Binary Encoded Word Mover's Distance

Christian Johnson
Universität Osnabrück, Institut für Kognitionswissenschaft
Wachsbleiche 27, 49074 Osnabrück, Germany
cjohnson@uni-osnabrueck.de

Abstract

Word Mover's Distance is a textual distance metric which calculates the minimum transport cost between two sets of word embeddings. This metric achieves impressive results on semantic similarity tasks, but is slow and difficult to scale due to the large number of floating point calculations. This paper demonstrates that by combining pre-existing lower bounds with binary encoded word vectors, the metric can be rendered highly efficient in terms of computation time and memory while maintaining competitive accuracy on several textual similarity benchmarks.

1 Introduction

A textual distance metric which can be used to accurately and quickly quantify the semantic dissimilarity between documents is useful for many natural language processing (NLP) tasks including text classification, document clustering, and document retrieval.

Word Mover's Distance (WMD) proposed by (Kusner et al., 2015) is a variant of the Earth Mover's Distance which measures the semantic distance between texts. Earth Mover's Distance is a well-studied transportation problem for measuring the distance between two probability distributions and was originally proposed for image retrieval applications (Rubner et al., 1998).

WMD leverages semantic distance information from pre-trained neural word embeddings to calculate a minimum transportation cost needed to 'move' the words from one text to those of another (Kusner et al., 2015). Let $c(i, j)$ represent the Euclidean distance transport cost between word embeddings i and j in documents d and d', respectively. The minimum cumulative (weighted) transport cost differentiating the two documents can then be summarized as,

$$\begin{aligned} &\min_{T \geq 0} \sum_{i,j=1}^{n} T_{ij}\, c(i,j) \\ \text{subject to:} \quad &\sum_{j=1}^{n} T_{ij} = d_i \quad \forall\, i \in \{1, ..., n\} \quad (1) \\ &\sum_{i=1}^{n} T_{ij} = d'_j \quad \forall\, j \in \{1, ..., n\} \end{aligned}$$

where T_{ij} represents *how much* of word embedding i travels to word embedding j. Solving this linear program given the above constraints provides the WMD. When evaluated via a *k* Nearest Neighbors (*k*NN) classification task on eight document classification datasets, WMD is demonstrated to outperform a variety of text similarity metrics such as the Okapi BM25, TF-IDF vector distance, and Latent Dirichlet Allocation (Kusner et al., 2015; Blei et al., 2003).

Per Equation 1, embedding distances must be computed for all pairs of unique tokens in the evaluated documents, scaling the time complexity for solving the underlying optimization problem by $O(p^3 \log p)$, where p denotes the number of unique words in a set of query documents. Given these demands, the memory and online computation time requirements for calculating WMD quickly diminish the potential to scale this metric in a production environment, especially as vocabulary size grows.

Kusner et al. and others have introduced several lower bounds to the WMD which reduce computations by relaxing constraints on the original transport problem, albeit with accompanying performance trade-offs. The *Relaxed Word Mover's Distance* (RWMD) is obtained by removing the second or third constraints from Equation 1, reducing the number of distance calculations given that all probability mass for each word in document d is moved to its most similar word in d'. On the aforementioned *k*NN classification task, the

Proceedings of the 7th Workshop on Representation Learning for NLP, pages 167 - 172

RWMD produces an average error of 0.56, compared to 0.42 for the original WMD (Kusner et al., 2015).

Werner et al. (Werner and Laber, 2019) propose another lower bound which involves a preprocessing phase for computing real-value distances between vocabulary items. During this phase, distances are computed between each word's r nearest neighbors and stored in a distance matrix M (Werner and Laber, 2019). These precomputed vector distances are retrieved when calculating transport costs $c(i, j)$ between words; if $c(i, j)$ cannot be found in M, a default maximum value c_{max} is used instead. The *Related Relaxed Word Mover's Distance* (Rel-RWMD), which combines the preprocessing phase with the relaxed conditions of the RWMD, is significantly faster to compute while suffering a modest drop in accuracy compared to the WMD on text similarity tasks.

Of final note is the Word Centroid Distance (WCD), which is computed as the distance between the unweighted average of each text's word embedding vectors (Kusner et al., 2015). Although not an alignment-based distance metric, WCD is a competitive metric which likewise leverages neural word embedding vectors.

One pain-point of the original WMD which motivates the use of lookup tables by Werner et al. is the large amount of floating point calculations needed to resolve the underlying transport problem. Floating point arithmetic CPU instructions are generally slower than integer or bit operations. Tissier et al. (Tissier et al., 2019) have proposed an autoencoding methodology for computing binary embeddings as encoded representations of an original, real-valued embedding space. With normalized XOR Hamming distance calculations, these binary vectors allow one to circumvent CPU-intensive floating point arithmetic while also reducing the working memory footprint of the word vectors by up to 97% (Tissier et al., 2019).

This paper contributes a methodology for integrating the binary vectors of Tissier et al. with the existing Rel-RWMD lower bound as the *Binary Encoded Word Mover's Distance* (BEWMD). It is hypothesized that by replacing the default c_{max} value of the Rel-RWMD with normalized binary vector Hamming distances, the proposed metric will produce a more accurate lower-bound to the original WMD while maintaining competitive computation times and, importantly, low memory requirements.

2 Methodology

The proposed lower bound is realized via neural network encoding of real-valued word vectors and manipulation of the Rel-RWMD calculation. An autoencoder network is trained with resort to a novel regularization parameter, while the Rel-RWMD calculation is modified to accommodate the binary-encoded vectors and render it bidirectional.

2.1 Autoencoder Architecture

An autoencoder is a neural network composed of an encoder and a decoder. The encoder forces an input into a representation which is then decoded into a reconstruction of the original input. By comparing the original and reconstructed inputs via a loss function which minimizes their difference, the autoencoder learns an encoded representation of the input data which is assumed to preserve the structure of the original embedding space. Per the methodology of (Tissier et al., 2019), a neural autoencoder with a specialized loss function is employed to produce the binary word vectors which support the proposed lower bound.

The autoencoder is implemented in Python 3.x via Tensorflow 2.x, consisting of an encoder with two hidden layers and a decoder with a single hidden layer. The encoder's input layer is the same size as a single real-valued vector (300 dimensions), while the subsequent hidden layers and the input to the decoder are adaptable to a desired encoded vector size.

2.2 Correlation Based Regularization

This paper's approach deviates from that of (Tissier et al., 2019) in that the loss function used to minimize the difference between inputs and reconstructed vectors contains an additional correlation based regularization term. A standard mean squared error loss function is insufficient to preserve the original vector space's semantic relationships in the encoded vectors, as the network will learn to discard 'too much similarity information from the original space in favor of the reconstruction' (Tissier et al., 2019). Tissier et al. exploit the encoder's weight matrix W, along with its transposition W^T and corresponding identity matrix I to derive an additional differentiable regularization term l_{reg} which is added to the mean square error loss function, as demonstrated in Equation 2.

$$l_{reg} = \frac{1}{2}\|W^T W - I\|^2 \quad (2)$$

where the loss function, including the mean squared error as l_{rec}, amounts to

$$L = l_{rec} + \lambda_{reg}\ l_{reg} \quad (3)$$

where λ_{reg} is a regularization hyperparameter between 1 and 4. Implementing this regularization parameter per the methodology of (Tissier et al., 2019), it was found that the adjusted loss function indeed improves training, but still results in compressed vectors which discard much of the semantic information contained in the original vector space. Thus, the loss function used in this paper includes an additional regularization parameter determined via correlation analysis of batch-pairwise distance matrices.

Let B denote a single training batch of size m. In all experiments, a batch size of $m = 75$ is used. Considering an $m \times n$ input batch matrix, where n denotes the original vector size of 300 dimensions, the corresponding, binarized code batch matrix will be an $m \times k$ matrix, where k denotes the size of the reduced vectors.

Computing the pairwise $m \times m$ Euclidean distance matrices B_X and B_Y for both the input and encoded vector spaces, respectively, one converts the matrices into ranks rB_X and rB_Y to compute the Spearman rank correlation coefficient r_s as

$$r_s = \frac{cov(rB_X, rB_Y)}{\sigma_{rB_X}\ \sigma_{rB_Y}} \quad (4)$$

where $cov(rB_X, rB_Y)$ denotes the covariance of the rank variables, while σ_{rB_X} and σ_{rB_Y} are the standard deviations of the rank variables. This calculation of the Spearman correlation coefficient is used because it allows for tie rank values, which is conceivable given the limited range of Hamming distances possible with the binary vectors. This coefficient is then integrated into the loss function as

$$L = l_{rec} + \lambda_{reg}(l_{reg} + r_s) \quad (5)$$

The additional regularizer r_s is summed with l_{reg} such that its contribution to the loss is also modulated by the hyperparameter λ_{reg}. Given that both regularization terms serve the same purpose with respect to the reconstruction loss (preserving distance information from the original vector space), it is sensible to combine them in this way and avoid the need for an additional regularization hyperparameter. With this new parameter, the adjusted objective function results in faster convergence and improves the binary embeddings' performance on downstream tasks, presumably by preserving more distance information from the original vector space. This improved loss function suggests that localized distance correlations are a valuable training objective.

2.3 Proposed Lower Bound Calculation

The resultant binary encoded word embeddings are integrated into a modified version of the RelRWMD, where they are used to compute a normalized Hamming distance in cases where the cosine distance between two word embeddings cannot be found in a precomputed lookup table (cache) C, defining the transport cost $c(i, j)$ between two words as:

$$c(i,j) = \begin{cases} 0, & \text{if } i = j \\ cosine(i,j) & \text{if } cosine(i,j) \in C \\ Hamming(i,j) & \text{otherwise} \end{cases} \quad (6)$$

As mentioned, the RWMD removes either the second or third constraint from Equation 1 to create the two relaxed solutions $\ell_1(d, d')$ and $\ell_2(d', d)$. These solutions require only the identification of each word's nearest neighbor from the other document in either of the two directions, given that each word's mass will be transferred entirely to its most similar word in the other document. Rather than opting for one of the two lower bounds (RelRWMD uses the maximum of the two), the solution presented in this paper computes both lower bounds and combines them via summation to render the distance calculation bidirectional. This summation approach is supported by the fusion methodology evaluated by (Hamann, 2018). Thus, the Binary Encoded Word Mover's Distance (BEWMD) is defined as

$$BEWMD = \frac{\alpha + \beta}{2}$$
$$\text{where:} \quad \alpha = \frac{1}{n} \cdot \min \sum_{i,j=1}^{n} c(i,j) \quad (7)$$
$$\beta = \frac{1}{n} \cdot \min \sum_{j,i=1}^{n} c(j,i)$$

Note that the summed costs are divided by the document length n, in order to constrain the metric to a value between 0 and 1; the sum of the pendant

unidirectional transport costs α and β is likewise divided by 2. Cosine distance is employed rather than Euclidean given that angular distance is theoretically more-resilient to variations in vector magnitude which are semantically-irrelevant artifacts of the vector-training process, as in the methodologies of (Mikolov et al., 2013; Zhang et al., 2018).

3 Results

Per the methodology of (Werner and Laber, 2019; Dai et al., 2015), BEWMD performance on a downstream semantic-similarity task is evaluated via the Stanford Triplets Wikipedia benchmark[1]. Metrics are assessed according to their ability to distinguish for a triplet of documents D^1, D^2, D^3, which pair of documents is most-related. Success for a single triplet is achieved if a metric computes the lowest distance score for the most-related document pair, namely D^1 and D^2. Triplet documents are Wikipedia articles which were preprocessed to remove non-alphanumeric characters and tokens which are not embedded under the attested models, in order to maintain computation time comparability across all tested metrics.

The autoencoder model used to produce the encoded vectors was fitted to pretrained word vectors from *FastText* (Bojanowski et al., 2016) (Common Crawl, 600B tokens)[2]. Consequently, all other distance metrics which rely on pretrained word embeddings used the same vectors. The autoencoder is fitted to the first 300,000 word vectors, which, given that the vectors are sorted by frequency, are assumed to be highly-representative of the full vector space. Training lasted for ten epochs with $\lambda_{reg} = 4$. Original real-valued vectors of 300 floating-point dimensions are encoded to 512-bit representations, per the register sizes of AVX-512 CPUs. Although the WMD was originally defined with Euclidean distances, cosine rather than Euclidean distance is used to compute the cost $c(i, j)$ for all metrics, so as to avoid spurious comparisons with BEWMD.

Tables 1, 2, and 3 demonstrate metric performance in terms of test error, offline, and online computation time, respectively. Table 4 documents the memory requirements of any vector models or lookup tables used during online computations. Online computation time is recorded as average time per evaluation iteration, or seconds required to evaluate a single triplet during online distance calculation. Offline computation time is defined as the amount of computation time in minutes to perform any one-time preprocessing such as the fitting of neural network models or calculation of lookup tables. Results are reported from Python implementations of the relevant metrics, where identical vector models or cache lookup tables are, where possible, used to maintain comparability across the metrics. The RWMD and Rel-RWMD lower bounds are interpreted as the maximum of the two possible relaxed solutions $\ell_1(d, d')$ and $\ell_2(d', d)$ (Kusner et al., 2015). The c_{max} value used when calculating Rel-RWMD is set to 0.8. All metrics are evaluated against the same 300 triplets. Calculations were performed on a machine with an Intel i7-8565U CPU and 8GB RAM.

BEWMD	*WCD*	*WMD*	*RWMD*	*Rel-RWMD*
0.393	0.436	**0.389**	0.594	0.641

Table 1: Triplets test error as a value between 0 and 1

BEWMD	*WCD*	*WMD*	*RWMD*	*Rel-RWMD*
283.50	0.00	0.00	0.00	88.85

Table 2: Offline computation time (min)

BEWMD	*WCD*	*WMD*	*RWMD*	*Rel-RWMD*
1.68	**0.006**	23.18	19.39	0.52

Table 3: Online computation time (sec/iter)

BEWMD	*WCD*	*WMD*	*RWMD*	*Rel-RWMD*
836	4409	4409	4409	**790**

Table 4: Online memory requirements (MB)

Additionally, this paper compares metric performance on three of the *k*NN benchmarks from (Kusner et al., 2015): *BBC Sport* contains sports articles between 2004-2005, *Classic* contains sets of sentences from academic papers, and *Ohsumed* is a collection of medical abstracts categorized by disease groups. Datasets are retrieved from the repository associated with the original paper[3]. Each dataset is subsampled to 100 randomly-selected samples (80 train, 20 test) across 5 sampling iterations. Table 5 shows the evaluated datasets and mean *k*NN

[1] http://cs.stanford.edu/ quocle/triplets-data.tar.gz
[2] https://fasttext.cc/docs/en/english-vectors.html
[3] https://github.com/mkusner/wmd

test classification error across all 5 random subsamplings for each metric. In all cases, a value of $k = 5$ is used. The same preprocessing techniques employed during the aforementioned Triplets evaluation were also used when performing the distance calculations which support the *k*NN evaluation. Although the reported results cannot address those from the original paper, they provide an additional indication of the proposed metric's performance relative to its peers.

	BEWMD	*WCD*	*WMD*	*RWMD*	*Rel-RWMD*
BBC Sport	0.12	**0.07**	0.16	0.23	0.45
Classic	**0.11**	0.25	0.14	0.21	0.42
Ohsumed	**0.58**	0.6	0.6	0.65	0.7

Table 5: Mean test error on kNN classification task

4 Discussion

Results on these limited tasks demonstrate that the proposed metric is competitive with the original WMD in terms of test error while offering respectable online computation time improvements and a lower memory footprint.

Comparing BEWMD Triplets performance to that of the other metrics, it can be ascertained that for this task it is the most accurate lower bound to the original WMD. Notably, the BEWMD offers a clear improvement upon the Rel-RWMD, assumedly by utilizing normalized binary vector Hamming distances as opposed to a default c_{max} value (Werner and Laber, 2019). Furthermore, BEWMD test error is highly-competitive with the original WMD, confirming the accuracy of the encoded vectors as compared to their real-valued counterparts.

Results for the three *k*NN benchmark tasks further substantiate BEWMD as a consistently-effective lower bound. BEWMD consistenly outperforms RWMD, Rel-RWMD, and even the original WMD. These results pose the hypothesis that the encoded vectors can offer a more-effective word representation for certain tasks. However, rigorous evaluation of the potential representation advantages offered by the encoded vectors is outside the scope of this paper.

Regarding computation and memory demands, BEWMD compares favorably with RWMD and Rel-RWMD, offering a reasonable computation time trade-off against the Rel-RWMD when BEWMD's improved test error is considered. All metrics which employ the real-valued vectors demand some 4GB of memory, while BEWMD and Rel-RWMD benefit from the reduced memory footprint of precomputed lookup tables and binary encoded vectors. Optimization of the BEWMD calculation so as to remove the need for precomputed lookup tables altogether remains a promising next step for further reducing the memory demands of the proposed metric.

It is worth noting the efficacy of WCD given that, in terms of Triplets test error alone, this metric outperforms all WMD lower bounds except the BEWMD. Furthermore, *k*NN benchmark evaluations place WCD consistently ahead of RWMD and Rel-RWMD, even surpassing WMD on the *BBC Sport* dataset. Although WCD requires more memory than either BEWMD or Rel-RWMD, its low computation time coupled with a relatively low test error on several evaluations pose this metric as a pragmatic alternative to WMD and the proposed metric for many use-cases.

These evaluations suggest that the BEWMD offers a balanced alternative to the original WMD, improving speed up to 14x while achieving competitive test error on several tasks. Furthermore, BEWMD's reduced memory footprint makes it suitable for low-resource compute environments. The demonstrated benefits of the BEWMD are, however, offset by its considerable offline computations, which demand, under the experimental parameters used in this paper, some 4 hours to compute both the encoded vectors and the nearest neighbor lookup table. For applications under constrained hardware where an upfront computation investment is tolerable, the BEWMD offers a consistently optimal lower bound to the original WMD.

As an ethical aside, it must be mentioned that optimized libraries for several of the lower bounds presented here outperform the evaluated BEWMD Python implementation in terms of online computation time, and it is not this paper's intent to evade these performance discrepencies. The evaluations presented here aim only to compare lower bound performance using comparable software implementations.

This paper suggests that binary encoded word vectors are valuable towards improving the scalability of WMD-derived distance metrics. The presented lower bound may be enhanced by optimizing the distance calculation in order to circumvent WMD approximation or the need for precomputed lookup tables.

References

David Blei, Andrew Ng, and Michael Jordan. 2003. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2016. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5.

Andrew Dai, Christopher Olah, and Quoc Le. 2015. Document embedding with paragraph vectors.

Felix Hamann. 2018. A neural embedding compressor for scalable document search. page 0.

Matt Kusner, Y. Sun, N.I. Kolkin, and Kilian Weinberger. 2015. From word embeddings to document distances. *Proceedings of the 32nd International Conference on Machine Learning (ICML 2015)*, pages 957–966.

Tomas Mikolov, G.s Corrado, Kai Chen, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. pages 1–12.

Yossi Rubner, Carlo Tomasi, and Leonidas Guibas. 1998. Metric for distributions with applications to image databases. pages 59–66.

Julien Tissier, Amaury Habrard, and Christophe Gravier. 2019. Near-lossless binarization of word embeddings. In *AAAI*.

Matheus Werner and Eduardo Laber. 2019. Speeding up word mover's distance and its variants via properties of distances between embeddings.

Ruqing Zhang, Jiafeng Guo, Yanyan Lan, Jun Xu, and Xueqi Cheng. 2018. *Aggregating Neural Word Embeddings for Document Representation*, pages 303–315.