ECAI 2024
U. Endriss et al. (Eds.)

doi:10.3233/FAIA240966

BRECS: Enhanced Binary Representation of Word Embeddings via Cosine Similarity

Rajdeep Sarkar [a,*], **Sourav Dutta** [b] **and John McCrae** [c]

[a]Yahoo Research
[b]Huawei Ireland Research Centre
[c]Data Science Institute, University of Galway

Abstract. Word representations like GloVe and Word2Vec encapsulate semantic and syntactic attributes and constitute the fundamental building block in diverse Natural Language Processing (NLP) applications. Such vector embeddings are typically stored in *float32* format, and for a substantial vocabulary size, they impose considerable memory and computational demands due to the resource-intensive *float32* operations. Thus, representing words via binary embeddings has emerged as a promising but challenging solution.

In this paper, we introduce *BRECS*, an autoencoder-based Siamese framework for the generation of enhanced binary word embeddings (from the original embeddings). We propose the use of the novel Binary Cosine Similarity (BCS) regularisation in *BRECS*, which enables it to learn the semantics and structure of the vector space spanned by the original word embeddings, leading to better binary representation generation. We further show that our framework is tailored with independent parameters within the various components, thereby providing it with better learning capability. Extensive experiments across multiple datasets and tasks demonstrate the effectiveness of *BRECS*, compared to existing baselines for static and contextual binary word embedding generation. The source code is available at https://github.com/rajbsk/brecs.

1 Introduction

Word embeddings are continuous vector representations of words sourced from a vocabulary $\mathcal{V}$. These representations are constructed from the association between words from a large text corpus based on the *distributional hypothesis* [13]. Word embeddings have been shown to encapsulate both semantic and syntactic linguistic knowledge and have found widespread utility in the domain of Natural Language Processing (NLP). Applications of word embeddings form the fundamental basis for various applications, including language models, question-answering, machine translation, and dialogue systems among others. The construction process for word embedding involves techniques like generating word vectors such that the representations of co-occurring words are closer together [20] or by performing matrix factorisation of the co-occurrence statistics matrix [24]. These methodologies generate vector space depicting *semantic similarity* wherein words with similar meanings or contextual usage exhibit shorter distances relative to dissimilar words.

Given that each word embedding is represented by n-dimensional *float32* vector (wherein n is typically in the order of 100s), the storage requirements for such word embeddings can be substantially high. To illustrate, consider a vocabulary set comprising 3 million words, each associated with 300-dimensional word vectors (i.e., 300 *float32* real numbers per word). Storing such embeddings necessitates approximately 3.6 GB of storage space, posing challenges for its portability. Moreover, when deployed on resource-constrained embedded devices like mobile phones, word embeddings with their high memory demands and the computational burden of floating-point arithmetic operations can impede efficient performance. A commonly employed strategy for reducing the memory footprint of embeddings involves employing techniques like Principal Component Analysis (PCA) or Locality Sensitive Hashing (LSH). However, recent studies have demonstrated that embeddings resulting from dimensionality reduction exhibit poor performances [32].

To alleviate the above problem, methods for deriving *binary word embeddings* from continuous word embeddings [33, 23, 21, 29] have been proposed, termed as *Binary Quantization Learning (BQL)*. Binary vectors offer the advantage of reduced computational demands, as operations are performed using inexpensive binary operations rather than floating-point arithmetic. Additionally, each element in a binary vector consumes only one bit of storage, thus significantly decreasing the overall memory footprint. Consider, 3 million words each represented as a 300-dimensional binary vector would thus require a mere 112.5 MB – providing a storage reduction of 32 times (as compared to 3.6 GB). Further, such binary representation has enabled the development of large-scale vector database search for efficient retrieval [1].

Semantic similarity between binary word representations is then computed using the *Hamming distance* [5]. Tissier et al. [33] introduced the utilisation of an autoencoder architecture for the word embedding binarisation task. The autoencoder comprises an encoder network responsible for transforming continuous embeddings into binary embeddings and a decoder network for the reverse conversion from binary to continuous embeddings, minimising the reconstruction loss. It uses the *Heaviside step function* for the conversion of encoder representations into binary space. Due to the non-differentiable nature of this function, the same parameters were shared between the encoder and decoder, thereby limiting the learning capability of the network. Navali et al. [23] extended the aforementioned work by incorporating semantic preservation regularisation into the network to

* Corresponding Author. Work done when the author was a PhD student at the University of Galway. Email: rajdeep.sarkar@yahooinc.com.

[1] blog.vespa.ai/billion-scale-knn & github.com/cohere-ai/BinaryVectorDB

enhance the generation of binary embeddings. The semantic preservation loss, a variant of contrastive loss, minimises the Hamming distance between binary embeddings for pairs of similar words and increases it for dissimilar word pairs. Nonetheless, it is important to highlight that the innovative semantic preservation regularisation technique relies on cosine similarity to assess if one pair of words is more similar than another pair of words, as opposed to directly approximating cosine similarity, leading to potentially suboptimal information capture.

In this paper, we present *Binary Representation lEarning via Cosine Similarity* (*BRECS*), a Siamese autoencoder-based binary word embedding framework to address the aforementioned challenges associated with existing methodologies. We introduce a novel Binary Cosine Similarity (BCS) loss function, that enables the network to directly approximate the cosine similarity between continuous embeddings and binary embeddings, thereby enabling the generated binary representation to efficiently capture semantic information from the real-valued word embedding vector space. Specifically, we employ the BCS function to propose a novel information capture mechanism that leverages continuous vector contexts for learning rich binary word embeddings. Further, we argue that the parameters of the encoder and decoder in an autoencoder architecture should be disentangled for better performance, as they inherently cater to distinct learning tasks. To tackle the issue of non-differentiability in this setup, we leverage Straight Through Estimator (STE) [4] to approximate the gradients of the non-differentiable Heaviside step function.

Our contributions can be summarised as follows:

- *BRECS*, an autoencoder-based binary word embedding framework for generating high-quality binary word embeddings;
- Novel BCS loss function for approximating the cosine similarity of continuous word embeddings within the binary representations to better capture semantic information;
- Use of STE function to decouple the encoder and decoder learning parameters, thus increasing the model performance; and,
- Extensive experiments on multiple benchmark tasks and datasets demonstrating performance improvement in *BRECS* over existing methods, for both *static* and *dynamic* embeddings.

2 Related Work

Word embeddings are the basic components of many Natural Language Processing (NLP) applications. Owing to the large vocabulary size, the memory footprint of these embeddings can render them expensive for application on small devices. In fact, current large models like XLM-R$_{XXL}$ have a vocabulary size of 250K and produce vector representations with 4096 dimensions [26] – making it infeasible for most scenarios. Recent studies have investigated techniques such as PCA for compressing the memory footprint of word embeddings [25] However, these embeddings are plagued by two significant issues. Firstly, the computations involved still operate at the precision of 32-bit floating-point numbers, leading to increased computational costs. Secondly, Thakur et al. [32] demonstrated that such embeddings exhibit suboptimal performance in various tasks compared to binary embeddings. Binary embeddings address this issue as they occupy less size in memory and faster bit operations.

Shen et al. [28] proposed NASH, a neural network architecture for fast retrieval of documents using binary encodings. Charikar [7] proposed Locality Sensitive Hashing that utilises random projections to generate binary embeddings that tend to approximate the cosine similarity. However, Xu et al. [34] pointed out that these methods are suboptimal in preserving the semantic information. Thereafter, Faruqui et al. [8] proposed to create binary embeddings preserving semantics similarities by increasing the vector size to create sparse vectors and then applying a binarisation function to these embeddings. Although innovative, the increased size of the embeddings fails to fit in CPU registers [33]. The Fasttext embedding by Joulin et al. [16] binarizes embeddings through clustering and concatenating the binary representations of the “k” closest centroids for each word. However, it is important to note that the resulting binary vectors are specialised and not suitable for general-purpose tasks, as they are primarily designed for document classification.

Tissier et al. [33] proposed an autoencoder architecture for transforming continuous word embeddings into binary embeddings. Navali et al. [23] built upon the prior autoencoder architecture and added a semantic preservation regularisation loss to capture the relationship between similar/dissimilar pairs of words in the binary space. Cohere[2] contextually embed texts using their transformer-based proprietary models for similarity and classification tasks. Further, hashing-based techniques coupled with autoencoder architecture have been proposed recently for retrieval tasks [10]. We observe *BRECS* to outperform previous methods on two benchmark tasks suggested by [15] on multiple datasets as showcased in Section 5.

3 *BRECS* Framework

In this section, we formally introduce the problem statement and discuss the architecture and working of our proposed *Binary Representation Learning via Cosine Similarity* (*BRECS*) framework. We initially introduce our novel Binary Cosine Similarity (BCS) function on the binary representation of words. Subsequently, in Section 3.3 we present the autoencoder architecture of *BRECS* with Straight Through Estimator (STE) for generating binary embedding, along with the use of BCS for semantic information capture in Section 3.5.

3.1 Problem Definition

Given a vocabulary $\mathcal{V}$ of words, where each word $w_i \in \mathcal{V}$ is represented by an m-dimensional real-valued vector $x_i \in \mathbb{R}^m$, the task is to generate an n-dimensional binary representation $b_i \in \{0,1\}^n$ for w_i that can effectively capture the semantic knowledge from the continuous word embedding x_i.

3.2 Binary Cosine Similarity (BCS) Function

Cosine similarity between two word embeddings presents the defacto measure to enumerate the semantic relatedness between the words. Semantically similar words are mapped close to each other in the embedding space and depict high cosine similarity (and vice-versa for dissimilar words). The Continuous Cosine Similarity (CCS) of two real-valued word embedding vectors is,

$$CCS(x_{w_1}, x_{w_2}) = \frac{\vec{x_{w_1}} \cdot \vec{x_{w_2}}}{||\vec{x_{w_1}}|| \times ||\vec{x_{w_2}}||} \tag{1}$$

, where $\cdot$ denotes vector dot product and $||\vec{x}||$ denotes the norm, with $-1 \leq CCS(x_{w_1}, x_{w_2}) \leq 1$.

BRECS aims to capture the cosine similarity between two word vectors by learning binary word embeddings capable of approximating this similarity measure via the proposed Binary Cosine Similarity (BCS) function.

[2] www.cohere.com

Let w_1 and w_2 be two words represented by real-valued m-dimensional vectors x_{w_1} and x_{w_2} (i.e., original embeddings). Further, consider that the corresponding n-dimensional binary embeddings of the words are b_{w_1} and b_{w_2} respectively. We define the BCS function so as to map a binary vector to a real number (to approximate the cosine similarity value), i.e., BCS : $\{0,1\}^n \to \mathbb{R}$. Observe, $CCS : \mathbb{R}^m \to \mathbb{R}$. Specifically, we define the cosine similarity in the binary space using BCS as follows:

- Compute the bitwise similarity between b_{w_1} and b_{w_2}. This is computed by the bitwise XNOR operation ($\oplus$) on the binary vectors. Hence, we define bit overlap b_o as,

$$b_o = b_{w_1} \oplus b_{w_2} \tag{2}$$

  Here, the i^{th} bit in b_o is 1 if both b_{w_1} and b_{w_2} have the same bit value at the i^{th} position, else is set to 0 (for $i \in [0, 1, \cdots, n-1]$).
- The bit positions in b_o are then given exponentially decreasing weights. In our framework, we set the weights of the i^{th} bit in b_o to be 2^{-i}. The value of b_o is then computed via BCS as the summation of the product of bit weights and bit values. That is,

$$BCS(b_{w_1}, b_{w_2}) = BCS(b_o) = \sum_{i=0}^{n-1} 2^{-i} \cdot b_{o_i} \tag{3}$$

  Observe, that when all the bits of b_o are 0, we get the lower bound of BCS to be 0. Alternatively, when all bits of b_o are 1, BCS takes the value $\frac{1-2^{-n}}{1-2^{-1}}$. Notably, BCS monotonically increases with n, and as it approaches infinity ($n \to \infty$), BCS converges to the limiting value of 2. Thus we have, $0 \leq BCS(b_{w_1}, b_{w_2}) \leq 2$. This provides the basis of BCS used to learn binary word embeddings that approximate the cosine similarity of the real-valued word embeddings. Since, cosine similarity has a range of [-1, 1], we operate with $BCS(b_{w_1}, b_{w_2}) \approx CCS(x_{w_1}, x_{w_2}) + 1$, which *BRECS* is trained to approximate.

The Hamming similarity between two binary vectors is defined as the count of indices where the corresponding bits are identical. Consequently, vectors with a higher number of similar bits exhibit a high Hamming similarity, indicating a strong resemblance. A variant of the Hamming similarity is the BCS, which assigns a weight of 2^{-i} to similar bits at index i. *BRECS* leverages BCS during the training phase, whereas Hamming distance is employed during evaluation. Despite the difference in metrics, both training and evaluation operate within the same space, ensuring consistency and coherence.

In this work, we optimise the approximation of BCS with CCS to capture the different semantic information from continuous embedding to the binary embedding space as later detailed in Section 3.5.

3.3 BRECS Siamese Autoencoder

BRECS uses a Siamese autoencoder architecture having two components: an encoder and a decoder network as illustrated in Figure 1. The encoder network maps the input continuous (or real-valued) word embedding to a binary embedding, while the decoder reconstructs the continuous embeddings from the internal binary representations. Given a word w_i with continuous word embedding x_i, we compute its binary embedding b_i using the encoder as:

$$b_i = \psi(x_i) = \sigma(W_{enc}^T x_i), \tag{4}$$

where W_{enc} is the weight parameter to be learned and $\sigma(.)$ is an element-wise function that outputs a binary value given a real value. For this work, we use the Heaviside step function as σ.

The decoder maps the binary word embedding b_i back to the continuous space and is defined as:

$$y_i = f(W_{dec}^T b_i + c), \tag{5}$$

where W_{dec} is the learnable weight matrix. The function f is an element-wise hyperbolic tangent function to be able to map the word embeddings to the continuous space. The decoder and encoder parameters are trained to make the decoder outputs as close to the input word embeddings, i.e., trained to minimise the *reconstruction loss* between the original word embedding x_{w_i} and the reconstructed embedding generated by the decoder $\hat{x}_{w_i}$, as:

$$\mathcal{L}_{rec} = \frac{1}{m} \sum_{i=1}^{m} (x_i - \hat{x}_{w_i})^2 \tag{6}$$

We utilise a Siamese network for the autoencoder architecture to learn the binary word embeddings. Given a pair of words w_1 and w_2, the Siamese network uses the encoder to generate their binary word embeddings b_{w_1} and b_{w_2} from their real-valued representations x_{w_1} and x_{w_2} (given as input). The network is trained to learn the binary embeddings such that the approximation error between the BCS(b_{w_1}, b_{w_2}) and the cosine similarity between x_{w_1} and x_{w_2} is minimised.

3.4 Straight Through Estimator (STE)

Notably, the Heaviside step binarisation function σ in $\psi(.)$ in Eq. (4) is non-differentiable. Hence, to enable backpropagation, previous works [33, 23] used shared weight parameters W among the encoder and decoder networks. Utilising identical weights for both the encoder and decoder would detrimentally affect the learning process, as the objective learned by the encoder inherently differs from that of the decoder.

In this work, we leverage Straight Through Estimator (STE) [4] to address this challenge, thereby enabling the decoupling of parameters of the encoder and decoder. STEs enable learning a non-differentiable neural network by approximating the gradients of the non-differentiable component. The forward pass and the backward pass of the STE used in *BRECS* are defined as:

$$\text{Forward Pass} : \sigma(x) = \begin{cases} 1 & x > 0 \\ 0 & x \leq 0 \end{cases} \tag{7}$$

$$\text{Backward Pass} : \frac{\partial \sigma}{\partial x} \approx \frac{\partial x}{\partial x} = 1 \tag{8}$$

Thus, instead of computing the gradients of the Heaviside step function $\sigma(.)$, we use the identity function as a surrogate to estimate its gradients.

3.5 BCS Loss

The BCS function captures the similarity between b_i and b_j, the binary embeddings of words w_i and w_j, as follows:

$$\text{BCS}(b_i, b_j) = \sum_{k=0}^{n-1} 2^{-k} \cdot (b_{i_k} \oplus b_{j_k}) \tag{9}$$

where b_{i_k} denotes the k^{th} bit in binary representation b_i of word w_i.

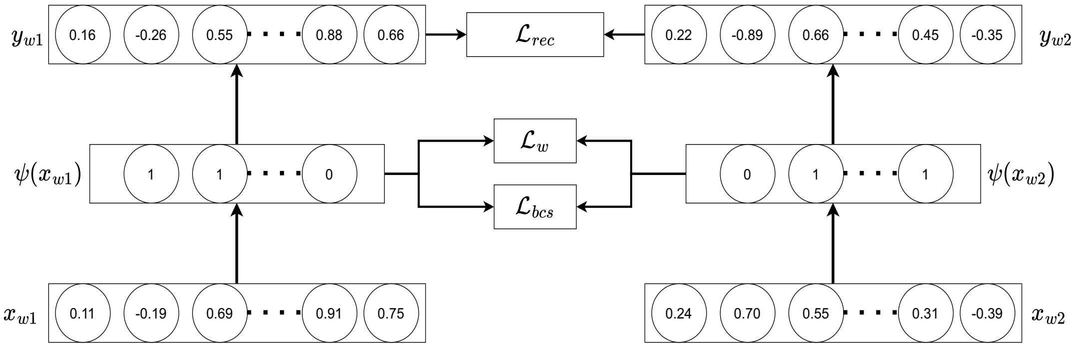

Figure 1. Siamese network framework of *BRECS* to generate binary word representations from the real-valued embeddings using autoencoder architecture.

The *BCS Loss* $\mathcal{L}_{bcs}$ aims to minimise the deviation of the similarity between the binary embeddings b_i and b_j (using the BCS function) and the cosine similarity between the real-valued *float32* embeddings (x_i and x_j), for word pairs w_i and w_j. Thus the learning process can be formulated via the squared error as:

$$\mathcal{L}_{bcs} = \left(e^{\widehat{CCS}(x_i,x_j)} - e^{BCS(b_i,b_j)}\right)^2 \tag{10}$$

, where $\widehat{CCS}(x_i, x_j) = CCS(x_i, x_j) + 1$, and CCS computes the cosine similarity of the continuous word embeddings (refer Eq. (1) in Sec. 3.2).

3.6 Expansive Regulariser

Following Tissier et al. [33], we introduced the expansive regulariser for optimising the autoencoder weights. Given that binary embeddings possess significantly lower capacity compared to their continuous counterparts, it becomes imperative to diminish the inter-feature correlations within the binary embeddings. Hence, we use,

$$\mathcal{L}_w = 0.5(||W_{enc}^T W_{enc} - I||^2 + ||W_{dec}^T W_{dec} - I||^2) \tag{11}$$

where W_{enc} and W_{dec} are the encoder and decoder weights respectively, while I is the identity matrix. The formulation of the loss term $\mathcal{L}_w$ serves to reduce such correlations within the latent binary representations. This prevents redundancy in the learned features across different binary attributes and, consequently, enhances the efficiency of information transfer to the binary embeddings.

The overall loss function used by *BRECS* during the training process is defined as:

$$\mathcal{L} = \mathcal{L}_{rec} + \lambda_w \mathcal{L}_w + \lambda_{bcs} \mathcal{L}_{bcs} \tag{12}$$

where the weights λ_w and λ_{bcs} are model hyperparameters. As discussed earlier in Section 3.2, BCS (used only during the training phase) can be viewed as a weighted Hamming similarity function. Similarly, during the evaluation phase, we employ the *Sokal & Michener similarity function* (refer Section 4.1), a modification of the Hamming similarity, to compute the similarity between binary embeddings. Thus, the training and evaluation phases operate on similar objectives. Observe, that this is in the same spirit as the Euclidean distance-based training and cosine similarity-based evaluation for language models (since Euclidean distance and cosine similarity are closely related).

4 Experimental Setup

We now describe the empirical setup for evaluating *BRECS* against multiple baselines on different tasks across a variety of benchmark datasets. We describe the tasks and datasets used for evaluating the quality of binary embeddings, the competing baselines and the parameter details of *BRECS*.

4.1 Tasks, Datasets and Evaluation Metrics

Following Navali et al. [23], we evaluate the quality of the binary embeddings on the following:

- **Word Similarity:** Given pairs of words, the objective here is to measure Spearman's rank correlation between the human-rated similarity score and the computed word embedding similarity. To achieve this, we compute the cosine similarity between continuous word embeddings, while for binary embeddings the similarity is measured using the Sokal & Michener similarity function [30] defined as $\frac{n_{00}+n_{11}}{n}$. Here n_{00} and n_{11} are the number of bits in the two binary embeddings that are both 0 and 1 respectively, while n is the length of the binary embedding. Following Tissier et al. [33] and Navali et al. [23], we evaluate the word embeddings on the MEN [6], RW [18], SimLex [12] and WS353 [9] datasets.
- **Word Categorisation:** This task focuses on utilising a noun categoriser for evaluating the word embeddings via clustering, and the methods are evaluated on the purity of the clusters generated. Following Navali et al. [23], we evaluate the different approaches using Agglomerative and K-Means clustering as suggested by Jastrzebski et al. [14]. The evaluation encompasses a range of datasets, including the AP dataset [1], the BLESS dataset [2], comprising 200 discrete nouns representing diverse classes, the 1969 Battig dataset [3] containing 5231 verbal items distributed across 56 categories. We also utilise ESSLI 2c dataset [19], consisting of 45 verbs categorised into 9 semantic classes, while the ESSLI 2b dataset [19] features 40 nouns classified into 3 categories based on their abstractness and the ESSLI 1a dataset [19], featuring 44 nouns distributed among 6 semantic categories, encompassing four animate and two inanimate classes.

These tasks measure the quality of the intrinsic semantic information retained by the binary word embeddings.

4.2 Pretrained Embeddings

We present a comprehensive evaluation of our methodology across three prominent word embedding spaces: Word2Vec, GloVe, and Cohere. The Word2Vec embeddings [20] comprise a large vocabulary of 3,000,000 words, learned from a corpus of news articles, based on word co-occurrences. In contrast, the GloVe embeddings [24] capture semantic relationships for 400,000 words, derived from the English Wikipedia and Gigaword 5 corpora, using co-occurrence matrix factorization. To further generalise our approach, we also evaluate our methodology on Cohere[3] embeddings, relying on contextualized embeddings from transformer architecture. Cohere, provides closed-source dense vector representations in both *float32* and *binary* formats. Notably, the static embeddings of Word2Vec and GloVe embeddings are represented in a 300-dimensional space, whereas the Cohere embeddings occupy a 1024-dimensional space.

4.3 Baseline Methodologies

We compare our proposed *BRECS* framework against the following baseline approaches:

- Tissier et al. [33]: A binary word embedding generator based on the autoencoder architecture. The encoder network of the autoencoder maps a continuous word embedding into a binary word embedding utilising a Heaviside step function. Subsequently, the decoder leverages these binary word embeddings to reconstruct the original continuous word embedding. We report the results as presented by Navali et al. [23].
- Navali et al. [23]: Similar to the architecture proposed by Tissier et al. [33], the methodology uses an autoencoder to construct binary word embeddings. Additionally, a semantic preservation regulariser is introduced to minimise the Hamming distance between similar word pairs while simultaneously increasing the Hamming distance between dissimilar word pairs. We report the results as published by the authors.
- *Cohere*[4]: A recent closed-source contextual embedding model capable of generating 1024-dimensional binary vector representations from textual input. We obtain the binary embeddings using the available API call.
- For completeness, we also compare the performance of *BRECS* with the original *float32* real-valued Word2Vec, GloVe and Cohere embeddings.

4.4 BRECS Implementation Details

For training *BRECS* to learn the binary embeddings, we randomly sample one million word pairs for Word2Vec, GloVe and Cohere. For training *BRECS* on the Cohere embeddings, we first embed the words in *float32* precision in Cohere and then train *BRECS* using these *float32* Cohere embeddings as input to learn the binary representations of the words. Following Tissier et al. [33], we set the learning rate to 0.001 for all the proposed models. The batch size is set to 256. The number of bits in the learned binary representation (n) is set to 640 for GloVe and Word2Vec, while for Cohere, it is set to 1024 (for even comparison with 1024 binary representation as returned by Cohere API). In *BRECS*, for experiments on Word2Vec embeddings, we set λ_w and λ_{bcs} to 0.3 and 0.7 respectively, observed to provide the best results, using grid search. Following the same procedure for GloVe and Cohere embeddings, we set λ_w, and λ_{bcs} to 0.4 and 0.6 respectively. For obtaining *float32* and *binary* Cohere representation, we use the *embed-english-v3.0*[5] model.

[3] https://cohere.com/
[4] https://cohere.com/blog/int8-binary-embeddings
[5] https://cohere.com/blog/introducing-embed-v3

Table 1. Performance of different methodologies on the word similarity task. For *BRECS*, we have reported the number over an average of five runs of the model. The numbers in bold depict the best-performing binary embedding framework (i.e., excluding $float32$ results).

	Model	MEN	RW	SimLex	WS353
GloVe	*float32*	73.75	36.70	37.05	54.33
	float32-whitened	75.43	43.55	39.70	63.56
	Tissier et al. [33]	69.96	33.40	36.36	54.54
	Navali et al. [23]: V2	71.72	33.67	37.15	57.78
	Navali et al. [23]: V3	74.60	38.49	38.57	59.90
	BRECS	**74.70**	**45.45**	**39.75**	**65.48**
Word2Vec	*float32*	67.80	48.48	43.58	62.61
	float32-whitened	75.92	54.51	47.46	64.47
	Tissier et al. [33]	74.32	42.95	**44.52**	58.37
	Navali et al. [23]: V2	74.33	33.67	37.15	57.78
	Navali et al. [23]: V3	61.76	38.49	38.57	59.90
	BRECS	**75.85**	**51.76**	44.29	**64.84**
Cohere	*float32*	74.85	56.81	59.93	71.51
	float32-whitened	71.53	56.82	59.92	71.53
	binary	69.36	53.30	56.89	69.41
	Tissier et al. [33]	71.02	54.26	56.62	69.18
	BRECS	**72.07**	**55.49**	**57.46**	**70.30**

For inference, we evaluated BRECS on 400,000 GloVe float32 word embeddings under two scenarios. With a batch size of 1000, binarization took about 25 seconds, or 0.07 ms per sample. In a real-world scenario where samples arrive individually, a batch size of 1 resulted in 960 seconds, or 2.5 ms per sample. These results demonstrate BRECS's scalability and effective deployment in resource-limited environments. All the experiments were implemented using the PyTorch library and evaluated on NVIDIA 1080Ti GeForce GPUs.

5 Results and Discussion

In this section, we report and discuss the performance of *BRECS* against other baselines for binary word embedding on different tasks. Additionally, we conduct ablation studies to understand the impact of changing the different components of *BRECS* that can potentially impact the performance.

5.1 Quantitative Results

We initially report the performance of the methodologies on open-source benchmark tasks.

5.1.1 Performance on Word Similarity

Table 1 reports the performance of different state-of-the-art binary embeddings on word similarity tasks on four datasets. For GloVe embeddings we observe that the performance of *BRECS* surpasses all the existing state-of-the-art baseline approaches. Notably, the most significant improvements are observed in the RW and WS353 datasets, where the relative performance increase exceeds 9%. For Word2Vec embeddings, *BRECS* exhibits enhancements over established baselines across all datasets, except for the RW dataset where it ranks as the second-best model, trailing the top-performing approach by a mere 0.23 points. Further, *BRECS* is seen to perform better than the recently released Cohere binary embeddings. In summary, *BRECS* demonstrates robust state-of-the-art performance in word similarity tasks by effectively harnessing information from continuous embeddings.

Table 2. Performance comparison of the approaches on the word categorisation task. For *BRECS*, we report performance across an average of 5 runs of the model. The numbers in bold depict the best-performing binary embedding framework (i.e., excluding $float32$ results).

	Model	AP	BLESS	Battig	ESSLI 1a	ESSLI 2b	ESSLI 2c
GloVe	*float32*	63.68	82.00	41.20	75.00	82.50	64.40
	float32-whitened	66.17	78.50	41.69	68.18	65.00	57.78
	Tissier et al. [33]	62.44	**81.00**	40.39	68.18	70.00	60.00
	Navali et al. [23]: V2	**63.43**	77.50	39.59	**72.73**	75.00	60.00
	Navali et al. [23]: V3	61.69	76.00	38.90	**72.73**	75.00	62.22
	BRECS	61.44	76.00	**40.79**	70.45	**77.50**	**64.47**
Word2Vec	*float32*	64.93	69.50	41.81	79.55	75.00	64.44
	float32-whitened	59.45	84.00	38.34	72.72	75.00	57.78
	Tissier et al. [33]	65.17	73.50	39.32	77.27	75.00	62.22
	Navali et al. [23]: V2	64.18	72.50	**40.07**	72.73	70.00	57.78
	Navali et al. [23]: V3	62.44	74.00	39.44	75.00	**75.20**	64.44
	BRECS	**66.91**	**75.00**	39.20	**81.81**	75.00	**68.89**
Cohere	*float32*	61.94	78.50	48.48	72.72	85.00	57.78
	float32-whitened	56.71	78.00	39.43	63.63	75.00	51.11
	binary	60.19	79.00	**46.93**	72.72	72.50	**55.55**
	Tissier et al. [33]	61.19	82.00	46.01	70.45	**75.00**	51.11
	BRECS	**63.18**	**82.50**	43.72	**79.54**	**75.00**	51.11

Observation. Interestingly, it is worth highlighting that we observe *BRECS* to outperform even the original continuous *float32* embeddings for both GloVe and Word2Vec embeddings on almost all datasets. We relate this to the *anisotropic property* [11] of embedding techniques as shown by Mu and Viswanath [22]. It reports that static and contextualized embedding space is prone to a spatial bias, wherein embeddings of texts are concentrated within a narrow conical region, leading to unrelated words depicting high cosine similarities. This anisotropy, characterised by a non-uniform angular distribution of word vectors, can lead to inefficient utilisation of the embedding space. To mitigate this issue, *embedding whitening* has been proposed as a means of reducing anisotropy in text representations [27, 31].

The use of *expansive regularisation* in *BRECS* (see Eq. (11) in Sec. 3.6), tends to minimise the correlations between the features of the learnt binary representations, as discussed in [33]. This can be viewed as a type of representation whitening, leading to better performance in our model (along with the BCS function).

To validate the above, in this work, we also apply *ZCA-Whitening* [17] to the original *float32* GloVe, Word2Vec, and Cohere embeddings to render the embedding space isotropic. Experiments on these *float32* whitened embeddings (reported as *float32*-whitened) demonstrate that they significantly outperform their vanilla *float32* counterparts for GloVe and Word2Vec. In contrast, the whitened Cohere embeddings exhibit similar or inferior performance across different datasets. This may be attributed to the fact that the original Cohere *float32* embeddings already exhibit isotropic properties (based on the training objective), which diminishes the impact of whitening. We do not report the performance of Navali et al. [23] for Cohere embeddings as their code is not openly available.

5.1.2 Performance on Word Categorisation

Table 2 presents the results of various models for the word categorisation task. When considering GloVe embeddings, *BRECS* surpasses the other state-of-the-art techniques in 3 out of the 6 datasets with comparable performance for the others. For Word2Vec embeddings, *BRECS* achieves state-of-the-art performance in 4 out of 6 datasets. Notably, it outperforms vanilla continuous word embeddings in 4 out of the 6 datasets within this task and performs comparably on ESSLI 2b. For Cohere embeddings, *BRECS* outperforms other methodologies on 4 datasets, while the Cohere *binary* embeddings showcase strong performance on the other 2 datasets.

Similar to the word similarity task, on multiple datasets *BRECS* is seen to perform better than the original vanilla *float32* embeddings. It should be observed that reducing the anisotropy property of word embeddings via whitening affects the cosine similarity values between the float32 representations. Since here we evaluate the task of word categorisation, *float32-whitened* embeddings are not seen to bring much value in general.

5.2 Qualitative Study

In this section, we present a thorough examination of the design choices underlying the *BRECS* framework, with the goal of elucidating the implications of these decisions on the overall performance of the system.

5.2.1 Impact of Binary Embedding Dimensionality

The number of bits contained within binary embeddings exhibits a direct correlation with the capacity of the said binary word embeddings. To investigate the influence of binary word embedding size on the performance of *BRECS* on word similarity task, we analyze with varying binary embedding dimensions. The results are summarized in Table 3.

Evidently, *BRECS* displays a notable monotonic enhancement as the binary embedding dimension size increases. This improvement can be attributed to the expanded capacity of the word embeddings. Notably, *BRECS* consistently outperforms the methodology proposed by Tissier et al. [33] across all binary embedding sizes. It is interesting to observe that when configured with 256 and 512 bits, *BRECS* surpasses conventional Word2Vec embeddings in performance metrics on the MEN dataset. This phenomenon underscores the capture of rich semantic and syntactic representation achieved by our binary embeddings, given adequate learning space.

Table 3. Performance impact of *BRECS* with varying number of bits (n) on the word similarity task for Word2Vec.

Size	Model	MEN	RW	SimLex	WS353
300	*float32*	67.80	48.48	43.58	62.61
	float32-whitened	75.92	54.51	47.46	64.47
64	Tissier et al. [33]	46.10	25.10	20.50	30.10
	BRECS	52.86	34.29	30.23	46.31
128	Tissier et al. [33]	63.30	34.30	31.40	44.90
	BRECS	65.44	41.82	37.07	57.26
256	Tissier et al. [33]	69.40	40.70	37.20	56.60
	BRECS	71.43	48.14	41.15	60.84
512	Tissier et al. [33]	72.70	40.20	36.80	60.30
	BRECS	74.67	50.62	43.78	63.83

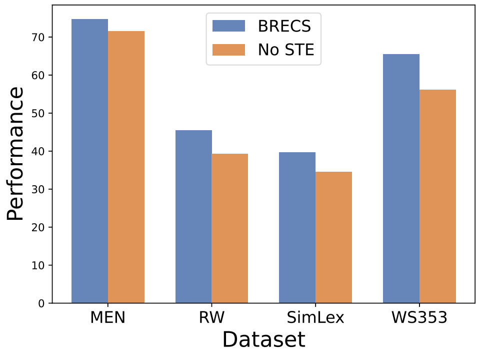

Figure 2. Impact of *BRECS* without STE on word similarity task performance.

5.2.2 Impact of STE for Decoupling Parameters

BRECS distinguishes itself from prior research by employing distinct parameters for its encoder and decoder, as we postulated that the tasks undertaken by them are inherently differ. To substantiate our hypothesis, we conducted an ablation analysis in which we configured *BRECS* with encoder and decoder weight matrices (W_{enc} and W_{dec}) set to be equal, and omitted the utilisation of the STE function while keeping all other model components consistent between the two configurations.

As depicted in Figure 2, we observe a marked disparity in the performance of *BRECS* and the ablated model. Specifically, when the encoder and decoder networks share parameters (depicted as No STE by the orange bar), the performance is notably inferior compared to the use of STE function for decoupling these networks (illustrated by the blue bar). This contrast depicts the usefulness of STE to enhance the learning of binary word representations.

5.2.3 Impact of Exponential Weights in BCS

We investigate the efficacy of our proposed BCS function, which assigns exponential weights to each bit in the binary vector. To further analyse the impact of this formulation, we conduct ablation studies by modifying the BCS function to employ *uniform weights*, where each bit is assigned a weight of $\frac{2}{n}$, with n denoting the dimensionality of the binary vector. In this setting, the underlying assumptions inherent to the formulation of the BCS function are indeed satisfied (i.e., $0 \leq BCS(b_{w_1}, b_{w_2}) \leq 2$).

Additionally, we explore a *learnable weighting scheme*, where the bit weights are parameterised and optimised during the training phase. Results in Table 4 demonstrate that the proposed exponential weighting strategy, as used in *BRECS*, outperforms the alternative weighting schemes.

Table 4. Performance impact of different weighting schemes in BCS on word similarity task with GloVe embeddings.

Model	MEN	RW	SimLex	WS353
Uniform	73.13	40.48	36.58	57.78
Learnable	71.85	40.28	37.95	55.37
Exponential (*BRECS*)	**74.70**	**45.45**	**39.75**	**65.48**

In a nutshell, we observe that the binary embeddings generated by *BRECS* are robust across various tasks and provide state-of-the-art performance on several benchmark datasets.

6 Conclusion

In our research, we introduce a novel Siamese autoencoder-based word embedding binarisation architecture named *BRECS* for transforming continuous word embeddings into binary representations. The autoencoder comprises an encoder network responsible for converting continuous embeddings into binary ones and a decoder network for reverse transformation. The encoder network employs the non-differentiable Heaviside step function with STE to approximate its gradient. The incorporation of STE enables the decoupling of the parameters of the encoder and decoder networks, leading to significant performance improvements. We propose the innovative BCS learning function to approximate the cosine similarity between the origin embeddings – enable better capture of semantic information in the binary encodings.

As a future direction, we plan to extend our work to encompass the generation of binary sequence representations for document retrieval applications.

Acknowledgements

John McCrae is supported by Science Foundation Ireland under Grant Number SFI/12/RC/2289_P2 Insight_2, Insight SFI Centre for Data Analytics and Grant Number 13/RC/2106 P2, ADAPT Centre.

References

[1] A. Almuhareb and M. Poesio. Concept learning and categorization from the web. In *proceedings of the annual meeting of the Cognitive Science society*, volume 27, 2005.

[2] M. Baroni and A. Lenci. How we blessed distributional semantic evaluation. In S. Padó and Y. Peirsman, editors, *Proceedings of the GEMS 2011 Workshop on GEometrical Models of Natural Language Semantics, Edinburgh, UK, July 31, 2011*, pages 1–10. Association for Computational Linguistics, 2011. URL https://aclanthology.org/W11-2501/.

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

[4] Y. Bengio, N. Léonard, and A. C. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *CoRR*, abs/1308.3432, 2013. URL http://arxiv.org/abs/1308.3432.

[5] A. Bookstein, V. A. Kulyukin, and T. Raita. Generalized hamming distance. *Inf. Retr.*, 5(4):353–375, 2002. doi: 10.1023/A:1020499411651. URL https://doi.org/10.1023/A:1020499411651.

[6] E. Bruni, N. Tran, and M. Baroni. Multimodal distributional semantics. *J. Artif. Intell. Res.*, 49:1–47, 2014. doi: 10.1613/jair.4135. URL https://doi.org/10.1613/jair.4135.

[7] M. Charikar. Similarity estimation techniques from rounding algorithms. In J. H. Reif, editor, *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec,*

Canada, pages 380–388. ACM, 2002. doi: 10.1145/509907.509965. URL https://doi.org/10.1145/509907.509965.

[8] M. Faruqui, Y. Tsvetkov, D. Yogatama, C. Dyer, and N. A. Smith. Sparse overcomplete word vector representations. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers*, pages 1491–1500. The Association for Computer Linguistics, 2015. doi: 10.3115/v1/p15-1144. URL https://doi.org/10.3115/v1/p15-1144.

[9] L. Finkelstein, E. Gabrilovich, Y. Matias, E. Rivlin, Z. Solan, G. Wolfman, and E. Ruppin. Placing search in context: the concept revisited. *ACM Trans. Inf. Syst.*, 20(1):116–131, 2002. doi: 10.1145/503104.503110. URL https://doi.org/10.1145/503104.503110.

[10] Y. Gan, Y. Ge, C. Zhou, S. Su, Z. Xu, X. Xu, Q. Hui, X. Chen, Y. Wang, and Y. Shan. Binary Embedding-based Retrieval at Tencent. In *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, pages 4056–4067, 2023.

[11] J. Gao, D. He, X. Tan, T. Qin, L. Wang, and T. Liu. Representation Degeneration Problem in Training Natural Language Generation Models, 2019.

[12] D. Gerz, I. Vulic, F. Hill, R. Reichart, and A. Korhonen. Simverb-3500: A large-scale evaluation set of verb similarity. In J. Su, X. Carreras, and K. Duh, editors, *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP 2016, Austin, Texas, USA, November 1-4, 2016*, pages 2173–2182. The Association for Computational Linguistics, 2016. doi: 10.18653/v1/d16-1235. URL https://doi.org/10.18653/v1/d16-1235.

[13] Z. S. Harris. Distributional structure. *Word*, 10(2-3):146–162, 1954.

[14] S. Jastrzebski, D. Lesniak, and W. M. Czarnecki. How to evaluate word embeddings? on importance of data efficiency and simple supervised tasks. *CoRR*, abs/1702.02170, 2017. URL http://arxiv.org/abs/1702.02170.

[15] S. Jastrzebski, D. Lesniak, and W. M. Czarnecki. How to evaluate word embeddings? on importance of data efficiency and simple supervised tasks. *CoRR*, abs/1702.02170, 2017. URL http://arxiv.org/abs/1702.02170.

[16] A. Joulin, E. Grave, P. Bojanowski, M. Douze, H. Jégou, and T. Mikolov. Fasttext.zip: Compressing text classification models. *CoRR*, abs/1612.03651, 2016. URL http://arxiv.org/abs/1612.03651.

[17] A. Kessy, A. Lewin, and K. Strimmer. Optimal whitening and decorrelation. *The American Statistician*, 72(4):309–314, 2018.

[18] T. Luong, R. Socher, and C. D. Manning. Better word representations with recursive neural networks for morphology. In J. Hockenmaier and S. Riedel, editors, *Proceedings of the Seventeenth Conference on Computational Natural Language Learning, CoNLL 2013, Sofia, Bulgaria, August 8-9, 2013*, pages 104–113. ACL, 2013. URL https://aclanthology.org/W13-3512/.

[19] K. McRae, G. S. Cree, M. S. Seidenberg, and C. McNorgan. Semantic feature production norms for a large set of living and nonliving things. *Behavior research methods*, 37(4):547–559, 2005.

[20] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. In Y. Bengio and Y. LeCun, editors, *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, 2013. URL http://arxiv.org/abs/1301.3781.

[21] W. Mostard, L. Schomaker, and M. A. Wiering. Semantic preserving siamese autoencoder for binary quantization of word embeddings. In *NLPIR 2021: 5th International Conference on Natural Language Processing and Information Retrieval, Sanya, China, December 17 - 20, 2021*, pages 30–38. ACM, 2021. doi: 10.1145/3508230.3508235. URL https://doi.org/10.1145/3508230.3508235.

[22] J. Mu and P. Viswanath. All-but-the-top: Simple and effective postprocessing for word representations. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=HkuGJ3kCb.

[23] S. Navali, P. P. Sherki, R. Inturi, and V. Vala. Word embedding binarization with semantic information preservation. In D. Scott, N. Bel, and C. Zong, editors, *Proceedings of the 28th International Conference on Computational Linguistics, COLING 2020, Barcelona, Spain (Online), December 8-13, 2020*, pages 1256–1265. International Committee on Computational Linguistics, 2020. doi: 10.18653/v1/2020.coling-main.108. URL https://doi.org/10.18653/v1/2020.coling-main.108.

[24] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In A. Moschitti, B. Pang, and W. Daelemans, editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1532–1543. ACL, 2014. doi: 10.3115/v1/d14-1162. URL https://doi.org/10.3115/v1/d14-1162.

[25] V. Raunak, V. Gupta, and F. Metze. Effective dimensionality reduction for word embeddings. In I. Augenstein, S. Gella, S. Ruder, K. Kann, B. Can, J. Welbl, A. Conneau, X. Ren, and M. Rei, editors, *Proceedings of the 4th Workshop on Representation Learning for NLP, RepL4NLP@ACL 2019, Florence, Italy, August 2, 2019*, pages 235–243. Association for Computational Linguistics, 2019. doi: 10.18653/V1/W19-4328. URL https://doi.org/10.18653/v1/w19-4328.

[26] A. Rogers, I. Calixto, I. Vulić, N. Saphra, N. Kassner, O.-M. Camburu, T. Bansal, and V. Shwartz. Larger-Scale Transformers for Multilingual Masked Language Modeling. In *Proceedings of the 6th Workshop on Representation Learning for NLP (RepL4NLP)*, pages 29–33, 2021.

[27] S. Sasaki, B. Heinzerling, J. Suzuki, and K. Inui. Examining the effect of whitening on static and contextualized word embeddings. *Inf. Process. Manag.*, 60(3):103272, 2023. doi: 10.1016/J.IPM.2023.103272. URL https://doi.org/10.1016/j.ipm.2023.103272.

[28] D. Shen, Q. Su, P. Chapfuwa, W. Wang, G. Wang, R. Henao, and L. Carin. NASH: toward end-to-end neural architecture for generative semantic hashing. In I. Gurevych and Y. Miyao, editors, *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*, pages 2041–2050. Association for Computational Linguistics, 2018. doi: 10.18653/v1/P18-1190. URL https://aclanthology.org/P18-1190/.

[29] P. P. Sherki, S. Navali, R. Inturi, and V. Vala. Retaining semantic data in binarized word embedding. In *15th IEEE International Conference on Semantic Computing, ICSC 2021, Laguna Hills, CA, USA, January 27-29, 2021*, pages 130–133. IEEE, 2021. doi: 10.1109/ICSC50631.2021.00031. URL https://doi.org/10.1109/ICSC50631.2021.00031.

[30] R. Sokal and C. Michener. A statistical method for evaluating systematic relationships, 38 (university of kansas science bulletin). 1958.

[31] J. Su, J. Cao, W. Liu, and Y. Ou. Whitening Sentence Representations for Better Semantics and Faster Retrieval. *arXiv preprint arXiv:2103.15316*, 2021.

[32] N. Thakur, N. Reimers, and J. Lin. Injecting domain adaptation with learning-to-hash for effective and efficient zero-shot dense retrieval. *arXiv preprint arXiv:2205.11498*, 2022.

[33] J. Tissier, C. Gravier, and A. Habrard. Near-lossless binarization of word embeddings. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 7104–7111. AAAI Press, 2019. doi: 10.1609/aaai.v33i01.33017104. URL https://doi.org/10.1609/aaai.v33i01.33017104.

[34] J. Xu, P. Wang, G. Tian, B. Xu, J. Zhao, F. Wang, and H. Hao. Convolutional neural networks for text hashing. In Q. Yang and M. J. Wooldridge, editors, *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 1369–1375. AAAI Press, 2015. URL http://ijcai.org/Abstract/15/197.