HG2Vec: Improved Word Embeddings from Dictionary and Thesaurus Based Heterogeneous Graph

Qitong Wang and **Mohammed J. Zaki**
Computer Science Department
Rensselaer Polytechnic Institute
wangq19@rpi.edu and zaki@cs.rpi.edu

Abstract

Learning word embeddings is an essential topic in natural language processing. Most existing works use a vast corpus as a primary source while training, but this requires massive time and space for data pre-processing and model training. We propose a new model, HG2Vec, that learns word embeddings utilizing only dictionaries and thesauri. Our model reaches the state-of-art on multiple word similarity and relatedness benchmarks. We demonstrate that dictionaries and thesauri are effective resources to learn word embeddings. In addition, we exploit a new context-focused loss that models transitive relationships between word pairs and balances the performance between similarity and relatedness benchmarks, yielding superior results.

1 Introduction

Word embeddings are highly effective for various applications, ranging from recommender systems to named entity recognition (Kubal and Nimkar, 2019). They aim to map words into vectors in a high dimensional space, such that a higher similarity between word embeddings captures a closer semantic relationship, whereas a low or negative similarity indicates unrelated or opposite meaning. As a result, predicting whether the embeddings of two words are similar or not is a common approach to evaluate the quality of the embeddings.

There are two main perspectives to describe the relationship between a pair of words according to Hill et al. (2015): similarity and relatedness. Similarity means that the two words can substitute for each other without generating grammatical mistakes, while relatedness means that the two words always appear together in a context. For example, *coffee* and *tea* is a word pair with high similarity and low relatedness. Being popular breakfast drinks, they share many similar properties (both physically and grammatically), but they are not related to each other. On the other hand, *green* and *tea* is a word pair with high relatedness and low similarity since green tea is a common type of tea. They do not share similar proprieties, but it is common for them to appear together in a sentence.

Most existing works utilize a large corpus to train word embeddings. However, as pointed out by Kiela et al. (2015), models based on association data, like Wikipedia corpus, have better performance on relatedness tasks, whereas models based on thesauri have better performance on similarity tasks. Therefore, in this paper, we propose a new model, HG2Vec, that only relies on dictionaries and thesauri to construct contexts via a heterogeneous graph. We demonstrate that this method can improve the performance of relatedness tasks while maintaining high accuracy on similarity tasks. Our main contributions are as follows:

- We utilize dictionaries and thesauri as the only resources (without using any text corpus) to train word embeddings, and obtain state-of-art results on several benchmarks.
- We demonstrate that learning synonyms and antonyms is necessary for word embedding models, so a model can distinguish the words with opposite meanings that appear in similar contexts.
- We propose a context-focused loss to boost learning in closely related contexts. It is based on modeling transitive synonym and antonym relationships from the thesauri as well as word co-occurrences from dictionary definitions, which balances the performance between similarity and relatedness tasks.

2 Related Work

Many models require a large corpus to learn word embeddings; Word2Vec (Mikolov et al., 2013) is one of them. It iterates through the text with each target word generating a surrounding context window and corresponding negative samples. The goal is to learn embeddings that maximize the similarity between the target and context words while minimizing the similarity between the target and negative sampled words. BERT (Devlin et al., 2018) is an effective contextual embedding model. It tokenizes the input sentences, iterates through the corpus text by blocks, and learns token embeddings by utilizing the attention-based transformer model. BERT uses masked-language modeling and next sentence prediction as the self-supervised tasks during pre-training.

Several approaches leverage dictionaries and other external sources to improve the performance of Word2Vec and BERT. Faruqui et al. (2014) pretrain with other models and then use a dictionary-based relational graph to retrofit semantic information. The paper minimizes the difference between a word embedding from pre-trained models and its corresponding embeddings from the relational graph, and maximizes the similarity between a relational graph word embedding with its neighbors. dLCE (Nguyen et al., 2016) extends the skip-gram model with negative sampling introduced by (Levy and Goldberg, 2014) with synonyms and antonyms. Dict2Vec (Tissier et al., 2017) introduces dictionaries as a resource to train word embeddings. Word pairs are grouped into two types: strong pairs and weak pairs. For each target word, Dict2Vec maximizes the similarity between a context word and its strong and weak pairs for each target word. DRG2Vec (Shu et al., 2020) creates a graph based on the TF-IDF relationship between word pairs in dictionaries. It applies both depth-first and breadth-first based sampling to generate context paths from the graph. It trains the model on both the Wikipedia corpus and the generated context. Dict-BERT (Yu et al., 2021) appends the definition of rare words at the end of the input corpus. Besides mask language modeling, it maximizes the mutual information between the context and the definitions. It also samples wrong definitions during training to check whether the model can distinguish them.

Some models do not rely on a text corpus. CPAE (Bosc and Vincent, 2018) is an auto-encoder model that only relies on dictionaries. It uses an LSTM to reproduce the target word after processing its definitions. Ruzzetti et al. (2021) propose DefiNNet and DefBert to utilize dictionaries to predict the meanings for out-of-vocabulary words. Jana et al. (2022) create a Distributional Thesaurus Network to gather the information from thesauri, showing that utilizing thesauri can improve the performance of Word2Vec. In addition, Zhang et al. (2019) leverage dictionaries as a tool to visualize how other word embeddings contribute to the target word embedding, which is helpful in analyzing the model and improving the performance.

Our model is an enhancement over Dict2Vec and DRG2Vec, where we create a heterogeneous graph from dictionaries and thesauri, and generate documents from the sampled paths. Then, we train the documents to maximize the similarity with related pairs, explicitly considering synonyms and antonyms (along with their transitive relationships), as well as strong and weak pairs.

3 HG2Vec Methodology

3.1 Graph Construction

We first parse the word pairs from the dictionaries and thesauri, and construct a heterogeneous graph for sampling paths. Fig. 1 shows an example of an undirected heterogeneous graph. Each node refers to a word, and each edge refers to a relationship between two words. There are four different types of edges: strong edge (purple), weak edge (blue), synonym edge (black), and antonym edge (red). The edges also have weights, so the graph is weighted, and heterogeneous in terms of edges.

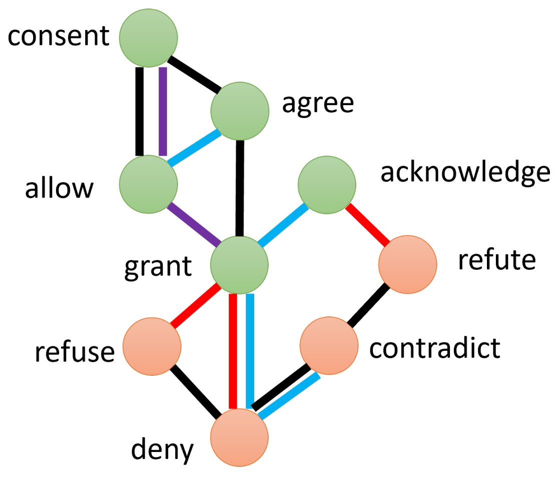

Figure 1: A sample undirected heterogeneous graph in HG2Vec. Each node refers to a word, and each edge refers to a relationship between words. There are four types of edges: strong (purple), weak (blue), synonym (black), and antonym (red). There may be more than one type of edge between two nodes, but all nodes are of the same type (for illustration only, we use green to mark the synonyms of *grant*, and red to mark its antonyms).

Strong and Weak Edges Following the approach in Dict2Vec (Tissier et al., 2017), for a word pair (w_a, w_b), if w_a appears in the definition of w_b in a dictionary, then there is a weak edge between them. If w_b also appears in w_a's definition, we instead add a strong edge between the two. In Fig. 1, since *agree* appears in the definition of *allow*, there is a weak edge between the two. Since *consent* and *allow* both appear in each other's definition, there is a strong edge between them. To model the relative importance of these two relationships, the weight of strong vs. weak edges is $2:1$ in our undirected heterogeneous graph.

Synonym and Antonym Edges We directly extract the synonym and antonym pairs from thesauri. For a word pair (w_a, w_b), if w_a appears as a synonym of w_b, then there is a synonym edge between w_a and w_b. If w_a appears as an antonym of w_b, then there is an antonym edge between them. For example, in Fig. 1, *refuse* is a synonym of *deny* and an antonym of *grant*. The weights of synonym and antonym edges are $1:-1$ in the graph.

Heterogeneous Graph From Fig. 1, we can see that there may be more than one type of edge between two nodes. In most cases, strong edges and weak edges indicate a relationship between words with similar meanings. However, some corner cases refer to the opposite meaning. For example, there is an antonym edge and a weak edge between *grant* and *deny*. As a result, although strong edges and weak edges can depict the relatedness of different words, they may also contain negative words in terms of semantics. If we only use strong edges and weak edges, the model may obscure the opposite meanings. Instead, we leverage thesauri as an external source to automatically consider these related but opposite pairs.

3.2 Path Generation

Our heterogeneous graph has edges with both positive and negative weights, so we cannot directly sample paths from the graph. We first illustrate the sampling for positive edges (strong and weak pairs, and synonyms) and then discuss how to handle negative weights (antonyms).

As such, dictionaries generate strong and weak edges, and thesauri generate synonym and antonym edges. We first generate the paths comprising strong and weak edges and then generate the paths comprising synonym and antonym edges.

Random Walks We generate the paths via random walks [1] that combine both Depth-first Sampling (DFS) and Breadth-frist Sampling (BFS) as first introduced in Node2Vec (Grover and Leskovec, 2016). We denote by n_x the x-th node in a walk, and we start generating the walk with node n_0. For a node $n_x = v_i$, the next node is denoted $n_{x+1} = v_j$, and the previous node is denoted $n_{x-1} = v_h$. Both v_h and v_j are neighbors of v_i. After we randomly choose one of v_i's neighbor as n_{x+1}, we choose one of v_j's neighbor as n_{x+2}. We keep iterating until the path reaches a desired length L (we use $L = 20$ in our model). The probability that we choose a node v_j from the neighbors $N(v_i)$ of v_i is defined as follows:

$$p(n_{x+1} = v_j | n_x = v_i) = \frac{\pi_{ij}}{\sum_{a \in N(v_i)} \pi_{ia}} \tag{1}$$

where

$$\pi_{ij} = \alpha(h, j) \cdot |w_{ij}| \tag{2}$$

In Eq. (2), w_{ij} is the edge weight from node v_i to node v_j. Since we have edges with negative weights, we use the absolute value of edge weight. Therefore, an antonym edge is equivalent to a synonym edge when calculating the probability. Note also that the probability of jumping to v_j also depends on the previous node v_h, which can be at a distance of $d = 0, 1,$ or 2 hops from v_j in the walk. As detailed below in Eq. (3), $\alpha(h, j)$ is a coefficient that controls the tendency of sampling between DFS and BFS; if $\alpha(h, j) = 1$, the algorithm is exactly the same as a random walk.

Node Sampling The distance from a node to the origin is increasing along the walk for DFS, which tends to explore the nodes away from the source. For example, in Fig. 1, the path may be $consent, agree, grant, deny$ if we utilize DFS. In contrast, BFS samples all the neighbors from the source first. It tends to explore the nodes around the origin. In Fig. 1, the path may be $consent, agree, allow, grant$ if we utilize BFS.

As above, let $n_{x-1} = v_h$, $n_x = v_i$, and $n_{x+1} = v_j$ for a sampled walk v_h, v_i, v_j. Here, $v_h = n_{x-1}$ is the node visited in the previous turn, and $v_j = n_{x+1}$ is the node sampled in the next turn. In Eq. (2), $\alpha(h, j)$ is a coefficient to balance

[1] We allow paths to have repeated nodes.

between DFS and BFS, defined as:

$$\alpha(h,j) = \begin{cases} p^{-1}, & \text{if } d(h,j) = 0 \\ 1, & \text{if } d(h,j) = 1 \\ q^{-1}, & \text{if } d(h,j) = 2 \end{cases} \quad (3)$$

In Eq. (3), $d(h,j)$ is the distance between v_h and v_j. If $d(h,j) = 0$, it means the next node is the same as the previous node, so it reflects the tendency to return to the source. When $d(h,j) = 1$, v_j and v_i are both neighbors of v_h. It reflects the tendency to explore around the node n_{x-1}. Finally, when $d(h,j) = 2$, the parameter reflects the tendency to sample new nodes away from the origin. Here, p is called *return parameter*, and a smaller p increases the tendency to return to the origin, whereas q is called *in-out parameter*, and a smaller q leads to a tendency to explore new nodes. For example, in Fig. 1, if v_h is *grant* and v_i is *agree*, a small p leads to choosing *grant* as v_j, a small q leads to choosing *consent*, and the walk samples *allow* if both p and q are large.

Generating Paths We separately sample the walks comprising of strong and weak edges, and those consisting of synonyms and antonym edges. The former directly uses the random walk approach outlined above. Fig. 2 shows a sample path consisting of strong and weak edges sampled from the heterogeneous graph in Fig. 1.

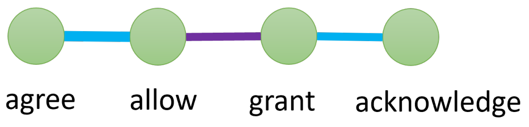

Figure 2: A path with strong (purple) and weak (blue) edges.

The sampling for synonym and antonym edges is more involved. We assume that all the nodes in one path have similar semantic meanings, so we cannot append an antonym to a path with synonyms since their meanings are opposite. However, since an antonym's antonym is essentially a synonym, we can directly attach that to an existing path. We then generate two paths: one for synonyms and another for antonyms. The first path stores the origin and nodes with a similar semantic meaning to the origin. The second path stores the nodes with opposite semantic meanings to the origin. We start at the source and generate the synonyms path via application of Eq. (1). However, when we encounter an antonym (of the source), we put this node as the first node in the antonyms path. We continue appending nodes to the antonyms path for all additional synonym edges encountered. If we visit another antonym, it means that the new node has a similar semantic meaning to the source, so we append the new node to the synonyms path. We continue to append to the synonyms path if we encounter additional synonym edges, and so on. Fig. 3 illustrates this process.

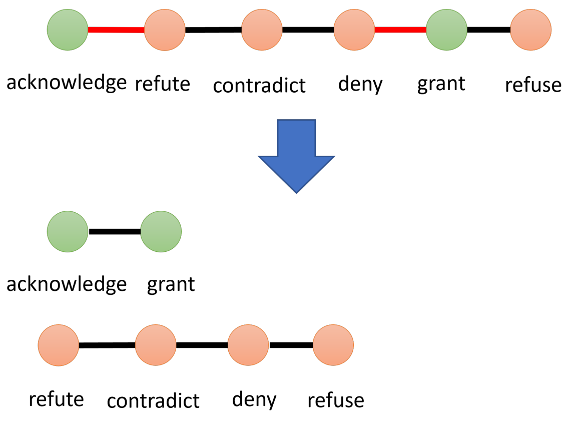

Figure 3: Random walks with synonym (black) and antonym (red) edges. The path on the top is if we were to sample both edges together. Instead, we divide it into two paths: the synonyms path stores the source node and other nodes with similar meanings, whereas the antonyms path stores the nodes with opposite meanings to the source. As a result, the nodes have a similar semantic meaning within each path. Starting with *acknowledge*, when we encounter *refute*, it goes to the antonyms path along with its synonyms *contradict* and *deny*. When we encounter *grant*, which is an antonym of deny, we append it to *acknowledge*. Finally, *refuse* again goes to the antonyms path.

3.3 Sampling Set for Training

We treat each sampled path above as a text document in our model. Next, we iterate through each text document with a context window. For a target word t at position x in a document, using window size ϕ, the context comprises the words $[x - \phi/2 : \; x + \phi/2]$. For example, consider the path *refute, contradict, deny, refuse* in Fig. 3. With window size $\phi = 3$, for the target word *contradict*, its context, denoted $\mathcal{C}(t)$, is *(refute, contradict, deny)*. The window selects the context nodes surrounding the target node, indicating a strong positive correlation with the target.

For each context word $c \in \mathcal{C}(t)$, we sample five negative, strong, weak, synonym, and antonym word pairs for c, denoted as $\mathcal{N}(c)$, $\mathcal{S}(c)$, $\mathcal{W}(c)$,

$\mathcal{Y}(c)$, and $\mathcal{A}(c)$, respectively. The negative words $\mathcal{N}(c)$ are the words sampled from the word dictionary that do not appear in the context $\mathcal{C}(t)$. The other word pairs are sampled from the dictionaries and thesauri, from the set of c's strong and weak pairs, or from c's synonyms and antonyms.

3.4 Context-focused Model

Like Word2Vec, we have two embeddings for each word w: input embeddings $\mathbb{I}(w)$, and output embeddings $\mathbb{O}(w)$. After training, we use the output embeddings for testing and evaluation.

To maximize the similarity of two embeddings, a and b, we calculate their dot product $p = a \cdot b$. A large positive value suggests that the two objects are similar, a large negative value suggests that they are negatively related, and a value around 0 suggests that the two are unrelated. We use $-\log\sigma(p)$ as the loss function where $\sigma(p) = 1/(1+\exp(-p))$ denotes the Sigmoid function.

Context-focused Transitive Loss HG2Vec considers six different contextual features to learn effective embeddings, namely, the context, negative sampled words, strong and weak pairs, and synonyms and antonyms. Each of these contributes to the overall loss function.

For a given target word t, we first maximize the similarity between t and its surrounding context words c. We use the output embeddings for the target words and the input embeddings for the context words. Therefore, the loss is given as the sum over each word pair (t, c) where $c \in \mathcal{C}(t)$:

$$\mathrm{I}_t = \sum_{c\in\mathcal{C}(t)} -\log\sigma(\mathbb{O}(t)\cdot\mathbb{I}(c))$$

To calculate the terms in overall loss, for each word $c \in \mathcal{C}(t)$, we sample a set of strong pairs $\mathcal{S}(c)$, weak pairs $\mathcal{W}(c)$, synonyms $\mathcal{Y}(c)$, antonyms $\mathcal{A}(c)$, and negative words $\mathcal{N}(c)$. The aim is to maximize the similarity of c with its strong, weak, and synonym pairs; and to minimize the similarity with its antonym and negative pairs. Therefore, we divide these sets into two groups, the positive group $\mathcal{P}^+(c) = \{\mathcal{S}(c), \mathcal{W}(c), \mathcal{Y}(c)\}$, and the negative group $\mathcal{P}^-(c) = \{\mathcal{A}(c), \mathcal{N}(c)\}$.

The loss functions within each group are similar; we use synonym pairs as an example. We want to maximize the similarity of c to any of its synonyms in $\mathcal{Y}(c)$; at the same time, the synonyms of other words in $\mathcal{C}(t)$ may also be synonyms of c. For example, in Fig. 1, for the context *(consent, allow, grant)*, the synonyms of *grant* (e.g., *acknowledge*) are also the synonyms of *consent*. However, there is no such synonym path from *acknowledge* to *consent* in the graph since thesauri do not include all the possible transitive relations. However, HG2Vec models these transitive synonym relationships by sampling five synonyms for each context word (e.g., if the context window is 5, then we will consider 25 synonyms in a context). We maximize the similarity between the word c and all the sampled synonyms from the context. The synonym loss for c is given as:

$$\mathrm{L^+}_{\mathcal{Y}(c)} = \beta_{\mathcal{Y}} \sum_{x\in\mathcal{C}(t)} \sum_{p\in\mathcal{Y}(x)} -\log\sigma(\mathbb{O}(c)\cdot\mathbb{I}(p))$$

Here, $\beta_{\mathcal{Y}}$ is a hyperparameter, denoting the importance of synonyms to the overall loss.

For the negative group $\mathcal{P}^-$, we want to minimize the similarity. Considering all (transitive) antonyms of c in a context, the loss is given as:

$$\mathrm{L^+}_{\mathcal{A}(c)} = \beta_{\mathcal{A}} \sum_{x\in\mathcal{C}(t)} \sum_{p\in\mathcal{A}(x)} -\log\sigma(-\mathbb{O}(c)\cdot\mathbb{I}(p))$$

Here, $\beta_{\mathcal{A}}$ specifies the weight for antonyms. As such $\beta_{P^+} = \{\beta_{\mathcal{S}}, \beta_{\mathcal{W}}, \beta_{\mathcal{Y}}\}$, and $\beta_{P^-} = \{\beta_{\mathcal{A}}, \beta_{\mathcal{N}}\}$ specify the weights for each set in the positive and negative groups.

The loss for one target word t in a path (or document) is given as:

$$\begin{aligned}\mathrm{L}_t = \mathrm{I}_t &+ \sum_{c\in\mathcal{C}(t)} \sum_{\mathcal{X}\in\mathcal{P}^+(c)} \mathrm{L^+}_{\mathcal{X}(c)} \\ &+ \sum_{c\in\mathcal{C}(t)} \sum_{\mathcal{X}\in\mathcal{P}^-(c)} \mathrm{L^-}_{\mathcal{X}(c)}\end{aligned} \tag{4}$$

Finally, we sum over all the targets in a path and sum up all the paths to get the total loss.

4 Experiments

For training, all experiments were conducted on a machine with a dual 20 core 2.5 GHz Intel Xeon Gold 6248 CPU, and Nvidia Tesla V100 GPU with 32GB memory. Our HG2Vec implementation is available as open source at `https://github.com/Qitong-Wang/HG2Vec`.

4.1 Datasets

Dictionaries and Thesauri For the strong pairs and weak pairs, we use the same data sources as Dict2Vec (Tissier et al., 2017) and DRG2Vec (Shu et al., 2020). After removing stop words, they extract the strong and weak pairs from the Cambridge, Oxford, and Collins dictionaries, and dictionary.com. We extract the synonym and antonym pairs from Roget's Super Thesaurus 4th Edition (McCutcheon, 2010). In total, our heterogeneous graph contains 211,675 unique nodes/words, with 4,273,743 strong and weak edges, and 119,512 synonym and antonym edges.

Wikipedia HG2Vec does not rely on text corpus data, so we do not need to use Wikipedia corpus during training. However, for the baseline models, we pre-process the Wikipedia corpus from Nov. 2021 (`https://dumps.wikimedia.org/`), using the first 50 million words from the cleaned data.

4.2 Experimental Settings

Hyperparameters The hyperparameters for Dict2Vec and DRG2Vec are the same as those reported in their papers. For HG2Vec, for generating the paths, we use $\phi = 5$, $p = 1.5$, $q = 5.0$ and $L = 20$. Word embedding size is 300. We use five strong, weak, synonym, antonym, and negative samples for the loss computation. We train the model for five epochs; in each epoch, we sample one strong-weak path and one synonym-antonym path per node in the heterogeneous graph. To tune the β hyperparameters, we use a grid search from 0.1 to 5. We find that $\beta_{\mathcal{S}} = 0.4, \beta_{\mathcal{W}} = 0.4, \beta_{\mathcal{Y}} = 1.0, \beta_{\mathcal{A}} = 1.0, \beta_{\mathcal{N}} = 3.5$ give the best results. We try from 0.001 to 0.01 for the learning rate and choose 0.003 as the default. The weight for strong vs. weak edges is $2 : 1$, and for synonym vs. antonym edges is $1 : -1$. Since both types of paths are sampled independently, the edge weights have importance only within their group.

Testing After we train the model, we extract the output word embeddings $\mathbb{O}(w)$, and use them for the benchmark evaluation. We train each model three times and report the average performance.

4.3 Benchmarks

We evaluate our model on word similarity benchmarks, which provide several pairs of words with a human-evaluated similarity score. We calculate the cosine similarity between the embedding vectors for each pair and rank the values from largest to smallest. Then, we use Spearman rank correlation to compare the order of pairs for each model versus the ground truth. A higher value indicates that the rank order of word pairs generated by a model is similar to the ground truth.

We collect the following datasets: Card-660 (Pilehvar et al., 2018), MC-30 (Miller and Charles, 1991), MEN-TR-3K (Bruni et al., 2014), MTurk-287 (Radinsky et al., 2011), MTurk-771 (Halawi et al., 2012), RG-65 (Rubenstein and Goodenough, 1965), RW-STANFORD (Luong et al., 2013), SimLex999 (Hill et al., 2015), SimVerb-3500 (Gerz et al., 2016), WS-353-ALL (Finkelstein et al., 2001), WS-353REL (Finkelstein et al., 2001), WS-353-SIM (Finkelstein et al., 2001), YP-130 (Yang and Powers, 2006). Among these datasets, MEN-TR-3K, MTurk-771, RG-65, YP-130, and WS-353-REL focus on testing the relatedness, and SimLex999, SimVerb-3500, and WS-353-SIM focus on testing the similarity between the word pairs. WS-353-ALL is the combination of WS-353-REL and WS-353-SIM. Card-660 and RW-STANFORD focus on testing rare words.

After generating the Spearman correlation of each dataset, we also report the weighted average over all the datasets, with the weight of each dataset being proportional to its size.

4.4 Baseline Methods

We compare HG2Vec with Word2Vec(Mikolov et al., 2013), Dict2Vec(Tissier et al., 2017), and DRG2Vec(Shu et al., 2020). For comparison, We include HG2Vec(wiki), for which we use the 50M Wikipedia corpus with dictionaries and thesauri as the input resource. We also compare with a target-only based model HG2Vec(target) where we replace $\mathcal{C}(t)$ with t in Eq. (4). This allows us to compare the effectiveness of context-focused loss. Note also that we report the result of Dict2Vec and DRG2Vec based on our runs. If we use wiki corpus as an external source and we use paths sampled with TF-IDF pairs from strong and weak pairs, we obtain DRG2Vec. If we omit the graph and use wiki corpus with strong and weak pairs, we obtain Dict2Vec. Finally, we also report ablation results when we omit strong, weak, synonyms, antonyms, and negative pairs, one at a time.

5 Evaluation

5.1 Similarity and Relatedness Experiments

Effectiveness of Dictionaries and Thesauri The results of evaluating the similarity and relatedness benchmarks are shown in Table 1. We can see that HG2Vec scores higher than Word2Vec,

Type	Benchmark	Word2Vec	Dict2Vec	DRG2Vec	HG2Vec wiki	HG2Vec target	HG2Vec
Relatedness	MEN-TR-3K	0.612	0.688	0.721	0.771	**0.782**	0.779
	MTurk-287	0.577	0.568	0.558	**0.666**	0.642	0.650
	MTurk-771	0.540	0.609	0.640	0.755	**0.778**	0.773
	RG-65	0.617	0.814	0.845	0.895	**0.915**	0.913
	WS-353-REL	0.548	0.579	0.605	**0.671**	0.658	0.660
	YP-130	0.257	0.528	0.610	0.756	**0.780**	0.771
Similarity	SimLex999	0.338	0.444	0.476	0.735	0.742	**0.752**
	SimVerb-3500	0.190	0.379	0.425	0.732	0.732	**0.742**
	WS-353-SIM	0 .679	0.696	0.728	**0.813**	0.811	0.811
Rare Words	Card-660	0.234	0.388	0.513	0.667	0.657	**0.668**
	RW-STANFORD	0.398	0.476	0.482	0.549	0.552	**0.558**
Mixed Words	MC-30	0.682	0.748	0.738	0.867	0.898	**0.900**
	WS-353-ALL	0.626	0.682	0.699	**0.747**	0.736	0.734
	Average	0.416	0.529	0.558	0.712	0.717	**0.721**

Table 1: Evaluation of HG2Vec versus baseline methods for similarity and relatedness benchmarks. Spearman rank correlation values shown for each benchmark, as well as a weighted average across datasets. Best results in bold.

Dict2Vec, and DRG2Vec on all the tests. Among the baselines, DRG2Vec typically outperforms others except for mixed words datasets. Our HG2Vec model improves the performance on all the relatedness tests. For example, compared to DGR2Vec, on MTurk-771 it has 20.8% higher performance, and on RG-65 8.0% higher. We have a considerable improvement on similarity tests, such as SimLex999 (57.9% higher) and SimVerb-3500 (74.6% higher). Likewise, we see substantial improvements on rare words. Looking at average correlation scores, HG2Vec outperforms DGR2Vec by 29.2%, Dict2Vec by 36.3%, and Word2Vec by 73.3%.

These improvements show that dictionaries and thesauri are suitable replacements for a large corpus for training word embeddings since our model does not make use of any text corpus. Furthermore, since Dict2Vec also uses dictionaries as a resource, the improvement in similarity benchmarks indicates that thesauri are a powerful resource for learning the similarity of word embeddings. Other models only use negative sampling to decrease the similarity score between dissimilar word pairs, which can contain high uncertainty. In contrast, thesauri contain both synonyms and antonyms which are more reliable sources for our model to learn the word pairs with similar and opposite meanings.

Corpus Resource HG2Vec(wiki) includes both the Wikipedia corpus and the dictionaries and thesauri as the input resource. Compared with HG2Vec, HG2Vec(wiki) has better performance on MTurk-287 by 2.4 % and WS-353-REL by 1.7 %, which are both relatedness datasets. In contrast, HG2Vec has 2.3 % improvement on SimLex999 and 1.4 % improvement on SimVerb-3500. It shows that combining Wikipedia corpus can increase the performance of relatedness tests but can also decrease the performance on similarity. However, since incorporating the Wikipedia corpus does not lead to a significant improvement, and the data pre-processing and training time are not negligible, we choose to discard them from input resources.

Context-focused versus Target-only loss Compared with the target-only model HG2Vec(target), our context-focused model HG2Vec increases the performance for similarity benchmarks while keeping most of the performance for relatedness benchmarks. Context-focused model has 1.3% improvement on SimLex999 and SimVerb-3500. Furthermore, the context-focused model increases the performance on benchmarks with rare words. It has a 1.7% improvement on Card-660. Target-only model only has a smaller improvement in relatedness tests, such as a 1.2% improvement in YP-130.

5.2 Ablation Tests

We conduct ablation tests to examine the contribution of each type of edge to the final Spearman correlation, as shown in Table 2. We can see that synonym and negative pairs contribute to both relatedness and similarity tasks. Strong pairs help the models learn relatedness but negatively im-

Type	Benchmark	-strong	-weak	-synonym	-antonym	-negative	HG2Vec
Relatedness	MEN-TR-3K	0.762	0.688	**0.788**	0.782	0.426	0.779
	MTurk-287	0.610	0.517	0.642	**0.677**	0.318	0.650
	MTurk-771	0.755	0.707	0.756	0.770	0.562	**0.773**
	RG-65	0.902	0.826	0.899	**0.913**	0.715	**0.913**
	WS-353-REL	0.642	0.542	0.641	0.650	0.359	**0.660**
	YP-130	0.754	0.756	0.727	**0.815**	0.660	0.771
Similarity	SimLex999	**0.771**	0.725	0.636	0.711	0.644	0.752
	SimVerb-3500	**0.755**	0.715	0.626	0.694	0.671	0.742
	WS-353-SIM	0.794	0.748	0.784	**0.816**	0.591	0.811
Rare Words	Card-660	**0.683**	0.578	0.589	0.682	0.169	0.668
	RW-STANFORD	0.553	0.555	0.504	0.569	0.352	**0.558**
Mixed Words	MC-30	0.898	0.826	0.888	**0.901**	0.707	0.900
	WS-353-ALL	0.720	0.654	0.720	0.732	0.499	**0.734**
	Average	0.718	0.679	0.665	0.707	0.515	**0.721**

Table 2: Ablation study: Column indicates the omitted word pairs; HW2Vec is the full model. Best results in bold.

keyword	DRG2Vec			HG2Vec		
Adjective Query Words						
red	reds	yellow	blue	carmine	crimson	vermillion
	purple	green	orange	scarlet	fuchsia	ponceau
local	locals	regional	district	regional	parochial	locals
	mudir	municipal	stations	subregional	provincial	endemic
Verb Query Words						
sleep	sleeps	sleeping	awake	doze	snooze	shuteye
	asleep	quiescent	waking	slumber	catnap	siesta
listen	call	attentively	hear	listened	listeners	relisten
	listening	listened	callers	listenin	heed	listener
Noun Query Words						
animal	animals	plant	ruminant	critter	varmint	brute
	person	pets	organism	beast	mammal	coyote
coffee	tea	grape	cocoa	coffees	coffea	canephora
	beverage	beans	drinks	tea	arabica	coffeebeans

Table 3: Qualitative Evaluation: Top 6 similar words to the query word.

pact some similarity tasks, such as SimLex999 and SimVerb-3500 (the model improves by removing strong pairs for these datasets). For some similarity datasets, omitting synonyms can sometimes help (e.g., MEN-TR-3K). Likewise, for antonyms (on YP-130). Overall, HG2Vec balances between relatedness and similarity tasks to yield the highest performance.

5.3 Case Study

To qualitatively evaluate the word embeddings, we divide the words into three types: adjectives, verbs, and nouns. We sample query words from each category and report the top six words with the highest cosine similarity to the query word in Table 3.

We observe that for the adjective query *red*, HG2Vec word pairs contain several of its synonyms, such as *vermillion* and *scarlet*. DRG2Vec lists other colors that co-occur in similar contexts, like *blue* and *green*. Likewise, for the adjective *local*, HW2Vec returns high similarity words, but DRG2Vec generates nouns that frequently combine with *local*, such as *stations*. In general, HG2Vec outperforms other models since it can generate more synonyms of the input word.

Comparing verb and noun queries is somewhat harder, but overall, HG2Vec focuses more on semantics, whereas DRG2Vec pays more attention to co-occurrence. For example, DRG2Vec outputs *awake* as similar to the query verb *sleep*, but they are in fact opposite in meaning. On the other hand, the top results for HG2Vec are all similar to *sleep*. Likewise, for the noun query *animal*, DRG2Vec returns *person*, and for *coffee* it returns *grape*, as one of the top similarity words. HG2Vec avoids such obviously non-similar words. This suggests that models can struggle to learn opposite word pairs without thesauri if their contexts are similar. HG2Vec performs better because of the synonym and antonym edges.

6 Conclusions

We propose HG2Vec, a model that solely relies on dictionaries and thesauri without the use of a huge textual corpus. Despite this, our model yields state-of-art on many similarity and relatedness benchmarks. We also provide strong evidence (via the ablation study) that learning synonyms and antonyms is necessary for improved word embedding models. Our work highlights the use of dictionaries and thesauri as effective sources to learn word embeddings.

One interesting finding is that when we parse the thesaurus, some word pairs are both synonyms and antonyms. For example, according to Roget's Super Thesaurus, the synonyms of *advance* are *help, hasten, quicken, aid, assist, further, forward, facilitate, back*, and the antonyms of *back* are *progress, advance*. The reason *back* is a synonym is because *to advance someone* is *to back them*. On the other hand, *back* is also an antonym since *to go back* is the opposite of *to advance*. Thus, *advance* and *back* are both synonyms and antonyms, suggesting that the relationship between two words may be opposite in different contexts and situations. This is a challenging topic for training word embeddings, which we plan to investigate in the future.

In the future, we plan to also include more relationship types, such as derivative words and participles. Furthermore, while HG2Vec uses definitions only to construct strong and weak pairs, we plan to study ways of using the full definition text in addition to the heterogeneous graph.

References

Tom Bosc and Pascal Vincent. 2018. Auto-encoding dictionary definitions into consistent word embeddings. In *EMNLP*, pages 1522–1532.

Elia Bruni, Nam-Khanh Tran, and Marco Baroni. 2014. Multimodal distributional semantics. *Journal of artificial intelligence research*, 49:1–47.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Manaal Faruqui, Jesse Dodge, Sujay K Jauhar, Chris Dyer, Eduard Hovy, and Noah A Smith. 2014. Retrofitting word vectors to semantic lexicons. *arXiv preprint arXiv:1411.4166*.

Lev Finkelstein, Evgeniy Gabrilovich, Yossi Matias, Ehud Rivlin, Zach Solan, Gadi Wolfman, and Eytan Ruppin. 2001. Placing search in context: The concept revisited. In *Proceedings of the 10th international conference on World Wide Web*, pages 406–414.

Daniela Gerz, Ivan Vulić, Felix Hill, Roi Reichart, and Anna Korhonen. 2016. Simverb-3500: A large-scale evaluation set of verb similarity. *arXiv preprint arXiv:1608.00869*.

Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864.

Guy Halawi, Gideon Dror, Evgeniy Gabrilovich, and Yehuda Koren. 2012. Large-scale learning of word relatedness with constraints. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1406–1414.

Felix Hill, Roi Reichart, and Anna Korhonen. 2015. Simlex-999: Evaluating semantic models with (genuine) similarity estimation. *Computational Linguistics*, 41(4):665–695.

Abhik Jana, Siddhant Haldar, and Pawan Goyal. 2022. Network embeddings from distributional thesauri for improving static word representations. *Expert Systems with Applications*, 187:115868.

Douwe Kiela, Felix Hill, Stephen Clark, et al. 2015. Specializing word embeddings for similarity or relatedness. In *EMNLP*, pages 2044–2048. Citeseer.

Divesh R Kubal and Anant V Nimkar. 2019. A survey on word embedding techniques and semantic similarity for paraphrase identification. *International Journal of Computational Systems Engineering*, 5(1):36–52.

Omer Levy and Yoav Goldberg. 2014. Neural word embedding as implicit matrix factorization. *Advances in neural information processing systems*, 27.

Minh-Thang Luong, Richard Socher, and Christopher D Manning. 2013. Better word representations with recursive neural networks for morphology. In *Proceedings of the seventeenth conference on computational natural language learning*, pages 104–113.

Marc McCutcheon. 2010. *Roget's Super Thesaurus*. Writer's Digest Books.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

George A Miller and Walter G Charles. 1991. Contextual correlates of semantic similarity. *Language and cognitive processes*, 6(1):1–28.

Kim Anh Nguyen, Sabine Schulte im Walde, and Ngoc Thang Vu. 2016. Integrating distributional lexical contrast into word embeddings for antonym-synonym distinction. *arXiv preprint arXiv:1605.07766*.

Mohammad Taher Pilehvar, Dimitri Kartsaklis, Victor Prokhorov, and Nigel Collier. 2018. Card-660: Cambridge rare word dataset-a reliable benchmark for infrequent word representation models. *arXiv preprint arXiv:1808.09308*.

Kira Radinsky, Eugene Agichtein, Evgeniy Gabrilovich, and Shaul Markovitch. 2011. A word at a time: computing word relatedness using temporal semantic analysis. In *Proceedings of the 20th international conference on World wide web*, pages 337–346.

Herbert Rubenstein and John B Goodenough. 1965. Contextual correlates of synonymy. *Communications of the ACM*, 8(10):627–633.

Elena Sofia Ruzzetti, Leonardo Ranaldi, Michele Mastromattei, Francesca Fallucchi, and Fabio Massimo Zanzotto. 2021. Lacking the embedding of a word? look it up into a traditional dictionary. *arXiv preprint arXiv:2109.11763*.

Xiaobo Shu, Bowen Yu, Zhenyu Zhang, and Tingwen Liu. 2020. Drg2vec: Learning word representations from definition relational graph. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–9. IEEE.

Julien Tissier, Christophe Gravier, and Amaury Habrard. 2017. Dict2vec: Learning word embeddings using lexical dictionaries. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 254–263.

Dongqiang Yang and David MW Powers. 2006. *Verb similarity on the taxonomy of WordNet*. Masaryk University.

Wenhao Yu, Chenguang Zhu, Yuwei Fang, Donghan Yu, Shuohang Wang, Yichong Xu, Michael Zeng, and Meng Jiang. 2021. Dict-bert: Enhancing language model pre-training with dictionary. *arXiv preprint arXiv:2110.06490*.

Juexiao Zhang, Yubei Chen, Brian Cheung, and Bruno A Olshausen. 2019. Word embedding visualization via dictionary learning. *arXiv preprint arXiv:1910.03833*.