Semantic Oppositeness Embedding Using an Autoencoder-Based Learning Model

Nisansa de Silva(✉) and Dejing Dou

Department of Computer and Information Science, University of Oregon, Eugene, USA
{nisansa,dou}@cs.uoregon.edu

Abstract. Semantic oppositeness is the natural counterpart of the much popular natural language processing concept, semantic similarity. Much like how semantic similarity is a measure of the degree to which two concepts are similar, semantic oppositeness yields the degree to which two concepts would oppose each other. This complementary nature has resulted in most applications and studies incorrectly assuming semantic oppositeness to be the inverse of semantic similarity. In other trivializations, "semantic oppositeness" is used interchangeably with "antonymy", which is as inaccurate as replacing semantic similarity with simple synonymy. These erroneous assumptions and over-simplifications exist due, mainly, to either lack of information, or the computational complexity of calculation of semantic oppositeness. The objective of this research is to prove that it is possible to extend the idea of word vector embedding to incorporate semantic oppositeness, so that an effective mapping of semantic oppositeness can be obtained in a given vector space. In the experiments we present in this paper, we show that our proposed method achieves a training accuracy of 97.91% and a test accuracy of 97.82%, proving the applicability of this method even in potentially highly sensitive applications and dispelling doubts of over-fitting. Further, this work also introduces a novel, unanchored vector embedding method and a novel, inductive transfer learning process.

Keywords: Semantic oppositeness · Autoencoder · Transfer learning · Unanchored Learning

1 Introduction

Semantic similarity measures are widely used in Natural Language Processing (NLP) applications [1–3]. The reason for this popularity is that NLP methods built around the simple exact matching approach would yield results with a weaker recall, in comparison with the gold standard. Free text has a tendency to use synonyms and similar text in substitution, which may go unnoticed if an exact match method is used in NLP. Semantic oppositeness is the natural counterpart of the semantic similarity function [4]. While semantic similarity

S. Hartmann et al. (Eds.): DEXA 2019, LNCS 11706, pp. 159–174, 2019.
https://doi.org/10.1007/978-3-030-27615-7_12

yields the degree to which two concepts are similar in a given domain, to be used in the purpose of confidence calculation in applications, semantic oppositeness yields the degree to which two concepts oppose each other in a given domain for the same purpose.

The use of an oppositeness measure in NLP is relevant in the case of contradiction finding, or in the case of extracting negative (negation) rules from a corpus. This, in turn, helps in building reasoning chains and other utilities for various front-end applications, such as question-answering systems and chat bots. It can also be used in the NLP applications which concern fake news or propaganda, given that the candidate text that is being analyzed would contain concepts opposing the generally accepted corpus of knowledge. In fact, one previous work on this domain [4] was an implementation of an oppositeness measure to discover contradictions from medical abstracts in the PubMed archive [5]. Some later considerations of this algorithm were in the legal domain [6] and in text classification [7].

Despite being both innovative and useful, the oppositeness calculation algorithm of the PubMed Study [4] is very computationally intensive. Therefore, in large tasks it would significantly slow down the process. It is in an attempt to avoid such computational complexity that most Natural Language Processing (NLP) tasks involve the incorrect generalization of reducing semantic oppositeness to antonymy [8] or inverse of semantic similarity. In the case of semantic similarity, this problem was overcome with the rise of the word embedding systems. Tasks which used to be a complex set of word similarity calculations [9,10] were reduced to simple K-NN look-ups in vector spaces [11,12]. The objective of this paper is to obtain such an embedding for semantic oppositeness, so that NLP applications that involve semantic oppositeness can become more efficient and cost effective. The proposed method first autoencodes word vectors, then it transfer learns the decode half of the deep neural network by using values obtained by the previous oppositeness algorithm [4] as the target.

In addition to the main research contribution of introducing an embedding for the semantic oppositeness function, in this paper we also introduce a novel, unanchored vector embedding approach and a novel, *inductive transfer learning* [13] process based on autoencoders [14], which utilizes both the learnt embeddings and the learnt latent representation.

2 Related Work

Even though not as extensively as its counterpart, semantic similarity [9,10], there have been a few studies on the derivation and uses of semantic oppositeness [4,8,15–17]. However, almost all of these studies reduce oppositeness from a scale to bipolar scales [16] or anonymity [8]. The study by [4] proves that the reduction of oppositeness to a binary function or defining it as the inverse of semantic similarity function is incorrect. The oppositeness function that we use in this study is heavily influenced by the alternative oppositeness function that is proposed in their study. The said study is an NLP application of the PubMed archive to find contradictions in medical research paper abstracts.

Word embedding has recently risen as an emerging field in the domain of NLP. The leading algorithms for this task are: Word2vec [11,18], GloVe [12], and Latent Dirichlet Allocation (LDA) [19]. In considering the flexibility, ease of customization, and wide usage, in this study we use word2vec as the starting point for our embedding system. Even though this study is focused on embedding oppositeness rather than embedding words, given that oppositeness is an emergent property between pairs of words, the points of embedding, in this study, remain as words. This is the reason it is possible to use word2vec as a reasonable starting point.

Autoencoders are one of the simplest forms of the representation learning algorithms. They consist of two components, an encoder and a decoder. While autoencoders are trained to preserve as much information as possible, special steps are taken to prevent them from learning the identity function [14]. Autoencoders are fairly common in contemporary research [20–22]. A study by [20] proved that stacked autoencoders can out-perform older models.

The proposed unanchored approach to word vector embedding, while being a novel idea introduced in this paper, has some similarity to studies by [23–25]. Transfer learning is a machine learning technique mainly employed when there is a task in a domain of interest with a scarcity of data, while another related domain exists with sufficient training data [13]. Given that the task employed in this work uses transfer learning in such a way that source and target domains are the same, while the source and target tasks are different but related, by the definition given by [13] it is possible to declare that this methodology is based on the principals of *inductive transfer learning*. In the NLP domain, transfer learning is commonly used for the task of document classification [26–28] and sentiment analysis [29]. The novelty in this paper pertaining to transfer learning on autoencoders is how the system is first trained as an autoencoder, and then how the transfer learning transforms it into a neural model, where the encoder toward latent space is kept intact, while the decoder is retrained into a mapper to a different vector space.

3 Methodology

The methodology of this work consists of two components. The first component is calculating the oppositeness value from the algorithm adapted from the work of [4]. For ease of reading, this method shall be referred to as the *original oppositeness measure (OOM)* in the remainder of this paper. The second component is embedding the oppositeness values obtained from the above step.

3.1 Linguistic Measures

The first component, which is needed to calculate the oppositeness between two given words, w_1 and w_2, in the algorithm proposed by the *OOM*, is the weighted semantic similarity. Among the various semantic similarity measures available as the *sim* function, the *OOM* picks the method proposed by [10] which gives

the similarity between two words in the 0 to 1 range. For their reasoning of selecting of this method for semantic similarity over the other methods, they refer to their earlier work [30] in which they comprehensively compared various semantic similarity measures. We, in this paper, decided to follow the same progression and use the Wu and Palmer method as the semantic similarity measure for this work. However, it should be noted here that the *OOM* was intended for finding oppositeness between relationships in triples, whereas the work discussed in this paper is interested in embedding oppositeness between individual words. Thus the length constant values in equation for $simil_{w_1,w_2}$ and all subsequent equations would be trivially collapsed to carry the value 1. Therefore, in this study, $simil_{w_1,w_2}$ simplifies to [10]'s $sim(w_1, w_2)$. However, we discuss the possibility of extending this algorithm to incorporate embedding oppositeness between phrases in Sect. 5 by reintegrating the above constants.

For the *difference* component of the oppositeness calculation, it is needed to calculate the lemma of the given words and then obtain the antonym set of all the possible senses of the given word. The *OOM* does not provide a formal mathematical expression on this step; but we define L_w as $lemma(w)$ and A_w as $antonyms(w)$. The final relative difference equation we use is almost identical to the equation proposed by the *OOM*. However, a few alterations are made to accommodate the cumulative changes we have performed above. The final relative difference equation is shown in Eq. 1 where $P = \{(w_1, w_2), (w_2, w_1)\}$.

$$reldif_{w_1,w_2} = \underset{(i,j)\in P}{avg}\left[\arg\max_{a_k \in A_i}\left(sim(L_j, a_k)\right)\right] \tag{1}$$

Original Oppositeness Model. The *OOM* is built on the principle that the oppositeness value of two words that are highly similar should be more correlated with their difference value than oppositeness value of two words that are less similar. This property is obtained by the Eq. 2. To explain this property, they use an example where they calculate the oppositeness values of *expand*, *decrease*, *change*, and *cat* against the word *increase*. The expectation is that the high difference values of the words *change* and *cat* would be augmented by the similarity value in such a way that the relevant word *change* would be put in the correct place on the oppositeness scale (between *expand* and *decrease*), while the irrelevant word *cat* would be easily distinguishable for removal. We continue to use this same example set of words (or extensions thereof) in our subsequent comparisons, for ease of comparison and preservation of flow. The calculated values for similarity, difference, and finally the oppositeness by Eq. 2 are shown in Table 1. The power scaling constant K is determined by the instructions from *OOM*. Further, we use the same visualization structure as used by the *OOM* in Fig. 1(a). However, we add an overlay of the placement of the four example words (*expand*, *decrease*, *change*, and *cat*) in relation to the word *increase*, for the ease of explanation of the subsequent alterations and additions we perform upon the basic algorithm.

$$oppo\_ori_{w_1,w_2} = reldif_{w_1,w_2}^{\left(K * simil_{w_1,w_2}+1\right)} \tag{2}$$

Table 1. Oppositness with $w_1 = increase$

	expand	*decrease*	*change*	*cat*
$simil_{w_1,w_2}$	0.80	0.75	0.46	0.25
$reldif_{w_1,w_2}$	0.63	1.00	0.72	0.25
$oppo\_ori_{w_1,w_2}$	0.25	1.00	0.49	0.11

The main weakness of this original model is the heavy dependence on the antonym property of the candidate words to calculate the difference. This weakness did not affect the performance of the application discussed by the *OOM*, because they were comparing the oppositeness between the relationship component extracted from triples from medical abstracts. In that application, the relationship component always returns one or more action verbs. Coupled with the fact that they are comparing relationship strings which contain more than one word, most of the instances translate to a high probability of encountering words with antonyms. But in the application of this study, not only should the algorithm handle single word instances, it also has to handle the possibility of that word not having an antonym. In such cases where one or both considered words do not have antonyms, the difference value calculated by Eq. 1 collapses to zero. This in turn further collapses the final oppositeness value calculated by Eq. 2 to zero; thus effectively rendering the particular data point obtained by the word pair in question, unusable.

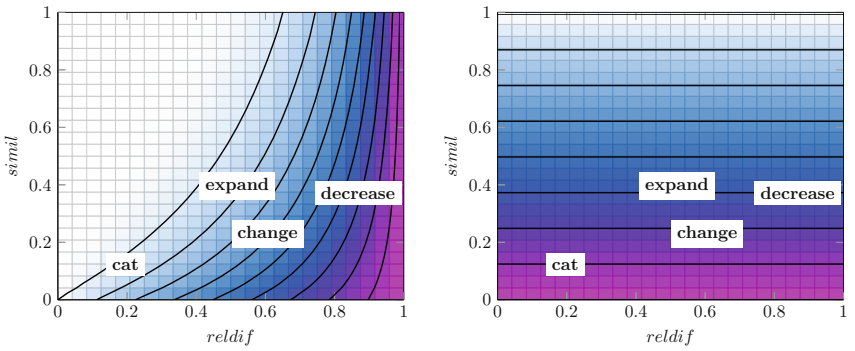
Proposed Balancing Model Derived from the Naïve Oppositeness Measure. In the attempt to solve the problem of incalculable difference values, we turn to the naïve oppositeness measure that was replaced by the oppositeness measure proposed by the *OOM*. This naïve oppositeness measure is the simple operation of declaring the complement of similarity as oppositeness. The Eq. 3 shows the definition of this measure. The Table 2 contains a comparative analysis of the above mentioned oppositeness measure and the naïve method. It is obvious from this data that the naïve method on its own does not achieve the desired properties of an oppositeness scale as stipulated in the *OOM*. While it is obvious that the naïve oppositeness measure is independent of the difference measure, we plot it on the same axis as the above oppositeness measure for the sake of comparison in Fig. 1(b). However, at this point it should be noted that this model does not suffer from the weakness to words without antonyms that impacts the improved model proposed by the *OOM*.

$$oppo\_nai_{w_1,w_2} = (1 - simil_{w_1,w_2}) \tag{3}$$

Combined Oppositeness Model. We have proposed that the solution to the weakness of the *OOM* is to augment it with the naïve model discussed in Sect. 3.1. However, given that the original oppositeness model is far superior to

Table 2. Oppositness with $w_1 = increase$

	expand	*decrease*	*change*	*cat*
$oppo\_ori_{w_1,w_2}$	0.25	1.00	0.49	0.11
$oppo\_nai_{w_1,w_2}$	0.20	0.25	0.54	0.75

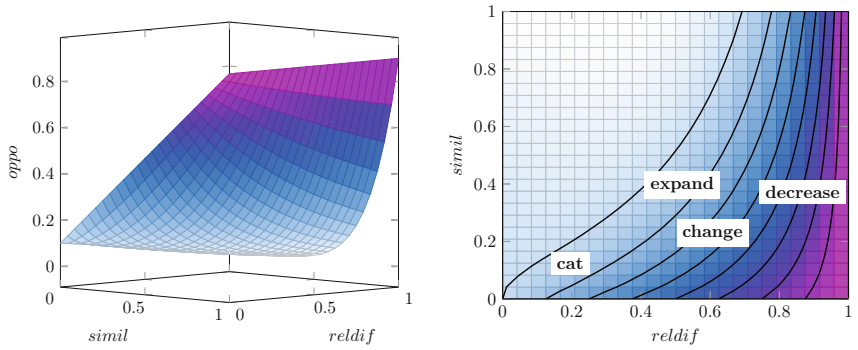


(a) OOM (b) Naive

Fig. 1. Contour plots

the naïve model, and that the original model is weak only at the specific instance where the difference measure is valued zero, it is vital that the two models are combined in a way that the naïve model would only take over at points where the original oppositeness model is weak. We achieved this by multiplying the naive oppositeness function with the term $(1 - reldif)$. Note here that there is no need to further multiply OOM with $reldif$, given that it is already positively correlated with $reldif$. To further fine tune the balance between the OOM and the naïve oppositeness measure, we introduced a hyper parameter α. The final combined oppositeness measure is shown in Eq. 4. Finally, we show the subtle alteration brought about by this improvement in the familiar visualization in Fig. 2. In the example visualization, we have set α to 0.9. While it was needed to set α to this value for the purpose of showing a difference in the graphs discernible to the human eye, in practice, it was observed that α value should be kept at 0.99 or higher, to prevent the naïve oppositeness measure from negatively affecting the overall calculation at points where the difference value is greater than zero. At this point it should be noted that the reason for employing this continuous method to aggregate the two methods, rather than using a case-based approach, where the naïve oppositeness measure is only used at points where the difference measure is zero, is to make-sure that the active surface of the oppositeness curve would be continuous and smooth at all points. Further, note the slight curvature present in Fig. 2(b) in comparison with Fig. 1(a), due to this addition.

$$oppo(w_1, w_2) = \alpha * oppo\_ori + (1 - \alpha)(1 - reldif) * oppo\_nai \tag{4}$$



(a) 3D plot (b) Contour plot

Fig. 2. Oppositeness function with $\alpha = 0.9$

3.2 Embedding Semantic Oppositeness

Once the algorithm described in Sect. 3.1 is used to calculate the oppositeness measures for word pairs, the next step of the process is to embed them in a vector space. Embedding the oppositeness gives applications the ability to do simple K-NN queries on the vector space, instead of running the costly algorithm *OOM* each time. This section discusses the process of embedding the said oppositeness values in a vector space.

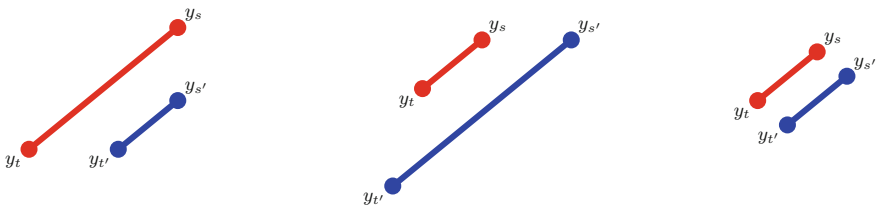
Minimization Constraint. In an embedding process it is important to first define what the minimization constraint is. In almost all cases it is defined as a function to calculate the distance between the vector currently obtained by the embedded object and the vector expected to be obtained by the embedded object. However, in this study our objective is novel in the sense that for this algorithm, it does not matter where the individual word vectors map to. All that matters is the difference between given two embedded word vectors approaching the oppositeness value calculated above. Therefore, in the learning process, instead of anchoring a vector (or a context) and trying to move the target vector close to it, we can employ an algorithm to push both vectors together with no contextual attachments. Thus the minimization constraint becomes a matching of two distance scalars, rather than minimizing the distance between two vectors. The proposed minimization constraint is given in Eq. 5, where $||a - b||$ denotes the Euclidean distance between vectors a and b. Note that we need to preserve the sign of the difference to use the unanchored training. Thus, the absolute value function (abs) is deconstructed into three cases in later steps.

$$min\Big[abs\Big(oppo(w_t, w_s) - ||y_t - y_s||\Big)\Big] \tag{5}$$

Expected Vector Calculation. The range of minimization constraint given in Eq. 5 is unbound. Which means that it can arguably obtain values ranging from $-\infty$ to $+\infty$. In practice, this is bounded by the upper and lower limits of the values obtained by the embedded vectors. However, in either case, this large range is undesirable for the embedding task. Therefore, we define f as shown in Eq. 6 where f would be limited to a range of $+1$ and -1, depending on the placement of the embedded vectors in relation to the expected value, and σ indicates the standard Sigmoid function.

$$f = 2 * \sigma\big[oppo\big(w_t, w_s\big) - ||y_t - y_s||\big] - 1 \tag{6}$$

Target Update Rule. As mentioned in Sect. 3.2, the embedding in this study does not confirm to the idea of anchoring one vector (or context) and pushing the candidate to match the expected vector. Instead, both the vectors in question are moved to make sure the oppositeness is defined by the distance between the said vectors are embedded. It should be noted that to the best of our knowledge, this study is the first to utilize such an unanchored approach to word vector embedding. In this section we derive the update rule for each of the two vectors. For the simplicity of subsequent calculations, we define ΔY as $y_t - y_s$, while the expected shifts are $y_{t'} - y_t$ and $y_{s'} - y_s$. There are three possible cases of vector placement, as shown by Fig. 3. Each of these cases are uniquely identifiable by the f value calculated by Eq. 6. The Table 3 summarizes all update cases.



(a) **Case:** $f < 0$ (b) **Case:** $f > 0$ (c) **Case:** $f = 0$

Fig. 3. Possible cases of vector placement

Table 3. Case-based update rules

	$f < 0$	$f > 0$	$f = 0$
$y_{t'} - y_t$	$-\eta_1 \Delta Y$	$+\eta_2 \Delta Y$	0
$y_{s'} - y_s$	$+\eta_1 \Delta Y$	$-\eta_2 \Delta Y$	0

Finally, it is possible to combine all the above embedding target update rules together, based on the fact that they are uniquely mapped to the value of f, as

shown in Eqs. 7 and 8.

$$y_{t'} = y_t + f\eta\Delta Y \tag{7}$$

$$y_{s'} = y_s - f\eta\Delta Y \tag{8}$$

Autoencoder-Based Learning. While the above algorithm is sound as a solution to embed words in a vector space guided by the oppositeness values, starting with fully empty or fully randomized word vectors would be counterproductive. In such an approach, our system will implicitly have to learn the word embeddings that are achieved by word embedding systems such as word2vec [11] or GloVe [12]. Further, there is the initial hurdle of declaring the input (x_s, x_t) in an unambiguous manner. The solution to both of these problems is to involve an already trained word embedding model as the starting point. In this study we decided to use word2vec. This solves the second problem outright. The declaration of input (x_s, x_t) in an unambiguous manner is now a simple matter of querying the trained Word2vec model with the expectant word string.

The second step is not so straightforward. The objective of this step is to utilize the already existing embedding of words in word2vec to make the oppositeness embedding faster. The rationale here is the fact that word2vec already clusters words by similarity and thus following the naïve method we discussed in above sections, it is reasonable to predict that the oppositeness embedding would be comparatively closer to achieve when starting from a similarity embedding than by a random or a zero embedding. Here, note the fact that the naïve assumption was to assume that the similarity embedding trivially translates to the oppositeness embedding. We do not confirm to that naïve assumption. We only claim that the similarity embedding would be reasonably closer to the expected oppositeness embedding rather than a zero or random starting point. Therefore we propose the novel idea of applying transfer learning [13] on the decoder portion of the autoencoder. The learning model proposed here is shown in Fig. 4.

First of all, to employ the proposed model, we should obtain a mapping from words to vectors. Among the various algorithms and models available to map words to vectors such as Word2Vec [11] and GloVe [12], we propose to use Word2Vec based on the wider support (especially the availability of large Google-trained data set[1]). This component of the ensemble would map a given word to a vector. Incidentally, this would become the input of our neural network.

The proposed model has two learning phases. The first phase of the proposed model is called the *Autoencoding Phase*. In this, we keep the word2vec model locked and the weights of the encoder and the decoder unlocked. The formal representation of an autoencoder is given in Eq. 9 where, the section $\sigma_1(W_1 x_i + b_1)$ correlates to the $encode(X)$ function where the W_1 and b_1 are weights and biases of the *encode* function. The $\sigma_1'(W_1' l) + b_1'$ portion, where l represents $\sigma_1(W_1 x_i + b_1)$ discussed above, correlates to the $decode(l)$ function where the W_1' and b_1' are weights and biases of the *decode* function and l is a

[1] https://goo.gl/yV57W3.

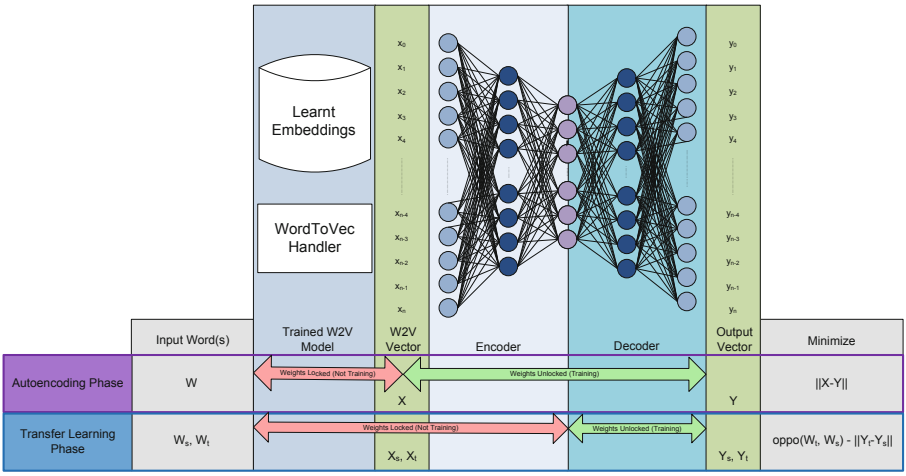


Fig. 4. Overall autoencoder-based learning model

latent representation output by the $encode(X)$ function. The learning objective of the autoencoder is to $minimize(||X - Y||)$ where X is the input vector of the encoder and Y is the output vector of the decoder. Note here that in literature, Y is commonly refereed as X' to showcase the fact that it is supposed to be a reconstruction of the original X. However, in this study we opted to use Y for the sake of clarity of the subsequent steps where we use transfer learning instead of reconstruction (autoencoding).

$$Y_i = (\sigma_1'(W_1'\sigma_1(W_1 X_i + b_1)) + b_1') \tag{9}$$

In summary, during the *Autoencoding Phase* of the proposed model, the neural network learns to reconstruct a given word vector. As mentioned above, this is an attempt to utilize the learnt artifacts of a word embedding system, where related words are clustered together while unrelated words are embedded far apart.

The second phase of the proposed model is the *Transfer Learning Phase*. The transfer learning proposed in this work differs from prior work in literature for three facts. Firstly, the transfer process done on the autoencoder is not done to train yet another model, but to map the same inputs to a different vector space. Thereby, at the end of the training process, the trained neural network is *not* an autoencoder. However, for the sake of readability of the paper, we would continue to refer to the two components of the neural network as *encoder* and *decoder*. It is imperative that after the training, the output of the *decoder* no longer tries to reconstruct the input to the *encoder*. However, given the linguistic properties, that vector will still be *reasonably close* to the input vector. The second difference is the fact that we lock the weights of the *encoder* along with the word2vec model in the training process of this phase. While it is a given property of transfer learning applications to lock a certain number of initial

layers and train only a certain number of layers close to the output layer, usually, that choice is open-ended and unrestricted. In this study however, we specifically lock all the layers that were previously in the *encoder* and keep the layers that were previously in the *decoder* unlocked. The rationale for this decision is as follows: the autoencoder has already learnt a latent representation of the word vectors, by using the autoencoding process, where the output is the input itself. Therefore, we can be sure of the accuracy of the learnt latent representation. The latent representation of a given vector need not be altered when the application is changed. For the best of our knowledge, this study is the first to propose this autoencoder-based *inductive transfer learning* [13] process to utilize both learnt embeddings and the learnt latent representation. The third aspect that distinguishes this model from the traditional learning processes is the fact that this phase uses two forward passes to calculate the error for a single back-propagation pass. This is due to the fact that, as discussed in Sect. 3.2, the learning objective of the neural network is to achieve the minimization proposed in Eq. 5.

4 Experiments and Results

4.1 Calculating the Oppositeness Data

For the purpose of obtaining an adequate collection of words for experimentation, we used the list available in the Linux dictionary[2]. The dictionary contained 72186 total strings. However, it was observed that a certain portion of the strings were non-words. Further, for the sake of preserving the variety of the sample set, it was decided to replace words by their lemmas in cases where there are multiple morphological forms. This process was achieved by passing the potential word strings through the WordNet [31] lemmatizer. This yielded a reduced word list of 65167.

Next the methodology discussed in Sect. 3.1 was used on the 65167 words taken as pairs. Given that each word was considered against all other words, this resulted in 4246737889 pairs of words. For each pair of words, the *similarity*, *difference*, and *oppositeness* were calculated. A few sample lines from the *increase* file are shown in Example 1.1. Note the minimal value of the *similarity* slot for *advents*. This implies that the similarity measure could not give a similarity value to the pair (i.e., the pair is disjoint).

Example 1.1. Sample Oppositeness Lines

```
increase,adrian:0.125,0.1,0.0313516
increase,adriatic:0.125,0.1,0.0313516
increase,advent:0.1875,0.2777778,0.088623986
increase,advents:1.4E-45,1.4E-45,0.01
```

The above files were then processed by applying the irrelevancy threshold proposed by *OOM*. Thus, for each word, all pairs with oppositeness value *less than* the oppositeness value of the most similar pair were eliminated. After this

[2] /usr/share/dict/words.

reduction step, only 76084553 pairs out of the original 4246737889 were left. This means, on average, each word contained 1168 pairs after this step, showing a reduction of 98.21%. As an example, the corresponding file to the word *increase* has only 1107 lines. This is a significant reduction in the case of potential computational load. Given that the file format stays the same as shown in Example 1.1, we do not provide a separate example here.

4.2 Autoencoding on Word2Vec Data

We used a TensorFlow [32,33] based implementation of the two-layer auto-encoder proposed by [34] for the purpose of training on the MNIST data set [35]. The input layer was altered to have the size of 300 to match the trained model of Google's word2vec embedding. The middle layer was of size 256, and the latent layer was of 128. By definition, the decoder layer had mirrored counts. Following the precedence set by [34] for this particular configuration of auto-encoder, we found 30000 epochs to balance accuracy against the threat of over-fitting. By employing the multiple random restart method, we obtained a trained auto-encoder with a validation accuracy of 94.83%. This is the model that we used for the next step of transfer learning.

4.3 Learning of Oppositeness Data

Here it was decided to use a $3:2$ split for the training-validation set vs. test set for the transfer learning of oppositeness data. It is reasonable to assume that the word2vec vector and the expected oppositeness vector will be close in the vector space. Thus, transferring the decoder weights to the system gives a more efficient starting point, compared to initiating with zero weights or random weights.

The training-validation set was divided to 30 equal parts, and each were used to train a clone of the implementation. Here inverse-n-fold cross-validation was used where for each clone, a single portion of data is used as the training data, and then the remaining $n-1$ portions of data are used for validation. We report in Fig. 5 the accuracies of the separate clones resulting from the cross-validation process. The Y axis (rows) of the Matrix corresponds to each transfer learning clone and the x axis (columns) corresponds to the portion of data set. Hence, the 30 entries on the diagonal of the matrix correspond to the training accuracies, and the 870 entries on the remainder of the matrix correspond to the validation accuracies. It is observable that while the diagonal is sightly distinguishable, some clones seem to be performing better than the others across the board. We claim that this is because of the linguistic property that some words are more central in a lexicon than others. These words might distinguish themselves by having more synonyms or by having polysemy. When the data set given to a clone has a majority of such words, it is possible to claim that the trained model would generalize better than in the case where the data set given to a clone has a minority of such words.

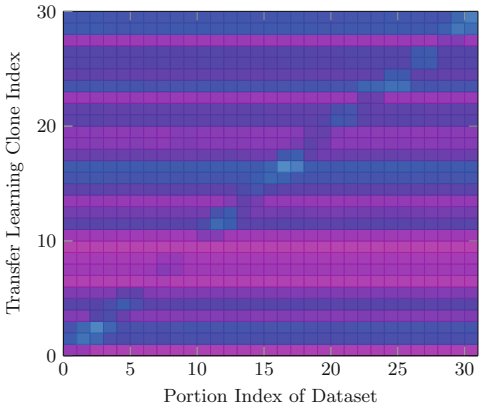


Fig. 5. Training/Validation Matrix of the Clones

Finally, the relevant vectors from the 30 separate embeddings were averaged together to produce the final singular embedding. For that combined singular embedding, on the complete 14820 training-word set (i.e., all 30 portions together), we obtained a mean training accuracy of **97.91%** with a standard deviation of 0.38. The reason for this re-calculation is the fact that we anticipated that since we merged the trained models, the performance of the merged model would not be the average of the separate components. The observable change in accuracy is proof that the said assumption is justified. Also, it is possible to note here that the above rigorous validation has cleared, in the case of the merged model, any possibility of over-fitting which may have threatened individual transfer learning clones. Following that, on the 9910 test words, we obtained a mean test accuracy of **97.82%** with a standard deviation of 0.43. Yet again, we present the closeness of the training accuracy and test accuracy as proof that the system has not over-fitted to the data despite obtaining very good training accuracy which is higher than 97%. Note that all accuracy values at this point are calculated by taking *OOM* output as the gold standard.

The embedding enables practical NLP applications to do a simple k-NN look-up as opposed to the slow and costly oppositeness calculation from scratch. Further, the algorithm by [4] completely fails if even one of the words does not have antonyms. The embedding approach proposed in this study works for all word pairs and gives the closest approximation. Therefore, we claim that this approach provides a more constantly reliable source of oppositeness look-up for practical NLP applications.

5 Conclusion

The main research contribution of this study was the introduction of semantic oppositeness embedding. This study successfully proposed and demonstrated an

embedding methodology on more than 49 million pairs of words, to obtain a training accuracy of 97.91% and a testing accuracy of 97.82%.

In addition to this main research contribution, this study also introduced a novel, unanchored vector-embedding approach and a novel, *inductive transfer learning* process based on auto encoders which utilizes both learnt embeddings and the learnt latent representation.

As for future work, we propose extending this algorithm to embed semantic oppositeness of phrases, similar to the phrase embedding extensions done to the implementations of other word embedding systems, such as Word2Vec.

Acknowledgement. This research is partially supported by the NSF grant CNS-1747798 to the IUCRC Center for Big Learning.

References

1. Gomaa, W.H., Fahmy, A.A.: A survey of text similarity approaches. Int. J. Comput. Appl. **68**(13), 13–18 (2013)
2. Stavrianou, A., Andritsos, P., Nicoloyannis, N.: Overview and semantic issues of text mining. ACM Sigmod Rec. **36**(3), 23–34 (2007)
3. Turney, P.D.: Mining the web for synonyms: PMI-IR versus LSA on TOEFL. In: De Raedt, L., Flach, P. (eds.) ECML 2001. LNCS (LNAI), vol. 2167, pp. 491–502. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44795-4_42
4. de Silva, N., Dou, D., Huang, J.: Discovering inconsistencies in PubMed abstracts through ontology-based information extraction. In: Proceedings of the 8th ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics, pp. 362–371. ACM (2017)
5. National Center for Biotechnology Information: PubMed Help, March 2017
6. Ratnayaka, G., Rupasinghe, T., de Silva, N., Gamage, V.S., Warushavithana, M., Perera, A.S.: Shift-of-perspective identification within legal cases. In: Proceedings of the 3rd Workshop on Automated Detection, Extraction and Analysis of Semantic Information in Legal Texts (2019)
7. de Silva, N.: Sinhala Text Classification: Observations from the Perspective of a Resource Poor Language (2019)
8. Paradis, M., Goldblum, M.C., Abidi, R.: Alternate antagonism with paradoxical translation behavior in two bilingual aphasic patients. Brain Lang. **15**(1), 55–69 (1982)
9. Jiang, J.J., Conrath, D.W.: Semantic similarity based on corpus statistics and lexical taxonomy. In: 10th International Conference on Research in Computational Linguistics, ROCLING 1997 (1997)
10. Wu, Z., Palmer, M.: Verbs semantics and lexical selection. In: Proceedings of the 32nd Annual Meeting on Association for Computational Linguistics. ACL 1994, pp. 133–138. Association for Computational Linguistics, Stroudsburg (1994)
11. Mikolov, T., Sutskever, I., et al.: Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781 (2013)
12. Pennington, J., Socher, R., Manning, C.D.: Glove: global vectors for word representation. In: EMNLP, vol. 14, pp. 1532–1543 (2014)
13. Pan, S.J., Yang, Q.: A survey on transfer learning. IEEE Trans. Knowl. Data Eng. **22**(10), 1345–1359 (2010)

14. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016). http://www.deeplearningbook.org
15. Mettinger, A.: Aspects of Semantic Opposition in English. Oxford University Press, New York (1994)
16. Schimmack, U.: Pleasure, displeasure, and mixed feelings: are semantic opposites mutually exclusive? Cogn. Emotion **15**(1), 81–97 (2001)
17. Rothman, L., Parker, M.: Just-about-right (jar) Scales. ASTM International, West Conshohocken (2009)
18. Mikolov, T., Sutskever, I., et al.: Distributed representations of words and phrases and their compositionality. In: Advances in Neural Information Processing Systems, pp. 3111–3119 (2013)
19. Das, R., Zaheer, M., Dyer, C.: Gaussian LDA for topic models with word embeddings. In: ACL, vol. 1, pp. 795–804 (2015)
20. Lv, Y., Duan, Y., et al.: Traffic flow prediction with big data: a deep learning approach. IEEE Trans. Intell. Transp. Syst. **16**(2), 865–873 (2015)
21. Alsheikh, M.A., Niyato, D., et al.: Mobile big data analytics using deep learning and apache spark. IEEE Network **30**(3), 22–29 (2016)
22. Hinton, G.E., Salakhutdinov, R.R.: Reducing the dimensionality of data with neural networks. Science **313**(5786), 504–507 (2006)
23. Hinton, G.E., Roweis, S.T.: Stochastic neighbor embedding. In: Advances in Neural Information Processing Systems, pp. 857–864 (2003)
24. Ono, M., Miwa, M., Sasaki, Y.: Word embedding-based antonym detection using thesauri and distributional information. In: Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pp. 984–989 (2015)
25. Chen, Z., Lin, W., et al.: Revisiting word embedding for contrasting meaning. In: Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics, vol. 1, pp. 106–115 (2015)
26. Fung, G.P.C., Yu, J.X., et al.: Text classification without negative examples revisit. IEEE Trans. Knowl. Data Eng. **18**(1), 6–20 (2006)
27. Al-Mubaid, H., Umair, S.A.: A new text categorization technique using distributional clustering and learning logic. IEEE Trans. Knowl. Data Eng. **18**(9), 1156–1165 (2006)
28. Sarinnapakorn, K., Kubat, M.: Combining subclassifiers in text categorization: a DST-based solution and a case study. IEEE Trans. Knowl. Data Eng. **19**(12), 1638–1651 (2007)
29. Blitzer, J., Dredze, M., Pereira, F.: Biographies, bollywood, boom-boxes and blenders: domain adaptation for sentiment classification. In: Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics, pp. 440–447 (2007)
30. de Silva, N.H.N.D.: SAFS3 algorithm: frequency statistic and semantic similarity based semantic classification use case. In: 2015 Fifteenth International Conference on Proceedings of Advances in ICT for Emerging Regions (ICTer), pp. 77–83. IEEE (2015)
31. Miller, G.A., Beckwith, R., et al.: Introduction to wordnet: an on-line lexical database. Int. J. Lexicography **3**(4), 235–244 (1990)
32. Abadi, M., Barham, P., et al.: Tensorflow: a system for large-scale machine learning. In: OSDI, vol. 16, pp. 265–283 (2016)
33. Abadi, M., Agarwal, A., et al.: TensorFlow: Large-scale machine learning on heterogeneous systems (2015). tensorflow.org

34. Damien, A.: Auto-Encoder Example. https://goo.gl/wiBspX (2017). Accessed 06 June 2018
35. LeCun, Y., Bottou, L., et al.: Gradient-based learning applied to document recognition. Proc. IEEE **86**(11), 2278–2324 (1998)