogy "prince is to princess like king is to ..." where the model should show that prince-princess $\equiv$ king-queen. In [3] the authors claim that models that capture this linearity have a superior understanding to others. In contrast, [16] show that the word analogy task is just a derivation of similarity. When asking the model "x is to y like z is to w" we are actually maximizing the dot product of $w(x+y-z)$ which is the same as maximizing $wx+wy-wz$ so the word analogy task is actually asking the model about words that are similar to x and y but different to z.

Word embeddings like GloVe or *word2vec* are not capable of capturing words that are different to a given word so they usually fail to find antonyms, this

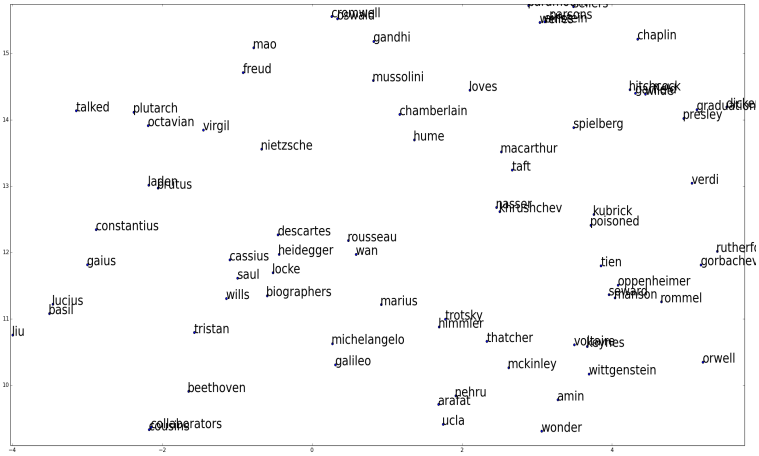

Fig. 3. The algorithm has grouped people, suggesting that the algorithm is able to recognize syntactical similarities by looking at the context words. We can also differentiate groups of philosophers, movie directors and famous Roman and Greek people.

means that the word analogy task is strongly dependent on the words that are similar to the positive words in the query.

Table 2. Word "linearity"

x	is to y	like z	is to ...
Paris	France	Moscow	**Russia**
Cow	Milk	Pig	**Meat**
Glass	Glasses	Horse	**Horses**
Nice	Ugly	Small	**Large**

Table 2 shows how Hash2Vec is able to solve some word analogy tasks. The vectors are not trained to preserve linearity in the word analogy task, but the original matrix (and therefore Hash2Vec) captures some of these properties.

4 Conclusions

In this work we detailed a very simple algorithm that is able to construct word embeddings in linear time. The algorithm does not require training and has minimal memory footprint. We showed the results of the algorithm to be comparable to GloVe [3] in the similar word and analogy tasks, which is one of the state of the art algorithms for word embeddings. While base Hash2Vec performs considerably worse than GloVe on the benchmarking datasets, the algorithm could be

used to approximate the word co-occurrence matrix to train models like GloVe or EMF with minimal memory consumption. It can also be used in streams or dynamic corpora.

The results show that feature hashing is a very powerful technique that can reduce the dimensionality of the full word vectors while capturing the semantic of each token in the vocabulary.

References

1. Hill, F., Cho, K., Korhonen, A., Bengio, Y.: Learning to understand phrases by embedding the dictionary. arXiv preprint (to be published)
2. Mikolov, T., Yih, W. T., Zweig, G.: Linguistic Regularities in Continuous Space Word Representations. In HLT-NAACL, 746–751 (2013)
3. Pennington, J., Socher, R., Manning, C. D. : Glove: Global Vectors for Word Representation. In EMNLP,Vol. 14, 1532–1543 (2014)
4. Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., Dean, J. : Distributed representations of words and phrases and their compositionality In Advances in neural information processing systems,3111–3119 (2013)
5. Shi, Q., Petterson, J., Dror, G., Langford, J., Strehl, A. L., Smola, A. J., Vishwanathan, S. V. N.: Hash kernels. In International Conference on Artificial Intelligence and Statistics, 496–503 (2009)
6. Attenberg, J., Weinberger, K., Dasgupta, A., Smola, A., Zinkevich, M.: Collaborative Email-Spam Filtering with the Hashing Trick. In Proceedings of the Sixth Conference on Email and Anti-Spam (2009)
7. Weinberger, K., Dasgupta, A., Langford, J., Smola, A., Attenberg, J.: Feature hashing for large scale multitask learning. In Proceedings of the 26th Annual International Conference on Machine Learning, 1113–1120 (2009)
8. Johnson, W. B., Lindenstrauss, J.: Extensions of Lipschitz mappings into a Hilbert space. Contemporary mathematics, 26,1, 189–206 (1984)
9. Achlioptas, D.: Database-friendly random projections: Johnson-Lindenstrauss with binary coins. Journal of computer and System Sciences, 66, 4, 671–687 (2003)
10. Huang, E. H., Socher, R., Manning, C. D., Ng, A. Y.: Improving word representations via global context and multiple word prototypes. In Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1, 873–882 (2012).
11. Li, Y., Xu, L., Tian, F., Jiang, L., Zhong, X., Chen, E.: Word embedding revisited: A new representation learning and explicit matrix factorization perspective. In Proceedings of the 24th International Joint Conference on Artificial Intelligence, 3650–3656 (2015)
12. Finkelstein, L., Gabrilovich, E., Matias, Y., Rivlin, E., Solan, Z., Wolfman, G., Ruppin, E.: Placing search in context: The concept revisited. In Proceedings of the 10th international conference on World Wide Web, 406–414 (2001)
13. Radinsky, K., Agichtein, E., Gabrilovich, E., Markovitch, S.: A word at a time: computing word relatedness using temporal semantic analysis. In Proceedings of the 20th international conference on World wide web, 337–346 (2011)
14. Van der Maaten, L., Hinton, G. Visualizing data using t-SNE. Journal of Machine Learning Research, 2579–2605 (2008)
15. Chris, D. P.: Another stemmer. In ACM SIGIR Forum, 24, 3, 56–61 (1990)
16. Levy, O., Goldberg, Y., Ramat-Gan, I.: Linguistic Regularities in Sparse and Explicit Word Representations. In CoNLL, 171–180 (2014)