Word Embedding Binarization with Semantic Information Preservation

Samarth Navali *, Praneet Prabhakar Sherki *, Ramesh Inturi, Vanraj Vala
Samsung R&D Institute, Bangalore, India
(s.pnavali,p.sherki,i.ramesh,vanraj.vala)@samsung.com

Abstract

With growing applications of Machine Learning in daily lives, Natural Language Processing (NLP) has emerged as a heavily researched area. Finding its applications in tasks ranging from simple Q/A chatbots to fully fledged conversational AI, NLP models are vital. Word and Sentence embeddings are one of the most common starting points of any NLP task. A word embedding represents a given word in a predefined vector-space while maintaining vector relations with similar or dis-similar entities. As such, different pretrained embedding such as Word2Vec, GloVe, FastText have been developed. These embeddings generated on millions of words are however very large in terms of size. Having embeddings with floating point precision also makes the downstream evaluation slow. In this paper we present a novel method to convert continuous embedding to its binary representation, thus reducing the overall size of the embedding while keeping the semantic and relational knowledge intact. This will facilitate an option of porting such big embedding onto devices where space is limited. We also present different approaches suitable for different downstream tasks based on the requirement of contextual and semantic information. Experiments have shown comparable result in downstream tasks with 7 to 15 times reduction in file size and about 5% change in evaluation parameters.

1 Introduction

Natural Language Processing (NLP) is quickly becoming one of the most important branch in Machine Learning field, with companies pouring millions to perfect their NLP engines. NLP is the cornerstone in the road to developing a perfect conversational AI. But NLP also finds its way in more trivial but important tasks in modern life, be it a simple chatbot, a Q/A site, Document Classifier. For most of these tasks, an embedding model or a pre-trained embedding is the perfect starting point. A word embedding is a collection of words, represented as vectors in a predefined space. Within this space the vectors follow all typical vector laws. Similar words are close together, dissimilar words are far off. Vector addition and subtraction are possible. Word Embeddings are very important for downstream tasks like document classification, query response, etc. Though Word Embeddings can be generated to be useful to a specific task on a specific dataset, there are embeddings available that are trained on a very large corpus, making them useful in variety of tasks and in turn making them generic. Word2Vec (Mikolov et al., 2013), Glove (Pennington et al., 2014), FastText (Bojanowski et al., 2016) are some of the best generic embeddings available that are trained on millions of vocabularies.

Though training on a large dataset makes the embedding useful, it also makes them very large in size. A 300-dimensional Word2Vec model trained on 3 million datapoints is aroung 3.4 GB. This large size makes the embedding very unportable, especially in low storage scenarios. As a result, many NLP operations are typically performed on the server. This raises issues like privacy concerns, as user data is continuously sent to server and processed there. The first step to solve this issue is to bring down the size of the embedding, small enough to carry on device such as mobile phones and devices where memory should be used judiciously. This will enable on-device training without depencency on server side.

*Equal Contribution

Proceedings of the 28th International Conference on Computational Linguistics, pages 1256–1265
Barcelona, Spain (Online), December 8-13, 2020

In this paper we propose two methods to binarize a pre-trained word embedding, while keeping all the semantic information intact. Our proposed methods are based on an auto-encoder architecture, which encodes continuous real vectors into its corresponding binary vector. The embedding generated by the proposed methods perform well on different types of tasks, with one approach performing good in tasks involving need of contextual and semantic data, while other performing well on the tasks where such data is not crucial (approaches are discussed Section 3). The embedding produced by both the proposed variations are a fraction of the original size with reduction of upto 7 to 15 times. We also demonstrate the performance of these embedding on different downstream tasks like similarity, classification and analogy.

2 Related Work

Word Embedding is starting point for many NLP applications, usually these embedding are very large in size and it requires a lot of time for subsequent computations.There is a need to reduce the memory footprint of these embedding and to make the computations faster. The idea is to binarize this embedding so that the amount of memory taken by a binary feature would be much lesser as compared to that of a float valued feature. In their research, (Yi et al., 2015) proposed a nonlinear methodology where the high dimensional data was embedded into a hamming cube while maintaining the structure of the original space. As this works on dimensions far greater than the dimensions that we deal with, this study did not prove to be useful for the scope of our research.

(Faruqui et al., 2015) proposed to binarize the real valued embedding by first increasing the dimensions of the original embedding to create a sparse matrix and then apply a binarization step to that embedding. Though this retains certain amount of semantic information, the produced vectors are not small. Fasttext.zip (Joulin et al., 2016) binarizes the embedding by clustering and concatenating the binary representations of 'k' closest centroids for each word. The resulting binary vectors produced cannot be used for generic tasks as this works only for document classification. Even though binary representations can make the computations faster, some NLP applications work only on real valued embedding (Ma and Hovy, 2016). In such applications reconstructing real embedding from binary representations becomes very vital.

The research of (Tissier et al., 2019) showed that we could binarize the word embedding using an autoencoder architecture. But the problem with this method was that it was not able to capture a lot of the semantic information. In this paper we have managed to beat the benchmarks from (Tissier et al., 2019) in most of the downstream tasks, which will be shown in Section 6. We have been able to achieve this feat with the inspiration of (Shen et al., 2019) where they binarize sentence embedding and use them for downstream tasks. We propose an approach of using these 2 studies as an inspiration and develop our binarized Word Embedding, which has done very well on benchmarks as show in Section 6.

3 Methodology

We propose a method that converts continuous word embedding into a binary form. Let x be the word, A_x refers to the continuous embedding for the respective word, B_x refers to the binary embedding for the word x, f refers to a function that will convert the continuous embedding, A_x to the binarized embedding of word x, B_x. Therefore,

$$f(A_x) \; = \; B_x \tag{1}$$

We aim to learn global function f that can be used to generate binary embedding of the highest quality. Quality in this context refers to how informative the binary embedding are as compared to the original ones, how much semantic data from the original it has captured and how much of a size reduction we have obtained as compared to the original embedding. Usability of these binary embedding in downstream tasks becomes an important aspect, as the main aim for us is to substitute the original embedding with the binarized one, and get an advantage by reducing the amount of memory used while still having comparable performance.

The function f can be implemented in many ways so as to achieve the end goal of obtaining binarized representations for continuous embedding, but not all of them will show the quality required for us to

achieve the end goal of substitution of the actual embedding. Some of these methods are hard thresholding, random projections and encoder-decoder models. Hard Thresholding is a method where each feature of the embedding will be converted to the binary form of 1s and 0s. Each feature of the embedding takes the value 1 if it is above a specific threshold value and value 0 if otherwise. The threshold can be selected in many ways, it can be zero, or a random value, or the mean value of the features. The problem with this method is that a huge chunk of the information will be lost directly and there is a good chance that these embedding will not be able to capture the semantic data from the original embedding. Random projection method randomly projects the already learnt continuous embedding onto a different vector space. Once we have projected the embedding, we use the method of Hard Thresholding to convert it into a binary representation. The advantage of this method is that we can alter the dimension of the word embedding to the required dimensions. The binary representations generated using the Hard Threshold and the Random Projection binarization techniques suffer from the limitation of retaining semantic information and the need to have an additional binarization step after the training of the embedding. Auto Encoder model can be used to overcome the above limitations by retaining the semantic information of the continuous embedding without the need of applying an individual binarization step after the training.

3.1 Auto Encoder Architecture

The above mentioned limitations of preserving semantic information and the need of an extra binarization step after training can be overcome by the use of an autoencoder architecture, which utilises a reconstruction loss so as to make sure that maximum information is carried from the continuous representations to the binarized representations. We propose the use of encoder decoder architecture, the encoder will be used to convert the continuous embedding into its binary representation, whereas the decoder will be used to convert the binary into a continuous embedding.

The encoder network comprises of a matrix operation followed by a binarization step. Let D be the dimension of the binary embedding, G be the dimension of the original embedding, a be a continuous embedding vector, b be a binarized embedding vector, for i = 0,1,2,3,4....D

$$b(i) = 1 \;\; if \;\; \sigma(W(i) \; . \; a + j(i)) \geq t \tag{2}$$

$$b(i) = 0 \;\; if \;\; \sigma(W(i) \; . \; a + j(i)) < t \tag{3}$$

Here, W is weight matrix, $j(i)$ is the i^{th} element of the bias term j, t is the threshold that has been selected to determine if the i^{th} feature of the vector is 1 or 0. During training we may use either a static or a dynamic value of t. For the static case we use the value of t to be zero, for the dynamic case we use a mean value of the embedding. All values which are equal to or greater than the value of t are mapped to 1 while all the values lower than t are mapped to 0. Comparison of these two binarization strategies are discussed in sections below. The details of the AutoEncoder architecture is as shown in Figure 1.

From the study of (Carreira-Perpinán and Raziperchikolaei, 2015; Dai and Le, 2015; Shen et al., 2018) encoder decoder framework, it has been shown that the linear decoders are the most favourable in the case for learning binary representations. Furthermore inspired by their work we employ a linear transformation to recreate the original embedding using the generated binary representations:

$$a'(i) = W'(i) \; . \; b + j'(i) \tag{4}$$

This makes sure that the learnt binary representations learnt, b carry more information from the continuous embedding a. The auto encoder model is optimized by minimizing the reconstruction loss, L_{rec}. Along with the reconstruction loss we use two different kind of regularizers, semantic preservation regularizer and the expansive regularizer, to make sure that we carry the most amount of information in our binarized embedding. After the training we use the trained parameters to create the binary embedding from the original continuous embedding.

$$L_{rec} = \frac{1}{G} \sum_{i=1}^{G} (a(i) - a'(i))^2 \tag{5}$$

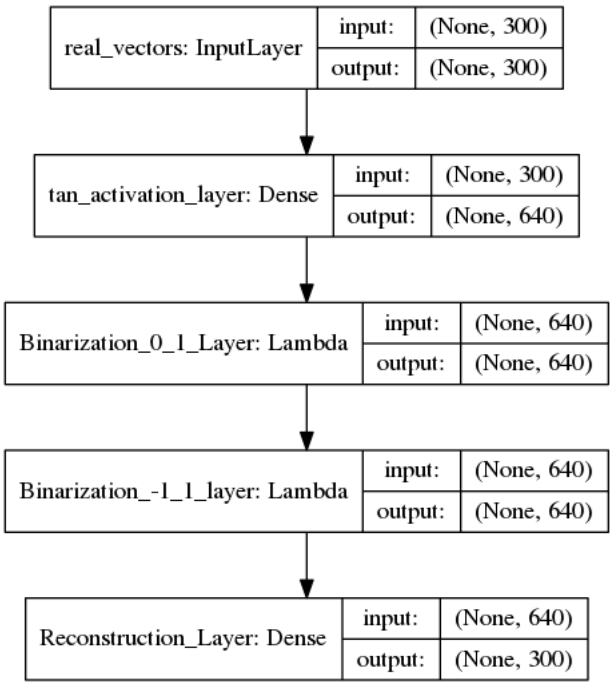

Figure 1: Auto-Encoder Architecture for Binary Embedding

3.1.1 Semantic Preservation Regularizer

Even though the reconstruction loss tries to ensure that the binary representations are filled with rich semantics, it still does not mean that these representations have similar information stored like the original continuous embedding. The model may be optimized with a low reconstruction error but will not yield optimal result on downstream tasks. To retain as much semantic information as we can, we introduce another additional component.

Suppose we have a group of words, (x_α, x_β, x_γ, x_δ), their continuous embedding are represented by (a_α, a_β, a_γ, a_δ) and their binary representations be represented by (b_α, b_β, b_γ,b_δ). If the cosine similarity between a_α and a_β is large, then the Hamming Distance between b_α and b_β should be small, a large cosine similarity means that the words x_α and x_β are similar and we try to capture this in the binary representations. Hamming Distance is used in comparing binary strings of equal length and defined as the number of bit positions in which the strings bits are different. If the cosine similarity between a_α and a_β is larger than the cosine similarity between a_γ and a_δ, then it would be preferable to have the Hamming Distance between b_α and b_β be lesser than the Hamming Distance between b_γ and b_δ.

Let d_c depict the cosine similarity in the continuous embedding space and d_h depict the Hamming Distance in the binary space. We introduce a term $I_{\alpha,\beta,\gamma,\delta}$ such that this indicator is 1 if d_c (a_α, a_β) >= d_c (a_γ, a_δ) and -1 for any other case. Thus this regularizer is defined as

$$l_{sp} = \sum_{\alpha,\beta,\gamma,\delta} max\{0, I_{\alpha,\beta,\gamma,\delta}[d_h(b_\alpha, b_\beta) - d_h(b_\gamma, b_\delta)]\} \tag{6}$$

This regularizer encourages the learning function to retain the semantic similarity from the original embedding. While training our auto-encoder model, to calculate the semantic preservation regularizer value, we divide the training data into four sets of equal size and random allocation. We then use the corresponding elements from these 4 groups as x_α, x_β, x_γ and x_δ respectively, and calculate the regularizer value as per equation 6. To capture maximum semantic information during each epoch we use a random permutation to get the 4 groups.

3.1.2 Expansive Regularizer

We observed a subpar performance from the learned vector in the case where optimizing the reconstruction loss was the main aim. This is because the learned embedding failed to perserve the semantic information from the original embedding. Due to this, we reinforced the amount of semantic data being preserved by adding the expansive regularizer. This regularisation term in the objective function is defined as:

$$l_{exp} = \frac{1}{2}||W^T W - I||^2 \tag{7}$$

Where W is the learnt weight matrix and I is an identity matrix. This term manages to minimize the correlation between the features of the embedding vector. The reduction in the correlation between the features means that the information stored in the features differ from each other. It also means that more semantic similarity information can be carried across to the features of the resulting binary embedding. As the model aims to build binary representations which are small in size, it becomes imperative to carry non duplicate information. This regularizer makes sure that the correlation between the features of an embedding is as less as possible.

The combination of the above regularizers give the final objective function, the ability to retain the information stored in the continuous real valued embedding, thereby making sure that the vectors that are closer in the real space are closer in the binary space as well. The balance between the reconstruction loss, the semantic regularizer and the expansive regularizer are brought out by the λ_{sp} and λ_{exp} parameters. The global objective function to minimize becomes:

$$L = L_{rec} + \lambda_{sp}\, l_{sp} + \lambda_{exp}\, l_{exp} \tag{8}$$

4 Dataset

4.1 Binary Embedding

We make the use of the Word2vec Embedding which is very widely used in majority of the NLP tasks. The embedding we used had 300 dimensions and the vocabulary consisted of 3,000,000 words. These embedding were created by training on various news articles from Google. We also used the Glove Embedding to generate Binary representation. Training of these embedding was performed on aggregated global word-word co-occurrence statistics from a corpus. The Glove embedding contain 6 Billion tokens with a 400,000 vocabulary size. Each word vector in the embedding is of 300 dimensions. The experiments are currently focused on English language and related embedding.

4.2 Benchmark

For the benchmarks, we use categorisation, similarity and analogy based tasks. For the Categorisation Benchmark, we have used AP, BLESS, Battig, ESSLI_2b, ESSLI_2c and the ESSLI_2a datasets. For the Similarity Benchmarks we make use of the MEN, WS353, SimLex999, RW, RG65, MTurk datasets. For the Analogy based tasks we make the use of Google, MSR and SemEval datasets. More information of these datasets are given in Section 5 under the respective task details.

5 Experiments

Several tasks have been run to measure the performance of the original continuous embedding and the binarized embedding. These tasks test how much semantic information has been retained in the binary embedding.

5.1 Binary Embedding

Three different variations of binary embedding (for Word2Vec and GloVe respectively) were created. Each binary embedding was compared with each other, which is shown in the following sections. The first variation of binary embedding is directly taken from (Tissier et al., 2019) and it uses the reconstruction loss and the expansive regularizer (Variation 1). The second variation of the binary embedding is

the static auto encoder architecture which had a threshold value (t) of 0 (Variation 2). The third variation of the binary embedding is the dynamic auto encoder architecture which had the mean value of the embedding as the threshold (t) (Variation 3). Variation 2 and Variation 3 consists of the reconstruction loss, the semantic preservation regularizer and the expansive regularizer as per equation (8). Each of these embeddings had an input vector size of 300 dimension and the binary embedding were scaled up to 640 dimensions. We compare our propsed Variation 2, Variation 3, with original embedding and Variation 1. We make the use of word-embedding-benchmarks [1] to compare the results of the different embedding as expained in the next section.

5.2 Categorization

This evaluation is done by using clustering techniques like Agglomerative and k-means. The embedding are evaluated based on purity value returned by this method. A noun categorizer was used to measure the purity of the embedding. The categorization tasks utilise the Almuhareb and Abdulrahman categorization dataset(Almuhareb and Poesio, 2005), Baroni and Macro categorization dataset(Baroni and Lenci, 2011) (consists of 200 discrete nouns from various different classes), the 1969 Battig dataset(Battig and Montague, 1969) (contains 5231 verbal items from 56 categories), ESSLI_2c dataset (contains 45 verbs which come from 9 semantic classes), ESSLI_2b dataset (40 nouns have been classified into 3 classes based on how abstract they are), ESSLI_1a dataset (consisits of 44 nouns from 6 semantic categories(four animates and two inanimate)).

5.3 Word Similarity

This evaluation is done by using the Spearman correlation between the cosine similarity of the model and the human rated similarity of word pairs. We evaluated the embedding using the spearman correlation value on multiple datasets, namely, MEN dataset (it is a dataset for testing similarity and relatedness, score is calculated on different scale but it has been changed to standard scale), WS353 dataset(Finkelstein et al., 2001) (dataset for testing attributional and relatedness similarity), Reubenstein and Goodenough Dataset(RG65)(Rubenstein and Goodenough, 1965) (dataset for testing attributional and relatedness similarity), Rare Words Dataset (dataset for testing attributional similarity), SimLex999 (dataset for testing attributional similarity).

5.4 Word Analogy

This evaluation creates a simple analogy solver using the several embedding that were built. The analogy based question varies over several different categories and a comparison is made using the accuracies across all the categories. The accuracy stands for the number of questions that were correctly answered. The embedding were standardized and normalized before they were shared with the analogy solver. The Google Word Rep dataset(Mikolov et al., 2013) (tests both semantic and syntactic analogies), MSR dataset(Gao et al., 2014) (testing performance on syntactic analogies) were used to evaluate the performance of the embedding. These embedding were also run on the SemEval 2012(Agirre et al., 2012) and the results were evaluated with the spearman correlation score.

6 Results

We have evaluated our binary embedding on above mentioned downstream tasks. The evaluation was done using 17 standard benchmarks to compare the proposed variations(Variation 2, Variation 3), Variation 1 of binary embedding and the original continuous embedding.

We have compared the reconstruction accuracies for the several variations of binary embedding. Reconstruction Accuracy denotes how much of the orignial embedding has been reconstructed from the generated binary embedding. As seen in Table 1, Variation 1 of Word2Vec and GloVe has been the most successful among the 3 variations in reproducing the original embedding. It is because in the case of Variation 1, unique non collinear data is being stored across the features of the embedding and this variation is doing well to reconstruct the original continuous embedding. This table focuses solely on

[1] https://github.com/kudkudak/word-embeddings-benchmarks

Embedding	Reconstruction Accuracy
Word2Vec(Variation 1)	32.3%
Word2Vec(Variation 2) - static threshold	25.8%
Word2Vec(Variation 3) - dynamic threshold	26.8%
GloVe(Variation 1)	46.0%
GloVe(Variation 2) - static threshold	45.0%
GloVe(Variation 3) - dynamic threshold	46.0%

Table 1: Comparison of reconstruction accuracies

the reconstruction without taking into account the quality of transfer of semantic similarity data from the original embedding to the binary embedding.

	MEN	MTurk	RG65	RW	SimLex999	WS353	WS353R	WS353S
Word2Vec (W2V)	0.6780	0.6801	0.7485	0.4848	0.4358	0.6261	0.5725	0.7204
W2V Variation 1	0.7432	0.6191	**0.7851**	**0.4295**	0.4452	0.5837	0.5124	0.6464
W2V Variation 2	**0.7433**	**0.6304**	0.7660	0.4175	**0.4516**	**0.6248**	**0.5337**	**0.6999**
W2V Variation 3	0.6176	0.5124	0.5668	0.3853	0.3890	0.5756	0.4373	0.6349
GloVe	0.7375	0.6332	0.7695	0.3670	0.3705	0.5433	0.4775	0.6620
GloVe Variation 1	0.6996	0.6227	**0.7363**	0.3340	0.3636	0.5454	0.5018	0.5969
GloVe Variation 2	0.7172	**0.6520**	0.7301	0.3367	0.3715	0.5778	0.5657	0.6013
GloVe Variation 3	**0.7460**	0.6186	0.7318	**0.3849**	**0.3857**	**0.5990**	**0.5686**	**0.6292**

Table 2: Comparison of embedding performance on similarity tasks

In Table 2, we compare the different variations of binary embedding on word similarity tasks. When comparing to original Word2Vec and GloVe, proposed Variation 2 performs comparably, given the reduction in the embedding size. When compared to the original binary embedding(Variation 1), our proposed Variation 2 outperforms in 6 out of 8 tasks for Word2Vec and outperforms in 7 out of 8 tasks for GloVe, with significant gains in WS353, WS353R and WS353S. However, binarization of Word2Vec using variation 3 performs poorly in word similarity tasks, when compared to both Variation 1 and Variation 2. Whereas Variation 3 on GloVe performs on par with other variations.

	Google	MSR	SemEval2012_2
Word2Vec	0.7262	0.6973	0.2093
Word2Vec Variation 1	0.3165	0.5578	0.2002
Word2Vec Variation 2	**0.3356**	**0.5589**	0.2024
Word2Vec Variation 3	0.2904	0.4643	**0.2026**
GLoVe	0.7174	0.6143	0.1699
GLoVe Variation 1	**0.6231**	0.4693	**0.1572**
GLoVe Variation 2	0.6137	**0.4709**	0.1518
GLoVe Variation 3	0.5953	0.4338	0.1497

Table 3: Comparison of embedding performance on analogy tasks

As seen in Table 3, original binary embedding (Variation 1) and proposed Variation 2, Variation 3

perform poorly in analogy tasks when compared to the original Word2Vec and GloVe embedding. The poor performance of the binary embedding on analogy tasks can be attributed to the fact that, analogy requires vector addition and subtraction, tasks for which binary vectors are unsuitable.

	AP	BLESS	Battig	ESSLI_1a	ESSLI_2b	ESSLI_2c
Word2Vec	0.6493	0.6950	0.4181	0.7955	0.7500	0.6444
Variation 1	**0.6517**	0.7350	0.3932	**0.7727**	0.7500	0.6222
Variation 2	0.6418	0.7250	**0.4007**	0.7273	0.7000	0.5778
Variation 3	0.6244	**0.7400**	0.3944	0.7500	**0.7520**	**0.6444**
GloVe	0.6368	0.8200	0.4120	0.7500	0.8250	0.6444
GloVe Variation 1	0.6244	**0.8100**	**0.4039**	0.6818	0.7000	0.6000
GloVe Variation 2	**0.6343**	0.7750	0.3959	**0.7273**	**0.7500**	0.6000
GloVe Variation 3	0.6169	0.7600	0.3890	**0.7273**	**0.7500**	**0.6222**

Table 4: Comparison of embedding performance on classification tasks

	AP	BLESS	Battig	ESSLI_1a	ESSLI_2b	ESSLI_2c		
λ_{sp} = 0.2 λ_{exp} = 0.8	0.6119	**0.8200**	**0.4122**	0.6818	0.7000	0.6000		
λ_{sp} = 0.4 λ_{exp} = 0.6	**0.6343**	0.7750	0.3959	0.7273	0.7500	0.6000		
λ_{sp} = 0.6 λ_{exp} = 0.4	0.6294	0.7850	0.4112	0.7500	0.7250	**0.6222**		
λ_{sp} = 0.8 λ_{exp} = 0.2	0.5796	0.7850	0.4034	**0.7955**	**0.8000**	0.6000		
	MEN	MTurk	RG65	RW	SimLex999	WS353	WS353R	WS353S
λ_{sp} = 0.2 λ_{exp} = 0.8	0.7087	0.6264	**0.7302**	0.3165	0.3570	0.5627	0.5273	0.5973
λ_{sp} = 0.4 λ_{exp} = 0.6	**0.7172**	**0.6520**	0.7301	**0.3367**	**0.3715**	**0.577772**	**0.5657**	0.6013
λ_{sp} = 0.6 λ_{exp} = 0.4	0.7150	0.6434	0.7190	0.3357	0.3556	0.5632	0.5402	0.5998
λ_{sp} = 0.8 λ_{exp} = 0.2	0.6910	0.6116	0.7164	0.3313	0.3666	0.5717	0.5414	**0.6077**
	Google	MSR	SemEval2012_2					
λ_{sp} = 0.2 λ_{exp} = 0.8	0.6123	0.4729	0.1619					
λ_{sp} = 0.4 λ_{exp} = 0.6	**0.6137**	0.4709	0.1518					
λ_{sp} = 0.6 λ_{exp} = 0.4	0.6097	**0.4806**	0.1687					
λ_{sp} = 0.8 λ_{exp} = 0.2	0.5954	0.4680	**0.1827**					

Table 5: Comparison of embedding performance using different λ_{sp} and λ_{exp} values

In Table 4, when compared to original Word2Vec embedding, our proposed binary Variation 3 gets similar results in 5 of the performed tasks. We can also see that the proposed Variation 3 outperforms Variation 1 and proposed Variation 2 in 3 of the 6 tasks for Word2Vec embedding. When compared to original GloVe embedding, our proposed binary Variation 3 gets similar results in 4 of the performed tasks. We can also see that the proposed Variation 3 outperforms Variation 1 and proposed Variation 2 in 3 of the 6 tasks and is on par in 1 task.

It is also evident from tables 2,3,4 that our proposed Variation 2 and Variation 3 performs better in different tasks. Variation 3 performs better in tasks where the contextual and semantic information is

crucial, such as text categorisation. In the case of a static threshold (as in Variation 2), it focuses more on the meaning of the word as compared to the contextual information. These are proved as the dynamic thresholding method (proposed Variation 3) performs better in the categorization tasks whereas the static thresholding method (proposed Variation 2) performs better in the analogical and word similarity tasks.

As seen in Table 5, it proves that more semantic information can be transferred to the binary embedding when we have high values for the λ_{sp} . The same can be seen from the benchmark tests, for all tasks where more semantic information is key such as categorization, higher values of λ_{sp} work better. With higher values of λ_{sp} and lower values of λ_{exp} we get better benchmark results for categorization. Whereas with lower values of λ_{sp} and higher values of λ_{exp} we are able to have more word understanding leading to better results in tasks such as word similarity. It also shows that a smaller difference in the value λ_{sp} and λ_{exp} seems to work well for word similarity based tasks. These results were built using binary embedding created from the Variation 2 using Glove Embedding.

Embedding	Word2vec	Glove
Original (300 dimensions)	3.39 GB	0.99 GB
Binarized (640 dimensions)	0.5 GB	0.063 GB
reduction ratio	7 times	15 times

Table 6: Comparison of embedding file sizes

As seen in Table 6, the file size reductions varies from 7 to 15 times reduction as compared to the original file size. Our binarized embedding are of 640 dimensions and are stored in a text(.txt) file in their hexadecimal format further facilitating the reduction in size. When the original embedding are stored in binary (.bin) file, float values are usually used to store the embedding, so on average 4 bytes are used to store for each dimension of every vector. Due to this we see only a 7 times reduction in the size in the case of Word2Vec as we store our embedding in a text(.txt) format. In the case of Glove we see a 15 times reduction in size because their values are directly stored in a textual format, so 6 precision point float values takes 8 bytes to store in this case. As a result of that we see a 15 times reduction in size as compared to the original embedding. Both the Word2Vec and GloVe were of 300 dimensions and the binarized embedding were of 640 dimensions.

7 Conclusion and Future works

This paper presents a novel approach to generate binary embedding from any continuous embedding with significant reduction in size. The proposed approaches also allows preservation of semantic infomation in binary vectors. Our proposed approach creates binary embedding which performs similar to original continuous embedding and performs better in most cases than previously proposed binarization approaches on different downstream tasks like word similarity and classification. A 640 dimensional binary vector is 15 times smaller than a 300 dimension real valued vector. However there are some inconsistencies in performances on some downstream tasks. Our future work includes investigating and addressing these inconsistencies. We will also experiment further with different embedding, embedding dimensions and observe the effect on the performance on downstream tasks.

References

Eneko Agirre, Johan Bos, Mona Diab, Suresh Manandhar, Yuval Marton, and Deniz Yuret. 2012. * sem 2012: The first joint conference on lexical and computational semantics–volume 1: Proceedings of the main conference and the shared task, and volume 2: Proceedings of the sixth international workshop on semantic evaluation (semeval 2012). In ** SEM 2012: The First Joint Conference on Lexical and Computational Semantics–Volume 1: Proceedings of the main conference and the shared task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation (SemEval 2012)*.

Abdulrahman Almuhareb and Massimo Poesio. 2005. Concept learning and categorization from the web. In *proceedings of the annual meeting of the Cognitive Science society*, volume 27.

Marco Baroni and Alessandro Lenci. 2011. How we blessed distributional semantic evaluation. In *Proceedings of the GEMS 2011 Workshop on GEometrical Models of Natural Language Semantics*, pages 1–10.

William F Battig and William E Montague. 1969. Category norms of verbal items in 56 categories a replication and extension of the connecticut category norms. *Journal of experimental Psychology*, 80(3p2):1.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2016. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*.

Miguel A Carreira-Perpinán and Ramin Raziperchikolaei. 2015. Hashing with binary autoencoders. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 557–566.

Andrew M Dai and Quoc V Le. 2015. Semi-supervised sequence learning. In *Advances in neural information processing systems*, pages 3079–3087.

Manaal Faruqui, Yulia Tsvetkov, Dani Yogatama, Chris Dyer, and Noah Smith. 2015. Sparse overcomplete word vector representations. *arXiv preprint arXiv:1506.02004*.

Lev Finkelstein, Evgeniy Gabrilovich, Yossi Matias, Ehud Rivlin, Zach Solan, Gadi Wolfman, and Eytan Ruppin. 2001. Placing search in context: The concept revisited. In *Proceedings of the 10th international conference on World Wide Web*, pages 406–414.

Bin Gao, Jiang Bian, and Tie-Yan Liu. 2014. Wordrep: A benchmark for research on learning word representations. *arXiv preprint arXiv:1407.1640*.

Armand Joulin, Edouard Grave, Piotr Bojanowski, Matthijs Douze, Hérve Jégou, and Tomas Mikolov. 2016. Fasttext. zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651*.

Xuezhe Ma and Eduard Hovy. 2016. End-to-end sequence labeling via bi-directional lstm-cnns-crf. *arXiv preprint arXiv:1603.01354*.

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.

Jeffrey Pennington, Richard Socher, and Christopher D 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.

Herbert Rubenstein and John B Goodenough. 1965. Contextual correlates of synonymy. *Communications of the ACM*, 8(10):627–633.

Dinghan Shen, Qinliang Su, Paidamoyo Chapfuwa, Wenlin Wang, Guoyin Wang, Lawrence Carin, and Ricardo Henao. 2018. Nash: Toward end-to-end neural architecture for generative semantic hashing. *arXiv preprint arXiv:1805.05361*.

Dinghan Shen, Pengyu Cheng, Dhanasekar Sundararaman, Xinyuan Zhang, Qian Yang, Meng Tang, Asli Celikyilmaz, and Lawrence Carin. 2019. Learning compressed sentence representations for on-device text processing. *arXiv preprint arXiv:1906.08340*.

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.

Xinyang Yi, Constantine Caramanis, and Eric Price. 2015. Binary embedding: Fundamental limits and fast algorithm. In *International Conference on Machine Learning*, pages 2162–2170.