MASTERARBEIT

Binarized Word Mover's Distance

Scalable Context-Sensitive Text Similarity with Binary Embedding Distances and Syntactic Dependency Information

Christian Johnson (981012)
December 6, 2020

Supervised by
Dr. Alexander Meier
Dr. Michael Franke

Abstract

This study explores improvement of the *Word Mover's Distance* textual distance measure's computational efficiency and accuracy. Computation time is reduced via approximate-nearest-neighbor-based intelligent caching of binary word embedding distances, while accuracy on downstream tasks is augmented via integration of syntactic dependency distance information. The proposed metric involves a novel procedure for dynamic caching of approximate-nearest-neighbor word-embedding-distance lookup tables, utilizing the lower memory footprint, binary word vectors of Tissier et al. (2018). Additionally, the word-context-insensitive metric is integrated with syntactic dependency distances derived from the Stanford Dependency hierarchy. The measure's performance is evaluated on the Stanford triplets task, where it achieves competitive results which validate the binary embedding approach, but not the syntactic dependency integration, suggesting several avenues for further investigation.

Contents

List of Figures

List of Tables

1 Introduction

A reliable distance metric for determining the degree of semantic similarity between two texts is useful for many downstream natural language processing (NLP) tasks including text classification, document clustering, and document retrieval. For such applications, it is necessary to develop a consistent metric which accurately reproduces human intuitions concerning the relative semantic distances between words, sentences, and documents.

A true metric satisfies the conditions of non-negativity, symmetry, and triangle inequality. A metric can be used to map points onto a metric space and can be inserted into other analytic approaches, such as cluster analysis, to readily accept new data without upsetting the validity of the metric.

This study explores the adaptation of a preexisting text distance metric in light of recent advances in word-representation compression and dependency parsing. Building on the work of Kusner et al. [16] and incorporating word representations developed by Tissier et al. [23], this study offers a more computationally-efficient means for computing a demonstratedly-effective metric. Simultaneously, this study addresses efforts by Chen et al. [6] in replacing part-of-speech (POS) information with syntactic dependency distance as a richer indicator of word context.

2 Theoretical Background

2.1 Text Distance

There exist two general approaches to the text distance task relevant to this study: statistical approaches and embedding approaches. For any approach, one aims to compute a distance or dissimilarity score between two documents (e.g. a single sentence or longer text) which accurately captures human intuitions about the degree of semantic agreement between the compared documents.

Statistical approaches are language and context independent, considering only the presence of identical tokens when comparing two documents. A minimal statistical approach involves term frequency hash values, through which one obtains a vector representation of a bag-of-words (BOW) document. Document vector representations are compared with vector distance metrics such as cosine distance, which then represent the dissimilarity of the two documents. On large datasets with many unique tokens, this approach suffers from the problem of *diagonally dominant matrices*, i.e. the phenomenon that the magnitude of the value of the diagonal entry for each row greatly exceeds the other values. This diagonal dominance, a form of data sparsity, has been shown to negatively effect downstream NLP tasks such as centroid-based document clustering [13].

The problem of diagonal dominance is addressed by best-matching (BM) ranking functions, which introduce document length normalization and additional free tuning parameters. Such ranking functions do not compute vector dissimilarity, but instead score documents according to statistical features of

queried documents, producing a document similarity ranking. The Okapi BM25 function is a popular, state-of-the-art retrieval function which scores a document D with respect to query Q, as adapted from [10] and summarized in Equations (1, 2):

$$\sum_{q \forall Q \cap D} \frac{(k_1 + 1) \cdot c(q, Q)}{k_1 + c(q, Q)} \cdot dtf(q, D) \cdot \log \frac{N + 1}{df(q) + 0.5} \tag{1}$$

$$dtf(q, D) = \frac{(k_2 + 1) \cdot c(q, D)}{k_2(1 - b + b\frac{|D|}{avdl}) + c(q, D)} \tag{2}$$

where $c(q, Q)$ and $c(q, D)$ denote the count of the term q in Q and D, respectively. N denotes the total number of documents and $df(q)$ is the document frequency of q. k_1, k_2, and b are free parameters which are generally set to independent constants [21] or adaptively defined on a per-term basis [17]. Equation (2) represents the term frequency normalization formula incorporating document length $|D|$, free parameter b, and average document length $avdl$. The implication of these equations is that terms with a low frequency in larger documents should not contribute to the similarity score. Via parameters k_1 and b, the influence of term frequency and document length, respectively, can be adjusted, with parameter values of 0 reducing the normalization term to 1, thereby imposing no normalization on the raw term frequency values.

One intuitive limitation of the aforementioned statistical approaches is insensitivity to words with related meanings, as terms with unique lexemes but similar semantic content cannot be accounted for. Consider the sentences *Obama speaks to the media in Illinois* and *The President greets the press in Chicago* (adapted from [16]). Although these sentences share no tokens, they intuitively convey the same information. Embedding approaches circumvent this phenomenon by introducing pretrained contextual word embeddings as the atomic constituent of document representations. Neural word embedding algorithms, such as *Word2vec* [18], identify co-occurrence patterns which efficiently represent word meaning in vector space [18]. With resort to embedding models trained on large corpora, semantic relationships between dissimilar tokens can be reliably incorporated into distance calculations.

One challenge for embedding approaches to document similarity is the normalization of concatenated word embeddings when comparing documents of different lengths. Although computing the vector similarity between pairs of embeddings is a trivial task, determining the contribution of individual token's vector representations to the computation of a document dissimilarity score is not. Most solutions to this problem leverage the demonstrated *compositionality* of the *Word2vec* vectors of Mikolov et al. [18], a behavior which allows for larger blocks of information to be represented by the manipulation of embedding constituents via vector operations. A naïve solution to the aforementioned challenge is the simple average of embedding vectors constituting a document. However, it has been shown that weighted averages involving smoothed term frequency [2], or term-frequency-inverse-document-frequency (TF-IDF) [26, 7] offer improved results on text similarity tasks.

Such compositional embedding approaches suffer from the dilution of individual token information in the averaged word embeddings. An alternative *alignment* approach sees the computation of distances between aligned word embeddings of compared documents, allowing that document similarity be computed on a token-to-token basis for more precise comparison of potentially-overlapping meanings. The Word Mover's Distance (WMD) evidences this embedding alignment approach and is the subject of the following section.

2.2 Word Mover's Distance

The WMD is a variation of the Earth Mover's Distance metric specialized to measuring 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 [22].

Proposed by [16], this specialized distance metric leverages semantic distance information from pretrained word embeddings to calculate a minimum transportation cost needed to 'move' the words from one text to those of another [16]. 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 transport cost differentiating the two documents can then be summarized as,

$$\begin{aligned} & \min_{T \geq 0} \sum_{i,j=1}^{n} T_{ij}\, c(i,j) \\ where: \quad & \sum_{j=1}^{n} T_{ij} = d_i \quad \forall\, i \in \{1, ..., n\} \\ & \sum_{i=1}^{n} T_{ij} = d'_j \quad \forall\, j \in \{1, ..., n\} \end{aligned} \tag{3}$$

In other words, by calculating the Euclidean distance between all pairs of words from documents d and d', one identifies the minimum distance for each word and consequently the document as an accumulation of words. On the k-nearest neighbor retrieval task, the WMD is demonstrated to outperform a variety of text similarity metrics such as the Okapi BM25, TF-IDF vector distance, and Latent Dirichlet Allocation [16, 3]. The WMD, however, carries several limitations including a lack of contextual information and the high computational cost of computing embedding distances.

Given that each document is represented as an unordered BOW, the WMD metric is insensitive to the syntactic structure of compared texts. Chen et al. [6] have partially-addressed this limitation by integrating an additional set of vectors encoding POS information for each token [6]. POS vectors are summed

with their corresponding word embeddings via a normalization term β, and the combined POS-WMD (PWMD) is computed as demonstrated in equation (3) using the summed word-POS vectors.

Chen et al.'s [6] integration of syntactic information via POS vectors superficially addresses the motivating limitations of the original WMD because this approach cannot distinguish sentences with identical lexical constituents playing different syntactic roles. Consider the sentence *The man saw the woman.* The original WMD cannot distinguish between this sentence and its reversed counterpart *The woman saw the man* because it considers only the unordered lists of tokens from the two documents. Likewise, the PWMD will consider the sentences identical because the POS are unchanged across the two sentences. As POS is a lexical classification scheme, the PWMD integrates only an additional attribute for resolving cases of lexical ambiguity, but it cannot compare syntactic structure per se [15].

An additional limitation of the WMD is its computational cost. Per Equation (3), embedding distances must be computed for all pairs of unique tokens in the evaluated documents, scaling the time complexity of solving the optimization problem by $O(p^3 \log p)$, where p denotes the number of unique words in a set of query documents. For large datasets used in downstream tasks such as document clustering, computation time quickly becomes a prohibiting factor. Kusner et al. [16] and others have introduced several cheaper lower-bounds to the WMD which improve computatational cost by imposing assumptions on the original transport problem.

In addition to a proposed *Word Centroid Distance*, which involves a distance between weighted average word vectors similar to the methodology proposed by [2], Kusner et al. [16] also propose the *Relaxed Word Mover's Distance* (RWMD) lower-bound. Obtained by removing the second and third constraints from Equation (3), the RWMD only requires computing the distance between each word's nearest-neighbor. Kusner et al. [16] also suggest that nearest-neighbor search can be optimized via out-of-the-box solutions for enhanced computation speed, and via precomputing pairwise distances between word vectors in the vocabulary. The RWMD expectedly performs worse than the complete WMD, with an average error on the aforementioned k-nearest-neighbors task of 0.56, compared to 0.42 for the WMD [16].

Although nearest-neighbor search vastly simplifies the computation demands of the WMD, one still needs to compute many distances during runtime which are preferably precomputed in a lookup table. Werner et al. [24] propose a modification to the WMD and RWMD by inserting a preprocessing phase for computing real-value distances between vocabulary items. During this phase, distances are computed between each word's r nearest neighbors in a distance matrix M while all other distances are set to a default maximum value [24]. Expectedly, this *Related Relaxed Word Mover's Distance* (Rel-RWMD) is significantly faster to compute while suffering a modest drop in accuracy compared to the WMD on text similarity tasks. However, given that this method depends on a precomputed lookup table, downstream performance is highly-dependent on the quality of token groupings and may suffer from inaccurate or approxi-

mated nearest-neighbor calculations.

Although superior in terms of computation time, the Rel-RWMD is limited by the free parameter r, which must be either experimentally-determined for a given dataset or estimated based on generalizations. This free parameter undermines one benefit of the WMD, namely that it is hyperparameter free, and imposes unnecessary assumptions on the distances between word embeddings which may be harmful to many downstream tasks. Werner et al. [24] also admit that precomputing the complete distance matrix M can still be exorbitantly expensive for large vocabularies, opting for a pre-clustering of word vectors to reduce the total number of preprocessing computations [24].

This study's approach builds on the innovations of [16] and [24] by addressing the problem of computation time with autoencoder-binarized word embeddings and construction of lookup tables held in memory during runtime, managed with an approximate-nearest-neighbor-based cache replacement policy. Similar to the approach of [24], this cache policy achieves the benefits of a related-word approach with dynamic memory management for enhanced speed, while also employing a custom partitioning scheme which circumvents the need for a preliminary cluster analysis. With this increased runtime, this study incorporates syntactic dependency information to hypotheticaly-improve evaluation accuracy by addressing the disambiguation limitations of the POS approach of [6]. It is hypothesized that the adjusted metric will surpass the original WMD in terms of computation time and evaluation accuracy, while remaining competitive with the metric's successors.

3 Methodology

3.1 Binarized Word Vectors

Computing the WMD for large datasets is slow primarily because of the need to calculate embedding distances between all combinations of word vectors. Additionally, these calculations involve real-value vectors and floating point arithmetic CPU instructions which are generally slower than integer or binary operations. Tissier et al. [23] have proposed a methodology for computing binary embeddings as compressed representations of an original, real-valued embedding space. These binary vectors circumvent CPU-intensive floating point arithmetic while also reducing the working memory footprint of vectors by up to 97% [23].

Given this study's goal to accelerate the WMD's computation time, the methodology of [23] is borrowed to produce binary word vectors which also maintain the crucial semantic relationships between words necessary for the WMD transport calculation. At the same time, the documented methodology deviates from that of [23] in the implementation loss function. The following section documents this approach while offering brief evaluations of the binary word vectors relevant to their integration into the proposed metric.

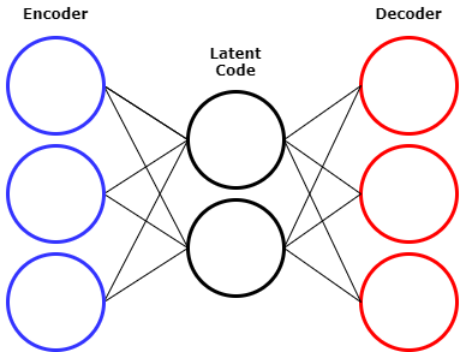

Figure 1: Diagram of autoencoder architecture

This project employs a neural autoencoder to produce the binary word vectors. An autoencoder is a neural network composed of an encoder and a decoder layer, designed to force an input into a lower-dimensional representation which is then reconstructed and compared with the input via a loss function which minimizes the difference between the input and reconstructed output. In this way, the autoencoder learns a compressed 'code' representation of the input data which is assumed to preserve the structure of the original.

The autoencoder is implemented in Python 3 via Tensorflow 2.0, 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 original vector (300 dimensions), while the subsequent hidden layers and the input to the decoder are the

desired reduced vector size. Figure (1) outlines the components of the autoencoder architecture, including the intermediate encoded representation which is extracted as the compressed vector used in downstream tasks.

This study's approach deviates from that of [23] in that the loss function used to minimize the difference between inputs and reconstructed vectors contains an additional 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' [23]. Tissier et al. [23] exploit the encoder's weight matrix W, along with its transposition W^T and corresponding identity matrix I to derive an additional differentiable term l_{reg} which is added to the mean square error loss function, as demonstrated in Equation (4).

$$l_{reg} = \frac{1}{2}\|W^T W - I\|^2 \tag{4}$$

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

$$L = l_{rec} + \lambda_{reg}\ l_{reg} \tag{5}$$

where λ_{reg} is a normalization coefficient between 1 and 4. Implementing this regularization parameter per the methodology of [23], 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 study includes an additional regularization parameter designed to minimize the difference in intra-batch spatial distances from the original and binary embedding spaces, respectively. The parameter value is 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. This project explores binarized vectors of sizes 256 and 512, in order to coordinate the bit size of the resultant vectors with CPU register sizes for enhanced bitwise computation speed under AVX2 (instruction set expansion for Intel and AMD processors).

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

$$r_s = \frac{cov(rB_X, rB_Y)}{\sigma_{rB_X}\ \sigma_{rB_Y}} \tag{6}$$

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 for the compressed, binary vectors. This coefficient is then integrated into the loss function as

$$L = l_{rec} + \lambda_{reg}(l_{reg} + r_s) \tag{7}$$

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.

The following tables summarize the performance and memory size of the compressed vectors produced by the autoencoder, to demonstrate their enhanced computational speed, reduced size, and performance on a sample NLP task. The autoencoder model was fitted to pretrained word vectors from *Word2vec* [18] (Google News, 3M tokens)[1], *FastText* [4] (Common Crawl, 600B tokens)[2], and *GloVe* [19] (Common Crawl, 840B tokens) [3]. These embedding models were selected given those utilized in [16, 24], as they are commonly-employed static embeddings. The specific pretrained models were chosen so as to be the largest trained on the most comparable corpora across the three algorithms. For all experiments, the autoencoder is fitted to the first 300,000 word vectors, which, given that the vectors are sorted by frequency, are assumed to be representative of the full vector space. All trainings lasted for ten epochs with $\lambda_{reg} = 4$.

Binary embeddings produced after fitting to the different pretrained vectors are first evaluated on a word similarity task, where human word similarity evaluations are compared to the Hamming distances between corresponding word vectors via the Spearman rank correlation coefficient. The similarity datasets used are SimVerb and WordSim [12, 11]. For each model is summarized the Spearman coefficient as a value between zero and one for the real-valued vectors and vectors of size 256 or 512 bits.

	word2vec			FastText			GloVe		
Data	*raw*	*256*	*512*	*raw*	*256*	*512*	*raw*	*256*	*512*
SimVerb	**0.36**	0.21	0.27	**0.37**	0.21	0.27	**.28**	0.17	0.24
WordSim353	**0.70**	0.46	0.50	**0.78**	0.46	0.58	**.74**	0.49	0.58

Table 1: Spearman rank correlation coefficients on word similarity task for compressed vectors

The compressed vector representations achieve moderately comparable performance increasing as the number of dimensions increases. As expected, the

[1] https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit
[2] https://fasttext.cc/docs/en/english-vectors.html
[3] https://nlp.stanford.edu/projects/glove/

real-valued vectors perform the best. Admitting that the original *Word2vec* vectors occupy around 10GB of memory, while the compressed vectors require only 800MB and 1.5GB for the 256 and 512 bit vectors, respectively, it is notable that the compressed vectors achieve up to 75 percent of the original vector's performance while requiring only 10 percent of the original memory space. Furthermore, computations with the compressed vectors are up to 3x faster. To load all vectors into memory and compute the evaluation coefficients for the *FastText*-fitted model requires 232.145s while the corresponding 256 bit vectors require only 76.376s.

As a further, qualitative evaluation of compressed vector performance relative to the real-valued vectors, Table (2) presents the $k = 5$ approximate nearest neighbors (ANNs) for a set of arbitrary test words when queried on the real-valued and compressed vectors. The displayed results were queried for *GloVe* real-value and 512 bit compressed vectors. *GloVe* vectors were selected given their relatively high-performance on the WordSim353 task for all compression sizes. The results suggest that the binary vectors preserve enough of the original embedding space's semantic information to justify their efficacy commensurate with the aforementioned memory and computation-time improvements.

Word	ANN Raw	ANN Binary
'car'	car, cars, vehicle automobile, truck	car, cars, vehicle truck, car
'apple'	apple, apples, fruit blueberry, strawberry	apple, cherry, pear tangerine, jack

Table 2: Approximate nearest neighbor (ANN) query for sample words

In the next phase of the project, the compressed vectors are used to compute multiple binary-space partitionings which inform a lookup table of real-valued ANN vector distances. For this task, binary vectors are required only for computing related word bins intended to reduce the total number of real-valued lookup table computations. Given that the binary vectors are demonstrated to maintain a majority of the original embedding space's distance information, these light-weight vectors offer a suitable stand-in for real-valued embeddings. The following section documents the methodology for computing the binary space partitionings, lookup tables, and the relevant cache removal policy developed to optimize the WMD calculations.

3.2 Nearest-Neighbor Caching

For computing the so-called Binarized Word Mover's Distance (BWMD), it is most practical to retrieve inter-word vector distances via precomputed lookup tables rather than compute values in real time. However, storing a full set of pairwise distances for N word vectors requires $N^2 \cdot (8 + 4)$ bytes (8 bytes for a 64bit address and 4 bytes for a 32bit float value) stored in working memory. As noted by Hamann [14], "Even when selecting the 40K most frequent words, this would lead to an additional footprint of around 20 gigabytes with rarely or never used information" [14].

Accordingly, the approach adopted in this study sees the repeated partitioning of the full vector space such that related words are identified. By computing distances only among related words as ANNs, it is assumed that the problem of inconsequential distances referenced by [14] is circumvented while also greatly increasing the capacity to reduce the total size of the lookup tables. Assuming the 40K vector scenario with $k = 20$ approximate nearest neighbors for each token, the total memory footprint becomes a trivial $40000 \cdot 20 \cdot (8 + 4) \approx 9600000$ bytes or less than ten megabytes.

As mentioned, the binary vectors described in Section 3.1 are employed towards identifying related words. Assuming compressed vectors capture the general structure of the original vector space, floating point cosine distances computed between corresponding real-value vectors will subsequently capture the fine-grained distances necessary to ensure an accurate distance metric calculation.

Although there exist several open-source libraries for efficient ANN search in high-dimensional space, such libraries could not be employed in this project because of a lack of support for Hamming distance computation and otherwise unsatisfactory nearest-neighbor queries in binary space above low query sizes. Thus, it was necessary to develop a Hamming space partitioning algorithm for ANN grouping. The algorithm is summarized below.

Algorithm 1: Hamming Space Partitioning

```
initialize K = 100, k = 2^I
 for int in range 100 do
 |  while n partitions < k do
 |  |  Randomly bisect into two partitions;
 |  |  if size partition < N / k then
 |  |  |  Store and continue;
 |  |  else
 |  |  |  Continue to bisect;
 |  |  end
 |  end
 |  Store all resulting partitions.
end
Result: K = 100 Hamming Space Partitionings
```

The resultant $K = 100$ partitionings are then unified for each token. Assuming that duplicated tokens across multiple randomly-seeded partitionings indicate semantic relationships between tokens, the full list of combined partition members for each token is first sorted by number of occurrences, and then further sorted by real-value cosine distance before the top $k = 20$ nearest neighbors to each query token are extracted. This iterative sorting process is intended to reduce memory load during the partitioning consolidation, as the full size of all partitioning iteration data and word vectors will collectively exceed 10 gigabytes for the full vector space. The sorted cosine distances, assumed to be the most accurate representation of inter-token distances, are then stored for each unique token to be retrieved during runtime.

To accelerate working memory management in coordination with the computed tables, it is necessary to devise an intelligent cache replacement policy whereby the most-relevant distances are made available at the expense of others. If the number of stored ANNs is to exceed $k = 20$, this policy is also necessary to accommodate working memory limitations on some devices. In this case, cache-relevance is defined according to a Least-Recently-Used (LRU) replacement policy, which removes from the in-memory cache the least-recently-used table above a predefined cache capacity. In other words, when computing the BWMD the precomputed ANN distance tables are sequentially loaded into the cache as-requested or shifted 'forward' in the cache order if already loaded. Once the cache size exceeds its capacity the 'oldest' table is replaced. The assumption behind this approach is that a corpus of documents typically revolves around semantic themes which will be represented by a subset of the available distance tables, given that the tables represent semantically-grouped sets of words. Assuming that the Hamming space partitioning accurately groups related words into groups of thematically-exhaustive size, the replacement policy should identify an optimal subset of distance tables which are sufficient to determine the BWMD for a given corpus while reducing the total memory footprint of the calculations.

3.3 Binarized Word Mover's Distance

The proposed distance metric closely resembles that of [24] and the so-called *Related Relaxed Word Mover's Distance*, albeit with minor additional normalizations and lower-bound assumptions derived from the aforementioned caching strategy and binary vectors. Let $c(i, j)$ represent the cosine distance transport cost between word embeddings i and j in documents d and d' of length n, respectively. The minimum cumulative transport cost differentiating the two documents can then be summarized as,

$$
\begin{aligned}
BWMD &= \frac{\alpha + \beta}{2} \\
where: \qquad \alpha &= \min \frac{1}{n} \sum_{i,j=1}^{n} c(i,j)(1-k) + \lambda(i,j)k \\
\beta &= \min \frac{1}{n} \sum_{j,i=1}^{n} c(j,i)(1-k) + \lambda(j,i)k
\end{aligned}
\tag{8}
$$

By iteratively identifying the nearest word from the other document, summing these tokens' distance values, and normalizing the summed distances by document length n, we obtain the minimum cumulative distance between the two documents. As opposed to the original WMD, all mass from matched tokens in d and d' are transferred, as opposed to a weighted transport derived from the word frequency flow matrix [16]. As a consequence of this lower-bound simplification, the cumulative distance is computed in both directions to render the metric bidirectional, per the fusion methodology of [14]. 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 [18, 25].

Given that distances are accessed from precomputed lookup tables containing distances only for approximated-nearest-neighbor word vectors, in cases where the precomputed cosine distance between two words does not exist, the Hamming distance between the two tokens' codes is returned. Per the methodology of [24], this default serves to give highest accuracy to comparisons between related words, given that the minimum distance transport problem aims to identify the closest word pairs from documents d and d'. Because this project's approach employs two sets of corresponding word vectors, rather than returning an arbitrary default value in cases where a precomputed cache cannot be used, Hamming distance acts as a more-accurate approximation of token-to-token distance while keeping computation time low. To avoid low distances between documents with similar sets of stop-words (particles such as *and*, *to*, *the*, etc.), such words are ignored during computation of the score. Stop-words are not removed during preprocessing as they are integral to the dependency-tagging required for computing the syntactic dependency distance described in the following paragraphs.

The value $\lambda(i,j)$ in Equation (8) denotes the syntactic dependency distance between words i and j which correspond to the minimum-transport word vectors identified via $c(i,j)$, with k as an arbitrary weighting coefficient between 0 and 0.5 used to negotiate the contribution of the syntactic distance information towards the overall distance score. In this study's implementation k equals 0.25. As mentioned in Section 2.2, syntax information is integrated as a proxy for word context, deviating from the analogous methodology of [6]. Syntactic distance is defined via graph-theoretic properties of the Stanford Dependen-

cies representation, which, as noted by [20], stand in a hierarchy that can be leveraged to compute a normalized path length between two dependency tags [20, 9].

For optimal dependency parsing, it is most-efficient to employ the SpaCy dependency parser, which is demonstrated to outperform Stanford's in terms of speed by 2.6x [1, 9]. However, because the SpaCy parser utilizes a unique tagset, these tags must first be converted to those in the Stanford hierarchy. For a full list of dependency-tag correspondences, please consult Appendix A. The dependency hierarchy is used to construct a network graph with nodes and edges replicating the hierarchical dependency relationships. From this graph is then computed the pairwise, normalized path lengths for all dependency combinations, which are then stored in an additional lookup table accessed when computing the BWMD. The normalized dependency distance is defined according to Leacock and Chodorow's normalized path length, where the similarity between two graph nodes c_i and c_j is defined as,

$$sim(c_i, c_j) = -\log \frac{path(c_i, c_j)}{2 \cdot maxdepth} \tag{9}$$

where $path(c_i, c_j)$ denotes the shortest path between the two nodes and $maxdepth$ the maximum depth of the hierarchy [5]. As the maximum value of the similarity score depends on the depth of the assessed hierarchy, the similarity scores for all combinations of dependency nodes are rescaled between 0 and 1 to maintain analogy with the cosine distances used in the BWMD. The scaled similarity scores are subtracted from 1 to convert them into distances, rendering $\lambda(i, j) = 1 - sim(c_i, c_j)$.

Superficial, qualitative evaluation of the integrated syntactic distance information confirms the assumption that this information is able to accurately distinguish sentences with similar semantic content yet divergent syntactic structures. When computing the BWMD for the aforementioned sentences *The man saw the woman* and *The woman saw the man*, the distance is expectedly 0.0 when ignoring syntactic information as the set of tokens in the two sentences is identical. However, when integrating syntactic dependency distance the score increases to 0.07, confirming that the approach is able to introduce additional distance information which distinguishes the divergent roles of the sets of tokens.

4 Evaluation

4.1 Procedure

Per the methodology of [24, 8], BWMD performance on a downstream semantic-similarity-related task is evaluated via the Stanford Triplets Wikipedia dataset[4], where distances are evaluated according to their ability to distinguish for a triplet of documents D^1, D^2, D^3, which of the two documents is most related. Success for a single triplet pair is achieved if the metric computes the lowest 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 with other metrics.

4.2 Results

Tables 3, 4, and 5 demonstrate performance for different vector models and as compared to other metrics in terms of test error and computation time, respectively. Computation time is recorded as average time per iteration, i.e. minutes required to evaluate a single triplet. In all cases the 512-bit compressed vectors are used when computing the BWMD. The label *512+S* refers to BWMD calculations with syntactic dependency information. Comparison 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.

	word2vec		FastText		GloVe	
Measure	*512*	*512+S*	*512*	*512+S*	*512*	*512+S*
Test Error	0.413	0.436	**0.396**	0.403	0.433	0.423
Compute Time	**0.0139**	0.0232	0.0159	0.027	0.0156	0.0263

Table 3: Test error (%) and compute time (min/iter) on Wikipedia Triplets task

BWMD	*BWMD+S*	*WCD*	*WMD*	*RWMD*	*Rel-RWMD*
0.396	0.403	0.44	**0.393**	0.611	0.621

Table 4: Test error (%) as compared to other metrics

[4] http://cs.stanford.edu/ quocle/triplets-data.tar.gz

BWMD	*BWMD+S*	*WCD*	*WMD*	*RWMD*	*Rel-RWMD*
0.014	0.024	**0.0001**	0.198	0.038	0.0037

Table 5: Computation time (min/iter) as compared to other metrics

5 Discussion

Results on this limited task demonstrate that the proposed metric is competitive with the original WMD in terms of test accuracy while offering respectable computation time improvements. BWMD results for the different vector models suggest that the employed model itself does not greatly influence downstream performance, although the *FastText* model's vectors appear particularly effective on this dataset, perhaps due to the representation of Wikipedia articles in its training data. Overall, BWMD test error is comparable to the original WMD while surpassing other metrics which prioritize speed.

As expected, computing the BWMD with syntactic dependency information increases computation time as a result of the dependency parsing. However, the compounded time with syntax does not render the metric egregiously slow. Unfortunately, the hypothesis that syntactic dependency information enhances test accuracy does not obtain; BWMD test error with syntax information offers inferior results. It is likely that, for this dataset and task, syntactic distance rather introduces error-inducing noise. Notably, test error for *GloVe* vectors appears improved as a result of syntax information, but given that the difference is so small, it is likely that this improvement is attributable to random chance. Although this project's evaluations cannot validate the inclusion of syntactic dependency information, it remains to be seen whether or not this additional information is performant for other datasets where structural distinctions are more relevant to the differentiation of documents. Further evaluation of the performance of syntactic dependency distance remains an avenue of further study outside the scope of this project.

Comparing the BWMD's performance to that of other relevant metrics, it can be ascertained that it is the most accurate lower-bound to the original WMD. Notably, the BWMD offers a clear improvement upon the *Rel-RWMD* by utilizing binary vector Hamming distances as opposed to a default *cmax* value [24]. These Hamming distance computations come with a demonstrable cost: the BWMD is clearly slower than the *Rel-RWMD*. Nevertheless, these evaluations suggest that the BWMD offers a balanced alternative to the original WMD, improving speed up to 14x while achieving comparable test accuracy.

Of additional note is the memory footprint of this project's binary vector approach. All other word-embedding-based metrics require up to ten gigabytes of high dimensional floating point vectors to be stored on disk. When computing cosine distances for the WCD or WMD, these vectors must be loaded into the in-memory cache, lest a value be impossible to compute. Combining the intelligent LRU cache with the compressed binary vectors, in-use memory can be reduced

to less than one gigabyte while still maintaining high speed and accuracy. On low-memory devices such as mobile devices, this metric offers performant results on limited hardware.

5.1 Limitations

Although results suggest that many of the proposed metric's hypothesized improvements do obtain, the study itself presents several limitations.

While this study was able to reproduce the autoencoder vector binarization methodology of Tissier et al. [23], the fitted vectors do not reproduce the text similarity task results seen in the original study. This is likely due to slight methodological differences resulting from the Tensorflow implementation and modified training hyperparameters. Additional training with a hyperparameter grid search may help to further optimize the encoder's weight matrices and improve downstream performance.

Evaluations on the Stanford Wikipedia Triplets task suggest competitive performance and accuracy, but cannot exhaustively confirm the metric's superiority given the limited scope of the evaluations and insignificant difference between some results. Further evaluations on additional downstream tasks and larger datasets will enhance the power of this analysis.

Although considerable effort was devoted to rendering the metric memory efficient and fast, further work can be done to improve performance. Currently, binary vectors are stored in text files which must be parsed at runtime; compressing these files will reduce required disk space and improve loading times. Additionally, the metric may be enhanced by moving computations to the GPU or introducing compiled binaries for computationally-intensive operations, such as the Hamming distance calculations, which will benefit from compilation for hardware-specific CPU instructions.

5.2 Outlook

In addition to addressing the weaknesses mentioned in the previous section, the project's results suggest several possible directions and next steps which will build upon the work done here.

One interesting direction is further experimentation with the autoencoder compression methodology and its applications. Given the high-quality compressed vectors of [23] and their successful application in this project, it is likely that further research into the potential of this compression methodology will render compressed word vector applications more accessible for a wider range of hardware. It may also be advantageous to explore compression formats utilizing different datatypes; under AVX2-512 modern CPUs now include instructions for int8 operations of comparable speed to bitwise operators, potentiating smaller file sizes and alternatives to Hamming distance.

Although the ANN cache strategy has produced reliable results on this project's limited evaluations, it is obvious that the quality of the metric is highly-dependent on the content of the precomputed lookup table. When comparing

the results of the *Rel-RWMD* and *BWMD*, which utilize the same precomputed lookup table, it can be assumed that the drastic performance increase of the *BWMD* is attributable to the inclusion of Hamming distances as opposed to a default maximum value. By improving the relevance of words grouped within the precomputed tables, the metric's speed can be further increased while rendering its performance theoretically more-comparable to the original WMD. Admittedly, an ANN approach is unlikely to be the most efficient or accurate solution to the construction of a reliable lookup table, and further research into an effective word embedding grouping and caching strategy remains an avenue for further development. For example, an efficient consensus clustering methodology involving binary-vector-space subsampling may lead to more-exhaustive related word bins while maintaining reasonable lookup table compute times.

Additionally, although offering a demonstrably accurate and memory-efficient alternative to existing metrics, this study's Python implementation[5] cannot compete with existing C and Java implementations of other metrics in terms of computation speed. Porting the current implementation to a compiled language promises further performance improvements.

Finally, further evaluations on diverse tasks will render the metric more interpretable and its assessment more robust. Although this project's evaluations suggest the metric is competitive towards distinguishing related documents, it remains to be seen if it remains performant in the context of more-recognizable tasks such as document clustering and text classification, and whether these results can be generalized across a variety of datasets. Notably, although syntactic dependency information was inconsequential for evaluating against the Wikipedia triplets task, it may be that this information is useful in other contexts, e.g. sentiment classification of social media comments.

5.3 Conclusion

This project has demonstrated that the application of binarized word vectors offers a performant enhancement to the Word Mover's Distance. Improved autoencoder training resulting from an additional integrable coefficient suggests that localized distance correlations are advantageous to compression objective functions. Although limited, evaluations suggest that further research with additional datasets will improve the metric's interpretability and potentially validate the presently-rejected hypothesis that syntactic dependency information will improve accuracy on downstream tasks.

[5] `https://github.com/christianj6/binarized-word-movers-distance`

References

[1] Spacy: Facts & figures. https://spacy.io/usage/facts-figures. Accessed: 2020-09-30.

[2] S. Arora, Y. Liang, and T. Ma. A simple but tough-to-beat baseline for sentence embeddings. 11 2016.

[3] D. Blei, A. Ng, and M. Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, 05 2003.

[4] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5, 07 2016.

[5] A. Budanitsky and G. Hirst. Evaluating wordnet-based measures of lexical semantic relatedness. *Computational Linguistics*, 32:13â47, 03 2006.

[6] X. Chen, L. Bai, D. Wang, and J. Shi. Improve word mover's distance with part-of-speech tagging. pages 3722–3728, 08 2018.

[7] J. Corrêa, V. Marinho, and L. Santos. Nilc-usp at semeval-2017 task 4: A multi-view ensemble for twitter sentiment analysis. 04 2017.

[8] A. Dai, C. Olah, and Q. Le. Document embedding with paragraph vectors. 07 2015.

[9] M.-C. De Marneffe and C. D. Manning. Stanford typed dependencies manual. Technical report, 2008.

[10] H. Fang, T. Tao, and C. Zhai. A formal study of information retrieval heuristics. pages 49–56, 01 2004.

[11] L. Finkelstein, E. Gabrilovich, Y. Matias, E. Rivlin, Z. Solan, G. Wolfman, and E. Ruppin. Placing search in context: The concept revisited. In *Proceedings of the 10th International Conference on World Wide Web*, WWW '01, page 406–414, New York, NY, USA, 2001. Association for Computing Machinery.

[12] D. Gerz, I. Vulic, F. Hill, R. Reichart, and A. Korhonen. Simverb-3500: A large-scale evaluation set of verb similarity. *CoRR*, abs/1608.00869, 2016.

[13] D. Greene and P. Cunningham. Practical solutions to the problem of diagonal dominance in kernel document clustering. volume 148, pages 377–384, 06 2006.

[14] F. Hamann. A neural embedding compressor for scalable document search. page 0, 10 2018.

[15] M. Haspelmath. *Word Classes and Parts of Speech*, pages 16538–16545. 12 2001.

[16] M. Kusner, Y. Sun, N. Kolkin, and K. Weinberger. From word embeddings to document distances. *Proceedings of the 32nd International Conference on Machine Learning (ICML 2015)*, pages 957–966, 01 2015.

[17] Y. Lv and C. Zhai. Adaptive term frequency normalization for bm25. pages 1985–1988, 10 2011.

[18] T. Mikolov, G. Corrado, K. Chen, and J. Dean. Efficient estimation of word representations in vector space. pages 1–12, 01 2013.

[19] J. Pennington, R. Socher, and C. Manning. Glove: Global vectors for word representation. volume 14, pages 1532–1543, 01 2014.

[20] T. Proisl, S. Evert, P. Greiner, and B. Kabashi. Semantiklue: Robust semantic similarity at multiple levels using maximum weight matching. In *In Proceedings of SemEval 2014: International Workshop on Semantic Evaluation*, 2014.

[21] S. Robertson and S. Walker. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. pages 232–241, 01 1994.

[22] Y. Rubner, C. Tomasi, and L. Guibas. Metric for distributions with applications to image databases. pages 59–66, 02 1998.

[23] J. Tissier, A. Habrard, and C. Gravier. Near-lossless binarization of word embeddings. In *AAAI*, 2019.

[24] M. Werner and E. Laber. Speeding up word mover's distance and its variants via properties of distances between embeddings. 12 2019.

[25] R. Zhang, J. Guo, Y. Lan, J. Xu, and X. Cheng. *Aggregating Neural Word Embeddings for Document Representation*, pages 303–315. 01 2018.

[26] J. Zhao, M. Lan, and J. Tian. Ecnu: Using traditional similarity measurements and word embedding for semantic textual similarity estimation. pages 117–122, 01 2015.

Appendices

A Dependency Tag Correspondences

Below are summarized the corresponding dependency tags fromm the SpaCy and Stanford tagsets, used to create a hierarchy of SpaCy tags which can accurately represent distances between syntactic roles. Correspondences which are potentially-contentious are marked in bold.

SpaCy	Stanford	Description
acl	**det**	clausal noun modifier
acomp	acomp	adjective complement
advcl	advcl	adverbial clause modifier
advmod	advmod	adverbial modifier
agent	agent	agent
amod	amod	adjectival modifier
appos	appos	appositional modifier
attr	**dobj**	attribute
aux	aux	auxiliary
auxpass	auxpass	auxiliary (passive)
case	**possessive**	case marking
cc	cc	coordinating conjunction
ccomp	ccomp	clausal complement
compound	nn	compound
conj	conj	conjunct
cop	cop	copula
csubj	csubj	clausal subject
csubjpass	csubjpass	clausal subject (passive)
dative	**iobj**	dative
dep	dep	unclassified dependent
det	det	determiner
dobj	dobj	direct object
expl	expl	expletive
intj	**punct**	interjection

SpaCy	Stanford	Description
mark	mark	marker
meta	**dep**	meta modifier
neg	neg	negation modifier
nn	nn	noun compound modifier
nounmod	**nn**	modifier of nominal
npmod	npadvmod	noun phrase as adv modifier
nsubj	nsubj	nominal subject
nsubjpass	nsubjpass	nominal subject (passive)
nummod	num	numeric modifier
oprd	**dobj**	object predicate
obj	obj	object
obl	**dobj**	oblique nominal
parataxis	parataxis	parataxis
pcomp	**acomp**	prepositional complement
pobj	pobj	prepositional object
poss	poss	possession modifier
preconj	preconj	pre-correlative conjunction
prep	prep	prepositional modifier
prt	**punct**	particle
punct	punct	punctuation
quantmod	quantmod	modifier of quantifier
relcl	rcmod	relative clause modifier
root	root	root
xcomp	xcomp	open clausal complement