Pattern Recognition Letters
journal homepage: www.elsevier.com

Representation learning for very short texts using weighted word embedding aggregation

Cedric De Boom[a], Steven Van Canneyt[a], Thomas Demeester[a], Bart Dhoedt[a]

[a]*Ghent University – iMinds, Department of Information Technology, Technologiepark 15, 9052 Zwijnaarde, Belgium*

ABSTRACT

Short text messages such as tweets are very noisy and sparse in their use of vocabulary. Traditional textual representations, such as tf-idf, have difficulty grasping the semantic meaning of such texts, which is important in applications such as event detection, opinion mining, news recommendation, etc. We constructed a method based on semantic word embeddings and frequency information to arrive at low-dimensional representations for short texts designed to capture semantic similarity. For this purpose we designed a weight-based model and a learning procedure based on a novel median-based loss function. This paper discusses the details of our model and the optimization methods, together with the experimental results on both Wikipedia and Twitter data. We find that our method outperforms the baseline approaches in the experiments, and that it generalizes well on different word embeddings without retraining. Our method is therefore capable of retaining most of the semantic information in the text, and is applicable out-of-the-box.

1. Introduction

Short pieces of texts reach us every day through the use of social media such as Twitter, newspaper headlines, and texting. Especially on social media, millions of such short texts are sent every day, and it quickly becomes a daunting task to find similar messages among them, which is at the core of applications such as event detection (De Boom et al. (2015b)), news recommendation (Jonnalagedda and Gauch (2013)), etc.

In this paper we address the issue of finding an effective vector representation for a very short text fragment. By effective we mean that the representation should grasp most of the semantic information in that fragment. For this we use semantic word embeddings to represent individual words, and we learn how to weigh every word in the text through the use of tf-idf (term frequency - inverse document frequency) information to arrive at an overall representation of the fragment.

These representations will be evaluated through a semantic similarity task. It is therefore important to point out that textual similarity can be achieved on different levels. At the most strict level, the similarity measure between two texts is often defined as being (near) paraphrases. In a more relaxed setting one is interested in topic- and subject-related texts. For example, if a sentence is about the release of a new Star Wars episode and another about Darth Vader, they will be dissimilar in the most strict sense, although they share the same underlying subject. In this paper we focus on the broader concept of topic-based semantic similarity, as this is often applicable in the already mentioned use cases of event detection and recommendation.

Our main contributions are threefold. First, we construct a technique to calculate effective text representations by weighing word embeddings, for both fixed- and variable-length texts. Second, we devise a novel median-based loss function to be used in the context of minibatch learning to mitigate the negative effect of outliers. Finally we create a dataset of semantically related and non-related pairs of text from both Wikipedia and Twitter, on which the proposed techniques are evaluated.

We will show that our technique outperforms most of the baselines in a semantic similarity task. We will also demonstrate that our technique is independent of the word embeddings being used, so that the technique is directly applicable and thus does not require additional model training when used in different contexts, in contrast to most state-of-the art techniques.

In the next section, we start with a summary of the related work, and our own methodology will be devised in Section 3. Next we explain how data is collected in Section 4, after which we discuss our experimental results in Section 5.

e-mail: cedric.deboom@ugent.be (Cedric De Boom)

2. Related work

In this work we use so-called word embeddings as a basic building block to construct text representations. Such an embedding is a distributed vector representation of a single word in a fixed-dimensional semantic space, as opposed to term tf-idf vectors, in which a word is represented by a one-hot vector (Achananuparp et al. (2008); Manning et al. (2009)). A word's term frequency (tf) is the number of times the word occurs in the considered document, and a word's document frequency (df) is the number of documents in the considered corpus that contain that word. Its (smoothed) inverse document frequency (idf) is defined as:

$$\text{idf} \triangleq \log \frac{N}{1 + \text{df}}, \tag{1}$$

in which N is the number of documents in the corpus (Manning et al. (2009)). A tf-idf-based similarity measure is based on exact word overlap. As texts become smaller in length, however, the probability of having words in common decreases. Furthermore, these measures ignore synonyms and any semantic relatedness between different words, and are prone to negative effects of homonyms.

Instead of relying on exact word overlap, one can incorporate semantic information into the similarity process. Latent Semantic Indexing (LSI) and Latent Dirichlet Allocation (LDA) are two examples, in which every word is projected into a semantic (topic) space (Deerwester et al. (1990); Blei et al. (2003)). At test time, inference is performed to obtain a semantic vector for a particular sentence. Both training and inference of standard LSI and LDA, however, are computationally expensive on large vocabularies.

Although LSI and LDA have been used with success in the past, Skip-gram models have been shown to outperform them in various tasks (Mikolov et al. (2013); Lebret and Collobert (2015)). In Skip-gram, part of Google's word2vec toolkit[1], distributed word embeddings are learned through a neural network architecture to predict its surrounding words in a fixed window.

Once the word embeddings are obtained, we have to combine them into a useful sentence representation. One possibility is to use an multilayer perceptron (MLP) with the whole sentence as an input, or a 1D convolutional neural network (Collobert et al. (2011); Hu et al. (2014); Xu et al. (2015); Johnson and Zhang (2015)). Such an approach, however, requires either an input of fixed length or aggregation operations – such as dynamic k-max pooling (Kalchbrenner et al. (2014)) – to arrive at a sentence representation that has the same dimensionality for every input. Recurrent neural networks (RNNs) and variants can overcome the problem of fixed dimensionality or aggregation, since one can feed word after word in the system and in the end arrive at a text representation (Sutskever et al. (2014); Sordoni et al. (2015); Sundermeyer et al. (2015)). The recently introduced Skip-thought vectors, heavily inspired on Skip-gram, combine the learning of word embeddings with the learning of a useful sentence representation using an RNN encoder and decoder (Kiros et al. (2015)). RNN-based methods present a lot of advantages over MLPs and convolutional networks, but still retraining is required when using different types of embeddings.

Paragraph2vec is another method, inspired by the Skip-gram algorithm, to derive sentence vectors (Le and Mikolov (2014)). The technique requires the user to train vectors for frequently occurring word groups. The method, however, is not usable in a streaming or on-the-fly fashion, since it requires retraining for unseen word groups at test time.

Aggregating word embeddings through a mean, max, min… function is still one of the most easy and widely used techniques to derive sentence embeddings, often in combination with an MLP or convolutional network (Weston et al. (2014); dos Santos and Gatti (2014); Yin and Schütze (2015); Collobert et al. (2011)). On one hand, the word order is lost, which can be important in e.g. paraphrase identification. On the other hand, the methods are simple, out-of-the-box and do not require a fixed length input.

Related to the concepts of semantic similarity and weighted embedding aggregation, there is extensive literature. Kusner et al. (2015) calculate a similarity metric between documents based on the travel distance of word embeddings from one document to another one. We on the other hand will derive vectors for the documents themselves. Kenter and de Rijke (2015) learn semantic features for every sentence in the dataset based on a saliency weighted network for which the BM25 algorithm is used. However, the features are being learned for every sentence prior to test time, and therefore not applicable in a real-time streaming context. Finally, Kang et al. (2014) calculate a cosine similarity matrix between the words of two sentences that are sorted based on their idf value, which they use as a feature vector for an MLP. Their approach is similar to our work in the sense that the authors use idf information to rescale term contribution. Their primary goal, however, is calculating semantic similarity instead of learning a sentence representation. In fact, the authors totally discard the original word embeddings and only use the calculated cosine similarity features.

3. Methodology

The core principle of our methodology is to assign a weight to each word in a short text. These weights are determined based on the idf value of the individual words in that text. The idea is that important words – i.e. words that are needed to determine most of the text's semantics – usually have higher idf values than less important words, such as articles and auxiliaries… Indeed, the latter appear more frequently in various different texts, while words with a high idf value mostly occur in similar contexts. The final goal is to combine the weighted words into a semantically effective, single text representation.

To achieve this goal, we will model the problem of finding a suitable text representation as a semantic similarity task between couples of short texts. In order to classify such couples of text fragments into either semantically related pairs or non-related pairs, the vector representations of these two texts are directly compared. In this paper we use a simple threshold function on the distance between the two text representations, as we

[1] Available at `code.google.com/archive/p/word2vec`

want related pairs to lie close to each other in their representation space, and non-related pairs to lie far apart:

$$g(t_1, t_2) = \begin{cases} \text{pair} & \text{if } d(t_1, t_2) \leq \theta \\ \text{non-pair} & \text{if } d(t_1, t_2) > \theta \end{cases}. \quad (2)$$

In this expression t_1 and t_2 are two short text vector representations of dimensionality ν, $d\colon (x, y) \in \mathbb{R}^{2\nu} \rightarrow \mathbb{R}^+$ is a vector distance function of choice (e.g. cosine distance, euclidean distance…), θ is a threshold, and $g(\cdot)$ is the binary prediction of semantic relatedness.

3.1. Basic architecture

As mentioned before, we will assign a weight to each word in a text according to that word's idf value. To learn these weights, we devise a model that is visualised in Figure 1. In the learning scheme, we use related and non-related couples of text as input data. First, the words in every text are sorted from high to low idf values. Original word order is therefore discarded, as is the case in usual standard aggregation operations. After that, every embedding vector for each of the sorted words is multiplied with a weight that can be learned. Finally, the weighted vectors are averaged to arrive at a single text representation.

In more detail, consider a dataset $\mathcal{D}$ consisting of couples of short texts. An arbitrary couple is denoted by C, and the two texts of C by C^α and C^β. We indicate the vector representation of word j in text C^α by C^α_j. All word vectors have the same dimensionality ν. Each text C^α also has an associated length $n(C^\alpha)$, i.e. the number of words in C^α. For now, in this section, we assume that $n \triangleq n(C^\alpha) = n(C^\beta), \forall C \in \mathcal{D}$, a notion we will relax in Section 3.3. The final goal of our methodology is to arrive at a vector representation for both texts in C, denoted by $t(C^\alpha)$ and $t(C^\beta)$. Denoting the sorted texts as $C^{\alpha'}$ and $C^{\beta'}$, we arrive at a vector representation $t(C^\alpha)$ and $t(C^\beta)$ through the following equation:

$$\forall \ell \in \{\alpha, \beta\} : t(C^\ell) = \frac{1}{n} \sum_{j=1}^{n} w_j \cdot C^{\ell'}_j, \quad (3)$$

in which $w_j, j \in \{1, \ldots, n\}$ are the weights to be learned. As such, we create a weighted sum of the individual embeddings, the weights of which are only related to the rank order according to the idf values of all words in the fragment.

The model we construct through this procedure is related to the siamese neural network with parameter sharing from the early nineties (Bromley et al. (1993)). The learning procedure of such models is as follows. We first calculate the vector representations for both texts in a particular couple through Equation (3), both using the same weights, after which we compare the two vector representations through a loss function $\mathcal{L}\left(t(C^\alpha), t(C^\beta)\right)$ that we wish to minimize. After that, the weights are updated through a gradient descent procedure using a learning rate η:

$$\forall j \in \{1, \ldots, n\} : w_j \leftarrow w_j - \eta \cdot \frac{\partial}{\partial w_j} \mathcal{L}\left(t(C^\alpha), t(C^\beta)\right). \quad (4)$$

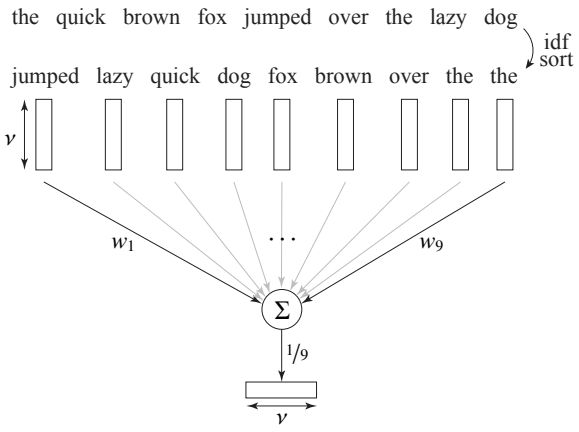

Figure 1. Illustration of the weighted average approach for a toy sentence of nine words long and word vectors of dimension ν.

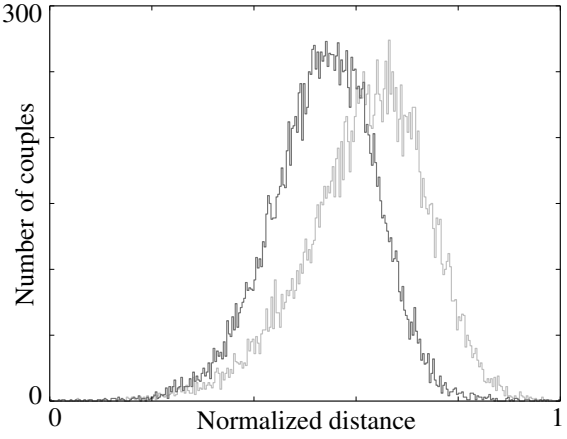

Figure 2. Example distributions of distances between non-related pairs (light grey) and related pairs (dark grey) on Twitter data, using a simple mean of word embeddings.

3.2. Loss functions

As pointed out in the beginning of this section, we want to have semantically related texts to lie close to each other in the representation space, and non-related texts to lie far apart from each other. We can visually inspect the distribution of the distances between every couple in the dataset. In fact, we calculate two distributions, one for the related pairs and one for the non-related pairs. Two examples of such distributions, created using Twitter data and an average word embedding text representation, are shown in Figure 2. Related pairs tend to lie closer to each other than non-related pairs. There is however a considerable overlap between the two distributions. This means that making binary decisions on similarity based on a well-chosen distance threshold will lead to a large error. The goal is to reduce this error, and thus to minimize the overlap between the two distributions. Directly minimizing the overlap is difficult, since it requires the overlap as a function of the model weights. We will instead describe two different loss functions as an approximation to this problem.

The first loss function is related to the contrastive loss function regularly used in siamese neural architectures (Hadsell

et al. (2006)). We define the quantity p_C as follows:

$$p_C \triangleq \begin{cases} 1 & \text{if } C \text{ is a related pair,} \\ -1 & \text{if } C \text{ is a non-related pair.} \end{cases} \quad (5)$$

The loss function, which we will conveniently call the contrastive-based loss, is then given by:

$$\mathcal{L}_{\text{c}}\left(t(C^\alpha), t(C^\beta)\right) = p_C \cdot d\left(t(C^\alpha), t(C^\beta)\right), \quad (6)$$

in which $d\colon (x, y) \in \mathbb{R}^{2\nu} \rightarrow \mathbb{R}^+$ is a vector distance function of choice, as before. Note that, when trying to minimize this loss, related pairs will get pushed to each other in the representation space, while non-related pairs will get dragged apart.

This loss function, however, has two main problems. First, there is an imbalance between the loss for related pairs and non-related pairs, in which the latter can get an arbitrarily negative loss, while the related pairs' loss cannot be pushed below zero. To solve this, we could add a maximum possible distance – which is e.g. 1 in the case of cosine distance – but this cannot be generalized to arbitrary distance functions. Second, this loss function can skew distance distributions, so that minimizing overlap between distributions is not guaranteed. In fact, the overlap can even increase while minimizing this loss. This happens, for example, when the distance between some of the related pairs can be drastically reduced while other related pairs get dragged farther apart, and vice versa for the non-related pairs. The loss function as such allows this to happen, since it can focus on data points that are easier to shift towards or away from each other and it can ignore what happens to the other data points. As long as the contribution of these shifts to the overall loss remains dominant, the loss will diminish, although the predictions according to Equation (2) will be worse. Despite these problems we still consider this loss function due to its simplicity. The derivative with respect to weight w_j is given by:

$$\frac{\partial}{\partial w_j}\mathcal{L}_{\text{c}}\left(t(C^\alpha), t(C^\beta)\right) = \frac{p_C}{n}\left(\nabla d(x, y)\right)\Big|_{x=\frac{1}{n}\sum_{j=1}^{n} w_j C_j^{\alpha\prime}, y=\frac{1}{n}\sum_{j=1}^{n} w_j C_j^{\beta\prime}} \begin{bmatrix} C_j^{\alpha\prime} \\ C_j^{\beta\prime} \end{bmatrix}. \quad (7)$$

In a second loss function we try to mitigate the loss imbalance and potential skewing of the distributions caused by the contrastive-based loss function. For this purpose we will use the median, as it is a very robust statistic insensitive to outliers. As we need multiple data points to calculate the median, this loss function can only be used in the context of minibatch gradient descent, in which the number of positive and negative examples in each minibatch is balanced. In practice we consider a minibatch $B \subset \mathcal{D}$ of $n(B)$ randomly sampled data points, in which there are exactly $n(B)/2$ related pairs and $n(B)/2$ non-related pairs. We consider the couple of texts $M(B) \in B$ as the median couple if $\mu(B) = d(M(B)^\alpha, M(B)^\beta)$ is the median of all distances between the couples in B:

$$M(B) \triangleq \arg\operatorname*{median}_{C \in B} d(C^\alpha, C^\beta); \quad (8)$$

$$\mu(B) \triangleq \operatorname*{median}_{C \in B} d(C^\alpha, C^\beta). \quad (9)$$

Since minibatch B is randomly sampled, we can consider $\mu(B)$ as an approximation to the optimal split point between related and non-related pairs, in the sense of threshold θ in Equation (2). We thus consider all related pairs with a distance larger than $\mu(B)$, and all non-related pairs with a distance smaller than $\mu(B)$ to be classified incorrectly. Since minimizing a 0-1 loss is NP-hard, we use a scaled cross-entropy function in our loss, which we will call the median-based loss:

$$\mathcal{L}_{\text{m}}\left(t(C^\alpha), t(C^\beta), B\right) = \ln\left[1 + \exp\left(-\kappa p_C\big(\mu(B) - d(t(C^\alpha), t(C^\beta))\big)\right)\right], \quad (10)$$

in which κ is a hyperparameter. The derivative with respect to weight w_j is given by the following expression (in which $\sigma(\cdot)$ is the sigmoid function):

$$\frac{\partial}{\partial w_j}\mathcal{L}_{\text{m}}\left(t(C^\alpha), t(C^\beta), B\right) = \kappa\sigma\left(-\kappa p_C\left(\mu(B) - d\left(t(C^\alpha), t(C^\beta)\right)\right)\right) \cdot \frac{\partial}{\partial w_j}\left(\mathcal{L}_{\text{c}}\left(t(C^\alpha), t(C^\beta)\right) - \mathcal{L}_{\text{c}}\left(t(M(B)^\alpha), t(M(B)^\beta)\right)\right). \quad (11)$$

3.3. Texts with variable length

The method described thus far is only applicable to short texts of a fixed length, which is limiting. In this section we will extend our technique to texts of a variable, but given maximum length. For this purpose we have devised multiple approaches, of which we will elaborate the one that performed best in the experiments.

Suppose that all texts have a fixed maximum length of $n_{\max}$. In the learning procedure we will learn a total of $n_{\max}$ weights with the techniques described earlier. To find the weights for a text with length $m \leq n_{\max}$ we will use subsampling and linear interpolation. That is, for an arbitrary text C^ℓ we first find the sequence of real-valued indices $I(C^\ell, j), j \in \left\{1, \ldots, n(C^\ell)\right\}$ through subsampling:

$$\forall j \in \left\{1, \ldots, n(C^\ell)\right\} : I(C^\ell, j) \triangleq 1 + \frac{(j-1)(n_{\max}-1)}{n(C^\ell)-1}. \quad (12)$$

Then, in the second step, we calculate the new weights $z_j(C^\ell), j \in \left\{1, \ldots, n(C^\ell)\right\}$ for the words in C^ℓ through linear interpolation, in which ϵ is arbitrarily small:

$$\forall j \in \left\{1, \ldots, n(C^\ell)\right\} : z_j(C^\ell) \triangleq \frac{(w_{\lceil I_j(C^\ell)\rceil} - w_{\lfloor I_j(C^\ell)\rfloor})(I(C^\ell, j) - \lfloor I_j(C^\ell)\rfloor)}{\lceil I_j(C^\ell)\rceil - \lfloor I_j(C^\ell)\rfloor + \epsilon} + w_{\lceil I_j(C^\ell)\rceil}. \quad (13)$$

In this, $\lceil\cdot\rceil$ and $\lfloor\cdot\rfloor$ are resp. the ceil and floor functions. Finally, Equation (3) needs to be updated with these new weights instead of w_j:

$$\forall \ell \in \{\alpha, \beta\} : t(C^\ell) = \frac{1}{n(C^{\ell\prime})}\sum_{j=1}^{n(C^{\ell\prime})} z_j(C^{\ell\prime}) \cdot C_j^{\ell\prime}. \quad (14)$$

Calculating the derivative of the new weights with respect to the original weights is straightforward.

4. Data collection

To train the weights for the individual embeddings and to conduct experiments, we collect data from different sources. First we gather textual pairs from Wikipedia which we will use as a base dataset to finetune our methodology. We will also use this dataset to test our hypotheses and perform initial experiments. The second dataset will consist of Twitter message pairs, which we will use to show that our method can be used in a practical streaming context.

4.1. Wikipedia

We will perform our initial experiments in a base setting using English Wikipedia articles. The most important benefit of using Wikipedia is that there is a lot of well structured data. It is therefore a good starting point to collect a ground truth to finetune our methodology.

We use the English Wikipedia dump of March 4th 2015, and we remove its markup and punctuation. We convert all letters to lower case and every number is replaced by a single character '0' (zero). Next we construct related pairs of texts which both have the same, fixed length n. To do this, we take a Wikipedia article and we extract n consecutive words out of a paragraph. Then we skip two words, after which we extract the next n consecutive words, as long as they remain in the same paragraph. To extract non-related text pairs we follow the same procedure, but we make sure that the two texts are from different articles, which we choose at random. This approach is closely related to the data collection practice used in (Hu et al. (2014)). We want to emphasize again that our vision of semantic similarity is one of topic-based similarity instead of paraphrase-similarity, as discussed in the introduction. This notion is reflected in our data collection. We extract a total of 4.9 million related pairs and 4.9 million non-related pairs, each for fixed-length texts of 20 words long. We also extract 4.9 million related and non-related pairs of which the texts varies in length between 10 and 30 words. All datasets are divided into a train set of 1.5 million pairs, a test set of 1.5 million pairs and a validation set of 1.9 million pairs.

4.2. Twitter

Twitter is a very different kind of medium than Wikipedia. Wikipedia articles are written in a formal register and mostly consist of linguistically correct sentences. Twitter messages, on the other hand, count no more than 140 characters and are full of spelling errors, abbreviations and slang.

We propose that two tweets are semantically related if they are generated by the same event. As in (De Boom et al. (2015b)), we require that such an event is represented by one or more hashtags. Since Twitter posts and associated events are very noisy by nature, we restrict ourselves to tweets by 100 English news agencies. We manually created this list through inspection of their Twitter accounts; the list is available through our GitHub page, see Section 6.

We gathered 341 949 tweets from all news agencies through the Twitter REST API at the end of August 2015. We convert all words to lowercase, replace all numbers by the single character '0' and remove non-informative hashtags such as *#breaking*, *#update* and *#news*. To generate related pairs out of these tweets, we consider four simple heuristic rules:

1. The number of words in each tweet, different from hashtags, mentions or URLs, should be at least 5.
2. The Jaccard similarity between the set of hashtags in both tweets should be at least 0.5.
3. The tweets should be sent no more than 15 minutes from each other.
4. The Jaccard similarity between the set of words in both tweets should be less than 0.5.

We add the last rule in order to have sufficient word dissimilarity between the pairs, as tweets that mostly contain the same words are too easy to relate. To generate non-related pairs, we remove rule 3 and rule 2 is changed: the Jaccard similarity between sets of hashtags should now be zero. Using these heuristics, we generate a train set of 15 000 pairs, a validation set of 20 000 pairs and a test set of 13 645 pairs, of which we remove all overlapping hashtags. We manually label 200 generated pairs and non-pairs, and we achieve an error rate of 28%. Due to the used heuristics and the linguistic nature of tweets in general, the ground truth can be considered very noisy; achieving an error rate lower than around 28% on this dataset will therefore be difficult, and the gain would not lead to a better model of the human notion of similarity anyway.

5. Experiments

In this section we discuss the results of several experiments on all aspects of our methodology given the data we collected. First we will discuss some results using the Wikipedia dataset, after which we also take a look at the Twitter dataset. We will use two performance metrics in our evaluation. The first is the optimal split error, i.e. we classify all pairs according to Equation (2) – after determining the optimal split point θ – and we determine the classification error rate. A second performance metric is the Jensen-Shannon (JS) divergence. This is a symmetric distance measure between two probability distributions, related to the – well-known, but asymmetric – KL divergence. We will use it to capture the difference between the related and non-related pairs' distributions, as shown in Figure 2. The bigger the JS divergence, the greater the difference between the two distributions.

In our experiments we will use Google's *word2vec* software to calculate word embeddings. We choose Skip-gram with negative sampling as the learning algorithm, using a context window of five words and 400 dimensions. We feed an entire cleaned English Wikipedia dump of March 4th 2015 to the algorithm, after which we arrive at a total vocabulary size of 2.2 million words. Since we also need document frequencies, we calculate these for each of the vocabulary words using the same Wikipedia dump.

In previous work we showed that using an Euclidean distance function leads to a much better separation between related and non-related pairs than the more often used cosine distance, so

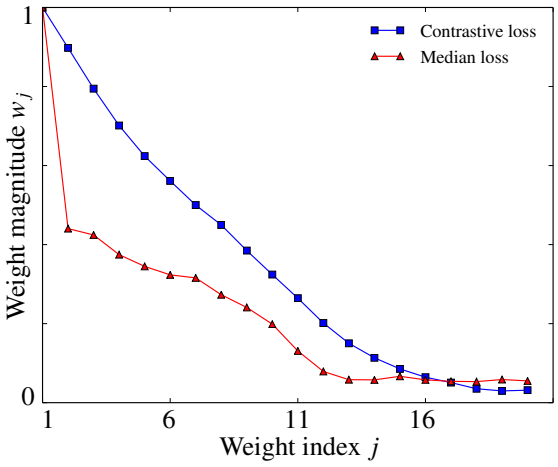

Figure 3. Plot of the learned weight magnitudes for texts of 20 words long.

we will also use the Euclidean distance throughout our experiments here (De Boom et al. (2015a)). Calculating the gradient of this distance function – which is used in Equation (7) – is straightforward.

To obtain the results hereafter, we use the following procedure. We use the train set to train the weights w_j in Equation (3). The validation set is used to determine the optimal split point θ in Equation (2). Finally predictions and evaluations are made on the test set. In the next two subsections we discuss the results on the Wikipedia and Twitter datasets.

5.1. Baselines

We will compare the performance of our methods to several baseline mechanisms that construct sentence vector representations. The simplest and most widely used baseline is a tf-idf vector. Comparing two tf-idf vectors is done through a standard cosine similarity. In a second baseline we simply take, for every dimension, the mean across all embeddings:

$$\forall \ell \in \{\alpha, \beta\} : \forall k \in \{1, \ldots, \nu\} : t(C^\ell)_k = \underset{j}{\text{mean}}\, C^\ell_{j,k}. \qquad (15)$$

In the third baseline we replace the mean by a maximum operation. Taking a mean or a maximum is a very common approach to combine word embeddings – or other intermediary representations in an NLP architecture – into a sentence representation. We can also replace the maximum in Equation (15) by a minimum. The fourth baseline is a concatenation of the maximum and minimum vectors, resulting in a vector having two times the original dimensionality ('min/max'). We can also apply the three previous baselines – i.e. 'mean', 'max' and 'min/max' – only considering words with a high idf value ('top 30% idf'). That is, we sort the words in a text based on their idf values, and we take the mean or maximum of the top 30%. In a final baseline we weigh each word in the sentence with its corresponding idf value and then take the mean ('mean, idf-weighted').

5.2. Details on the learning procedure

In the results below we use the methodology from Section 3 to learn weights for individual words in a short text. All procedures are implemented with the Theano[2] framework and executed using an Nvidia Tesla K40 GPU. We use mini-batch stochastic gradient descent as learning procedure and L_2 regularization on the weights. The total loss for one training batch thus becomes:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{batch}} + \lambda \sum_{j=1}^{n_{\max}} w_j^2. \qquad (16)$$

In this, we empirically set parameter λ to 0.001, and $\mathcal{L}_{\text{batch}}$ is either equal to the contrastive or median-based loss depending on the experiment. The batch size is equal to 100 text couples, of which 50 are related pairs and 50 are non-related pairs. An initial learning rate η of 0.01 is employed, which we immediately lower to 0.001 once the average epoch loss starts to deteriorate. After that, we stop training when the loss difference between epochs is below 0.05%. The weights are initialized uniformly to a value of 0.5. The entire procedure is visualised in Algorithm 1. To determine the optimal value of the hyperparameter κ in the median-based loss, we use five-fold cross-validation and a grid search procedure.

Algorithm 1: Detailed training procedure

$\forall j$: $w_j \leftarrow 0.5$
$\eta \leftarrow 0.01$
$\mathcal{L}_{\text{mean-old}} \leftarrow \infty$
repeat
  $\mathcal{L}_{\text{mean}} = 0$ ▷new epoch
  for *batch* $i \in$ *dataset* **do**
    $\mathcal{L}_{\text{total}} \leftarrow \mathcal{L}_{\text{batch}_i} + \lambda \sum_{j=1}^{n_{\max}} w_j^2$
    $\forall j$: $w_j \leftarrow w_j - \eta \cdot \frac{\partial}{\partial w_j} \mathcal{L}_{\text{total}}$ ▷gradient descent
    $\mathcal{L}_{\text{mean}} \leftarrow \frac{(i-1)\mathcal{L}_{\text{mean}} + \mathcal{L}_{\text{total}}}{i}$ ▷update mean loss
  if $\eta > 0.001 \wedge \mathcal{L}_{mean} > \mathcal{L}_{mean\text{-}old}$ **then**
    $\eta \leftarrow 0.001$
  if $\eta == 0.001 \wedge \mathcal{L}_{mean\text{-}old} - \mathcal{L}_{mean} < 0.0005$ **then**
    STOP
  $\mathcal{L}_{\text{mean-old}} \leftarrow \mathcal{L}_{\text{mean}}$
until *STOP*

5.3. Results on Wikipedia

We train weights for Wikipedia texts with a fixed length of 20 words using the training procedure described in the previous section. The weights already converge before the first training epoch is finished, that is, without having seen all examples in the train set. This is due to the simplicity of our model – i.e. there are only limited degrees of freedom to be learned – and the large train set. Through cross-validation and grid-search we find an optimal value for $\kappa = 160$ used in the median-based loss. The resulting weights for texts of 20 words long are visualized in Figure 3, for both the contrastive- and median-based loss. In both cases, the weights drop in magnitude with increasing

[2] deeplearning.net/software/theano

Table 1. Results for the Wikipedia data, for texts of length 20 and variable lengths, using Wikipedia and Google embeddings.

	Wikipedia, 20 words		Wikipedia, variable length		Google, variable length	
	Split error	JS divergence	Split error	JS divergence	Split error	JS divergence
Tf-idf	19.60%	0.3698	22.23%	0.3190	**22.50%**	**0.3130**
Mean	19.43%	0.3781	25.61%	0.2543	34.41%	0.1130
Max	19.05%	0.3981	27.09%	0.2308	37.05%	0.0842
Min/max	16.78%	0.4520	26.19%	0.2487	35.82%	0.1032
Mean, top 30% idf	17.02%	0.4435	22.67%	0.3182	32.07%	0.1512
Max, top 30% idf	18.05%	0.4193	28.32%	0.2150	36.38%	0.0999
Min/max, top 30% idf	16.40%	0.4581	27.86%	0.2260	35.61%	0.1140
Mean, idf-weighted	24.00%	0.2767	27.51%	0.2139	33.88%	0.1185
Learned weights, contrastive	14.44%	0.5080	21.20%	0.3503	30.50%	0.1776
Learned weights, median	**14.06%**	**0.5184**	**16.41%**	**0.4602**	25.10%	0.2641

index; this confirms the hypothesis that words with a low idf contribute less to the overall meaning of a sentence than high idf words. For the median-based loss, the first word is clearly the most important one, as the second weight is only half the first weight. It is important to point out that we observe a similar monotonically decreasing pattern for texts of 10 words and 30 words long, which means that we use an equal proportion of important to less important words, no matter how long the sentence is. From the 16th word on, the weights are close to zero, so these words almost never contribute to the meaning of a text. In comparison, there are, by experiment, eight non-informative words on average in a text of twenty words long.

The results of the experiments are summarized in Table 1. In a first experiment we compare the performance of our approach to all baselines for texts of length 20 and with word embeddings trained on Wikipedia. Our method significantly outperforms all baselines ($p < 0.001$, two-tailed binomial test) by a high margin for both losses. We also see that a plain tf-idf vector can compete with the simplest and most widely used baselines. We further notice that concatenating the minimum and maximum vectors performs approx 2.25% better than when using a maximum vector alone, which implies that the sign in word embeddings holds semantically relevant information.

In a second experiment we vary the length of the texts between 10 and 30 words. The overall performance drop can be addressed to the presence of text pairs with a length shorter than 20 words. Tf-idf now outperforms all baselines. For texts of 30 words the probability of word overlap is after all much higher than for texts of 10 words long; pairs of long texts thus help lower the error rate for tf-idf. Our method is still the overall best performer ($p < 0.001$, two-tailed binomial test), but this time the median-based loss is able to improve the contrastive-based loss by a high margin of almost 5% in split error, showing that the former loss is much more robust in a variable-length setting.

Finally, we also performed experiments with word embeddings from Google News, see footnote 1. We want to stress that we did not retrain the weights for this experiment in order to test whether our technique is indeed applicable out-of-the-box. There is only 20.6% vocabulary overlap between the Wikipedia and Google word2vec model, and there are only 300 dimensions instead of 400, which together can explain the overall performance drop of the word embedding techniques. It is also possible that the Google embeddings are no right fit to be used on Wikipedia texts. Although our model was not trained to perform well on Google embeddings, we still achieve the best error rate of all embedding baselines ($p < 0.001$, two-tailed binomial test), and again the median loss outperforms the contrastive loss by approx 5%. Tf-idf, on the other hand, is the clear winner here; it did almost not suffer from the vocabulary reduction. It shows that vector dimensionality and context of usage are important factors in choosing or training word embeddings.

5.4. Results on Twitter

Next we perform experiments on the data collected from Twitter. We do not train additional word embeddings for Twitter, but we keep using the Wikipedia embeddings, since we have restricted ourselves to tweets of news publishers, who mainly use clean language. We also keep the same setting for κ as in the Wikipedia experiments.

The results for the Twitter experiments are shown in Table 2. As expected, the error rate is quite high given the noise present in the dataset. We also notice that the split error remains slightly higher than the human error rate of 28%. Tf-idf performs worst in this experiment. Compared to Wikipedia, tf-idf vectors for tweets are much sparser, which leads to a higher error rate. Tf-idf is clearly not fit to represent tweets efficiently. The baselines on the other hand have a much better, but overall comparable performance. Our method with median-based loss performs the best. The approach using contrastive loss performs worst among all embedding baselines, as during training the distribution of distances between related and between non-related texts rapidly gets skewed and develops additional modes. This causes the overall training loss to decrease, while the overlap between the related pairs' and non-related pairs' distribution further increases. The overall improvement of the median-based loss over the idf-weighted baseline is not statistically significant ($p > 0.05$, two-tailed binomial test); so, based on this Twitter dataset alone, we cannot draw any statistically sound conclusion whether our method is better in terms of split error than the idf-weighted baseline. Combined with the results from Table 1, however, we can conclude that choosing median-based learned weights is generally recommended.

Table 2. Results for the Twitter data using Wikipedia embeddings.

	Split error	JS divergence
Tf-idf	43.09%	0.0634
Mean	33.68%	0.0783
Max	34.85%	0.0668
Min/max	33.80%	0.0734
Mean, top 30% idf	32.60%	0.0811
Max, top 30% idf	33.38%	0.0740
Min/max, top 30% idf	32.86%	0.0762
Mean, idf-weighted	31.28%	0.0886
Learned weights, contrastive	35.48%	0.0658
Learned weights, median	**30.88%**	**0.0900**

6. Conclusion

We devised an effective method to derive vector representations for very short fragments of text. For this purpose we learned to weigh word embeddings based on their idf value, using both a contrastive-based loss function and a novel median-based loss function that can effectively mitigate the effect of outliers. Our method is applicable to texts of a fixed length, but can easily be extended to texts of a variable length through subsampling and linear interpolation of the learned weights. Our method can be applied out-of-the-box, that is, there is no need to retrain the model when using different types of word embeddings. We showed that our method outperforms widely-used baselines that naively combine word embeddings into a text representation, using both toy Wikipedia and real-word Twitter data. All code for this paper is readily available on our GitHub page `github.com/cedricdeboom/RepresentationLearning`.

Acknowledgments

This work is soon to be published in Pattern Recognition Letters. Cedric De Boom is funded by a Ph.D. grant of the Flanders Research Foundation (FWO). Steven Van Canneyt is funded by a Ph.D. grant of the Agency for Innovation by Science and Technology in Flanders (IWT). We acknowledge Nvidia for its generous hardware support.

References

Achananuparp, P., et al., 2008. The Evaluation of Sentence Similarity Measures, DaWaK.

Blei, D.M., et al., 2003. Latent dirichlet allocation. JMLR.

Bromley, J., et al., 1993. Signature Verification Using a Siamese Time Delay Neural Network. NIPS.

Collobert, R., et al., 2011. Natural Language Processing (Almost) from Scratch. JMLR.

De Boom, C., et al., 2015a. Learning Semantic Similarity for Very Short Texts, in: ICDMW.

De Boom, C., et al., 2015b. Semantics-driven Event Clustering in Twitter Feeds., #MSM.

Deerwester, S.C., et al., 1990. Indexing by Latent Semantic Analysis. JASIS.

Hadsell, R., et al., 2006. Dimensionality Reduction by Learning an Invariant Mapping., CVPR.

Hu, B., et al., 2014. Convolutional Neural Network Architectures for Matching Natural Language Sentences, NIPS.

Johnson, R., Zhang, T., 2015. Semi-Supervised Learning with Multi-View Embedding: Theory and Application with Convolutional Neural Networks. arXiv.org arXiv:1504.01255v1.

Jonnalagedda, N., Gauch, S., 2013. Personalized News Recommendation Using Twitter, WI-IAT.

Kalchbrenner, N., et al., 2014. A Convolutional Neural Network for Modelling Sentences, ACL.

Kang, L., Hu, B., et al., 2014. A Short Texts Matching Method using Shallow Features and Deep Features, NLPCC.

Kenter, T., de Rijke, M., 2015. Short Text Similarity with Word Embeddings, CIKM.

Kiros, R., et al., 2015. Skip-Thought Vectors. arXiv.org arXiv:1506.06726v1.

Kusner, M.J., et al., 2015. From Word Embeddings To Document Distances. ICML.

Le, Q.V., Mikolov, T., 2014. Distributed Representations of Sentences and Documents. arXiv.org arXiv:1405.4053v2.

Lebret, R., Collobert, R., 2015. N-gram-Based Low-Dimensional Representation for Document Classification. arXiv.org arXiv:1412.6277.

Manning, C., et al., 2009. An Introduction to Information Retrieval. Cambridge University Press.

Mikolov, T., et al., 2013. Efficient Estimation of Word Representations in Vector Space, ICLR.

Mondal, A., Chaturvedi, I., Das D., Bajpai R., Bandyopadhyay, S., 2015. Lexical Resource for Medical Events: A Polarity Based Approach, ICDMW.

Robertson, S.E., Zaragoza, H., 2009. The Probabilistic Relevance Framework: BM25 and Beyond. Foundations and Trends in Information Retrieval.

dos Santos, C.N., Gatti, M., 2014. Deep Convolutional Neural Networks for Sentiment Analysis of Short Texts, COLING.

Sordoni, A., et al., 2015. A Hierarchical Recurrent Encoder-Decoder For Generative Context-Aware Query Suggestion. arXiv.org arXiv:1507.02221v1.

Sundermeyer, M., et al., 2015. From Feedforward to Recurrent LSTM Neural Networks for Language Modeling. IEEE/ACM Transactions on Audio, Speech, and Language Processing.

Sutskever, I., et al., 2014. Sequence to Sequence Learning with Neural Networks, NIPS.

Weston, J., et al., 2014. #TagSpace: Semantic embeddings from hashtags, EMNLP.

Xu, J., et al., 2015. Short Text Clustering via Convolutional Neural Networks, NAACL.

Yin, W., Schütze, H., 2015. Convolutional Neural Network for Paraphrase Identification., NAACL.