Learning Graph Embeddings from WordNet-based Similarity Measures

Andrey Kutuzov[1], **Mohammad Dorgham**[2], **Oleksiy Oliynyk**[2], **Chris Biemann**[2], and **Alexander Panchenko**[2,3]

[1]Language Technology Group, University of Oslo, Oslo, Norway
[2]Language Technology Group, University of Hamburg, Hamburg, Germany
[3]Skolkovo Institute of Science and Technology, Moscow, Russia
andreku@ifi.uio.no
{5dorgham,6oliinyk,biemann,panchenko}@informatik.uni-hamburg.de

Abstract

We present *path2vec*, a new approach for learning graph embeddings that relies on structural measures of pairwise node similarities. The model learns representations for nodes in a dense space that approximate a given user-defined graph distance measure, such as e.g. the shortest path distance or distance measures that take information beyond the graph structure into account. Evaluation of the proposed model on semantic similarity and word sense disambiguation tasks, using various WordNet-based similarity measures, show that our approach yields competitive results, outperforming strong graph embedding baselines. The model is computationally efficient, being orders of magnitude faster than the direct computation of graph-based distances.

1 Introduction

Developing applications making use of large graphs, such as networks of roads, social media users, or word senses, often involves the design of a domain-specific graph **node similarity measure** $sim : V \times V \rightarrow \mathbb{R}$ defined on a set of nodes V of a graph $G = (V, E)$. For instance, it can represent the shortest distance from home to work, a community of interest in a social network for a user, or a semantically related sense to a given synset in WordNet (Miller, 1995). There exist a wide variety of such measures greatly ranging in their complexity and design from simple deterministic ones, e.g. based on shortest paths in a network (Leacock and Chodorow, 1998) to more complex ones, e.g. based on random walks (Fouss et al., 2007; Pilehvar and Navigli, 2015; Lebichot et al., 2018). Naturally, the majority of such measures rely on walks along edges E of the graph, often resulting in effective, but prohibitively inefficient measures requiring complex and computationally expensive graph traversals. Also, there are measures that in addition take e.g. corpus information into account beyond what is directly given in the graph, see e.g. (Budanitsky and Hirst, 2006). We propose a solution to this problem by decoupling development and use of graph-based measures. Namely, once a node similarity measure is defined, we learn vector representations of nodes that enable efficient computation of this measure. We represent nodes in a graph with dense embeddings that are good in approximating such custom, e.g. application-specific, pairwise node similarity measures. Similarity computations in a vector space are several orders of magnitude faster than computations directly using the graph. Additionally, graph embeddings can be of importance in privacy-sensitive network datasets, since in this setup, explicitly storing edges is not required anymore. The main advantage over other graph embeddings is that our model can learn a custom user-defined application or domain specific similarity measure.

We show the effectiveness of the proposed approach *intrinsically* on a word similarity task, by learning synset vectors of the WordNet graph based on several similarity measures. Our model is not only able to closely *approximate* various measures, but also *to improve* the results of the original measures in terms of (1) correlation with human judgments and (2) computational efficiency, with gains up to 4 orders of magnitude. Our method outperforms other state-of-the-art graph embeddings models.

Besides, we evaluate it *extrinsically* in a WSD task (Navigli, 2009) by replacing the original structural measures with their vectorized counterparts in a graph-based WSD algorithm by Sinha and Mihalcea (2007), reaching comparable performance. Because of being inspired by the *word2vec* architecture, we dub our model '*path2vec*'[1] mean-

[1]https://github.com/uhh-lt/path2vec

ing it encodes paths (or other similarities) between graph nodes into dense vectors.

Our **first contribution** is an effective and efficient approach to learn graph embeddings based on a user-defined custom similarity measure sim on a set of nodes V, e.g. the shortest path distance. The **second contribution** is an application of state-of-the-art graph embeddings to word sense disambiguation task.

2 Related Work

Various methods have been employed in NLP to derive lexical similarity directly from geometrical properties of the WordNet graph, from random walks in (Rao et al., 2008) to kernels in (Ó Séaghdha, 2009). More recently, representation learning on graphs (Bordes et al., 2011) received much attention in various research communities; see (Hamilton et al., 2017a) for a thorough survey on the existing methods. All of them (including ours) are based on the idea of projecting graph nodes into a latent vector space with a much lower dimensionality than the number of nodes.

The method described in this paper falls into the category of 'shallow embeddings', meaning that we do not attempt to embed entire communities or neighborhoods: our aim is to approximate distances or similarities between (single) nodes. Existing approaches to solving this task mostly use either factorization of the graph adjacency matrix (Cao et al., 2015; Ou et al., 2016) or random walks over the graph as in *Deepwalk* (Perozzi et al., 2014) and *node2vec* (Grover and Leskovec, 2016). A completely different approach is taken by Subercaze et al. (2015), who directly embed the WordNet tree graph into Hamming hypercube binary representations. Their model is dubbed 'Fast similarity embedding' (*FSE*) and also optimizes one of our objectives, i.e. to provide a much quicker way of calculating semantic similarities based on WordNet knowledge. However, the *FSE* embeddings are not differentiable, limiting their use in many deep neural architectures, especially if fine-tuning is needed.

TransE (Bordes et al., 2013) interprets entities as vectors in the low-dimensional embeddings space and relations as a translation operation between two entity vectors. For a triplet (head, relation, tail) which holds, the embedding of the tail is close to the embedding of the head plus embedding of the relation. TransH (Wang et al., 2014) models each relation as a specific hyperplane and projects entity vectors onto the hyperplane. If connection holds then projected vectors of head and tail are connected by a translation vector with low error. As a result, entities have different representations for hyperplanes of different relations where they are involved. *TransR* (Lin et al., 2015) extends *TransE* (Bordes et al., 2013) and *TransH* (Wang et al., 2014), and is based on the idea that an entity may have a few aspects and different relations are focused on them. So the same entities can be close or far from each other depending on the type of the relation. *TransR* projects entity vectors into a relation specific space, and learns embeddings via translation between projected entities.

We quantitatively compare *path2vec* to these methods in Section 5. We did not compare our approach to the *GraphSAGE* embeddings (Hamilton et al., 2017b) and Graph Convolutional Networks (Schlichtkrull et al., 2018), since they make use of input node features, which are absent in our setup.

Also note that unlike retro-fitting and similar techniques (Rothe and Schütze, 2015; Pilehvar and Collier, 2016; Mrkšić et al., 2017), our approach does not use any training corpus or pre-trained input embeddings. The synset representations are trained on the WordNet graph alone.

3 Learning Graph Metric Embeddings

Definition of the Model The *path2vec* model learns low-dimensional vectors for the graph nodes $\{v_i, v_j\} \in V$ (synsets in the case of WordNet) such that the dot products between pairs of the respective vectors $(\mathbf{v}_i \cdot \mathbf{v}_j)$ are close to the user-defined similarities between the nodes s_{ij}. This first component of the objective encodes potentially long distances in the graph (the global structure). In addition, the model reinforces direct connections between nodes: We add to the objective similarities $\mathbf{v}_i \cdot \mathbf{v}_n$ and $\mathbf{v}_j \cdot \mathbf{v}_m$ between the nodes v_i and v_j and their respective adjacent nodes $\{v_n : \exists(v_i, v_n) \in E\}$ and $\{v_m : \exists(v_j, v_m) \in E\}$ to preserve local structure of the graph. Therefore, the model preserves both global and local relations between nodes by minimizing the following loss function $\mathcal{L}$:

$$\frac{1}{|B|} \sum_{(v_i,v_j)\in B} \left((\mathbf{v}_i^T \mathbf{v}_j - s_{ij})^2 - \alpha(\mathbf{v}_i^T \mathbf{v}_n + \mathbf{v}_j^T \mathbf{v}_m) \right),$$

where $s_{ij} = sim(v_i, v_j)$ is the value of a 'gold' similarity measure between a pair of nodes v_i and

v_j, $\mathbf{v}_i$ and $\mathbf{v}_j$ are the embeddings of the first and the second node, B is a training batch, α is a regularization coefficient. The second term ($\mathbf{v}_i \cdot \mathbf{v}_n + \mathbf{v}_j \cdot \mathbf{v}_m$) in the objective function is a regularizer which aids the model to simultaneously maximize the similarity between adjacent nodes (which is maximum by definition) while learning the similarity between the two target nodes.

We use negative sampling to form a training batch B adding n negative samples ($s_{ij} = 0$) for each real ($s_{ij} > 0$) training instance: each real node (synset) pair (v_i, v_j) with 'gold' similarity s_{ij} is accompanied with n 'negative' node pairs (v_i, v_k) and (v_j, v_l) with zero similarities, where v_k and v_l are randomly sampled nodes from V. Embeddings are initialized randomly and trained with the *Adam* optimizer (Kingma and Ba, 2014) with early stopping.[2]

Once the model is trained, the computation of node similarities is approximated with the dot product of the learned node vectors, making the computations efficient: $\hat{s}_{ij} = \mathbf{v}_i \cdot \mathbf{v}_j$.

Relation to Similar Models Our model is similar to the Skip-gram model (Mikolov et al., 2013), where pairs of words (v_i, v_j) from a training corpus are optimized to have their corresponding vectors dot product $\mathbf{v}_i \cdot \tilde{\mathbf{v}}_j$ close to 1, while randomly generated pairs ('negative samples') are optimized to have their dot product close to 0. In the Skip-gram model, the target is to minimize the log likelihood of the conditional probabilities of context words v_j given current words v_i, which is in the case on the negative sampling amounts to minimization of: $\mathcal{L} = -\sum_{(v_i,v_j)\in B_p} \log \sigma(\mathbf{v}_i \cdot \tilde{\mathbf{v}}_j) - \sum_{(v_i,v_j)\in B_n} \log \sigma(-\mathbf{v}_i \cdot \tilde{\mathbf{v}}_j)\Big)$, where B_p is the batch of positive training samples, B_n is the batch of the generated negative samples, and σ is the sigmoid function. The model uses local information. However, in *path2vec*, the target values s_{ij} for the dot product are not binary, but can take arbitrary values in the $[0...1]$ range, depending on the path-based measure on the input graph, e.g. the normalized shortest path length in the WordNet between *motor.n.01* and *rocket.n.02* is 0.389.

Further, in our model there is no difference between 'word' and 'context' spaces: we use a single embedding matrix, with the number of rows equal to the number of nodes and column width set to the desired embedding dimensionality. Finally, unlike the Skip-gram, we do not use any non-linearities.

Another closely related model is Global Vectors (GloVe) (Pennington et al., 2014), which approximates the co-occurrence probabilities in a given corpus. The objective function to be minimized in GloVe model is $\mathcal{L} = \sum_{(v_i,v_j)\in B} f(s_{ij})(\mathbf{v}_i \cdot \tilde{\mathbf{v}}_j - \log s_{ij} + b_i + b_j)^2$, where s_{ij} counts the number of co-occurrence of words v_i and v_j, b_i and b_j are additional biases for each word, and $f(s_{ij})$ is a weighting function to give appropriate weight for rare co-occurrences. Like the Skip-gram, GloVe also uses two embedding matrices, but it relies on global information.

4 Computing Pairwise Similarities

4.1 Selection of the Similarity Measures

Our aim is to produce node embeddings that capture given similarities between nodes in a graph. In our case, the graph is WordNet, and the nodes are its 82,115 noun synsets. We focused on nouns since in WordNet and SimLex999 they are represented better than other parts of speech. Embeddings for synsets of different part of speech can be generated analogously.

The training datasets consist of pairs of noun synsets and their 'ground truth' similarity values. There exist several methods to calculate synset similarities on the WordNet (Budanitsky and Hirst, 2006). We compile four datasets, with different similarity functions: Leacock-Chodorow similarities (*LCH*); Jiang-Conrath similarities calculated over the SemCor corpus (*JCN-S*); Wu-Palmer similarities (*WuP*); and Shortest path similarities (*ShP*). *LCH* similarity (Leacock and Chodorow, 1998) is based on the shortest path between two synsets in the WordNet hypernym/hyponym taxonomy and its maximum depth, while *JCN* similarity (Jiang and Conrath, 1997) uses the lowest common parent of two synsets in the same taxonomy. *JCN* is significantly faster but additionally requires a corpus as a source of probabilistic data about the distributions of synsets ('information content'). We employed the SemCor subset of the Brown corpus, manually annotated with word senses (Kucera and Francis, 1982).

WuP similarities (Wu and Palmer, 1994) are based on the depth of the two nodes in the taxonomy and the depth of their most specific ances-

[2] In our experiments, we identified the optimal values of $n = 3$ negative samples, batch size of $|B| = 100$, training for 15 epochs, $\alpha = 0.01$. We report on the influence of the embedding dimensionality parameter d in Section 5. We found it also beneficial to use additionally L_1 weight regularization.

tor node. *ShP* is a simple length of the shortest path between two nodes in the graph. We used the NLTK (Bird et al., 2009) implementations of all the aforementioned similarity functions.

Pairwise similarities for all synset pairs can be pre-computed. For the 82,115 noun synsets in the WordNet, this results in about 3 billion unique synset pairs. Producing these similarities using 10 threads takes about 30 hours on an Intel Xeon E5-2603v4@1.70GHz CPU for *LCH*, and about 5 hours for *JCN-S*. The resulting similarities lists are quite large (45 GB compressed each) and thus difficult to use in applications. But they can be used in *path2vec* to *learn* dense embeddings $\mathbb{R}^d$ for these 82,115 synsets, such that $d \ll 82,115$ and the dot products between the embeddings approximate the 'raw' WordNet similarity functions.

4.2 Pruning the Dissimilar Pairs of Nodes

In principle, one can use all unique synset pairs with their WordNet similarities as the training data. However, this seems impractical. As expected due to the small-world nature of the WordNet graph (Steyvers and Tenenbaum, 2005), most synsets are not similar at all: with *JCN-S*, the overwhelming majority of pairs feature similarity very close to zero; with *LCH*, most pairs have similarity below 1.0. Thus, we filtered low-similarity pairs out, using similarity threshold of 0.1 for the *JCN-S* and *ShP* datasets, 0.3 for the *WuP* dataset and 1.5 for the *LCH* dataset (due to substantial differences in similarities distributions, as shown in Figure 1). This dramatically reduced the size of the training data (e.g., to less than 1.5 million pairs for the *JCN-S* dataset and to 125 million pairs for the *LCH* dataset), thus making the training much faster and at the same time *improving* the quality of the resulting embeddings (see the description of our evaluation setup below).

With this being the case, we additionally pruned these reduced datasets by keeping only the first 50 most similar 'neighbors' of each synset: the rationale behind this is that some nodes in the WordNet graph are very central and thus have many neighbors with high similarity, but for our procedure only the nearest/most similar ones suffice. This again reduced training time and improved the results, so we hypothesize that such pruning makes the models more generally applicable and more focused on the meaningful relations between synsets. The final sizes of the pruned

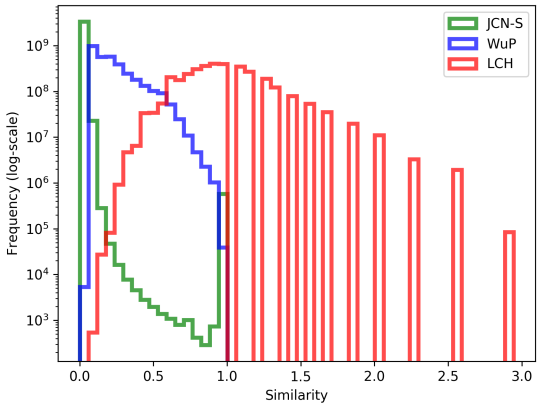

Figure 1: Distribution of similarities between WordNet noun synsets with different distance measures.

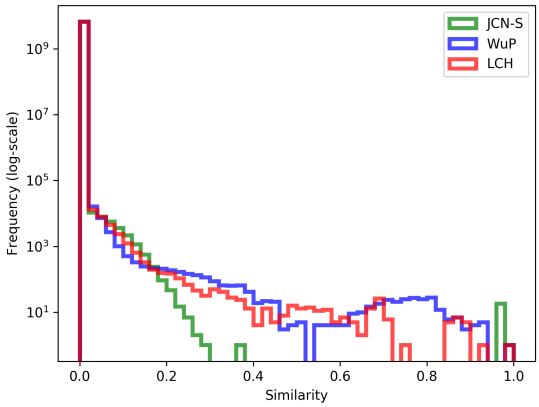

Figure 2: Distributions of pairwise similarities in *path2vec* models trained on different measures.

training datasets are 694,762 pairs for the *JCN-S*, 4,008,446 pairs for the *LCH*, 4,063,293 pairs for the *ShP* and 4,100,599 pairs for the *WuP*[3].

Note also that the *LCH* similarity can take values well above 1.0. After the pruning, we scaled similarities in all datasets to the $[0...1]$ range by unity-based normalization. Also, in some rare cases, NLTK produces *JCN* similarities of infinitely large values (probably due to the absence of particular synsets in SemCor). We clipped these similarities to the value of 1. All the datasets were shuffled prior to training.

[3]All the datasets and the trained graph embeddings can be downloaded from https://github.com/uhh-lt/path2vec

5 Experiment 1: Intrinsic Evaluation based on Semantic Similarity

Experimental Setting It is possible to evaluate the models by calculating the rank correlation of their cosine similarities with the corresponding similarities for all the unique pairs from the training dataset, or at least a large part of them. Subercaze et al. (2015) evaluated their approach on *LCH* similarities for all unique noun synset pairs from WordNet Core (about 5 million similarities total); their model achieves Spearman rank correlation of 0.732 on this task. However, this kind of evaluation does not measure the ability of the model to produce meaningful predictions, at least for language data: the overwhelming part of these unique pairs are synsets not related to each other at all. For most tasks, it is useless to 'know' that, e.g., '*ambulance*' and '*general*' are less similar than '*ambulance*' and '*president*'. While the distances between these node pairs are indeed different on the WordNet graph, we find it much more important for the model to be able to robustly tell really similar pairs from the unrelated ones so that they could benefit applications.

As a more balanced and relevant test set, we use noun pairs (666 total) from the SimLex999 semantic similarity dataset (Hill et al., 2015). SimLex999 contains lemmas; as some lemmas may map to several WordNet synsets, for each word pair we choose the synset pair maximizing the WordNet similarity, following (Resnik, 1999). Then, we measure the Spearman rank correlation between these 'gold' scores and the similarities produced by the graph embedding models trained on the WordNet. Further on, we call this evaluation score the 'correlation with WordNet similarities'. This evaluation method directly measures how well the model fits the training objective[4].

We also would like to check whether our models generalize to extrinsic tasks. Thus, we additionally used human-annotated semantic similarities from the same SimLex999. This additional evaluation strategy directly tests the models' correspondence to human judgments independently of WordNet. These correlations were tested in two synset selection setups, important to distinguish:

1. WordNet-based synset selection (static synsets): this setup uses the same lemma-to-synset mappings, based on maximizing WordNet similarity for each SimLex999 word pair with the corresponding similarity function. It means that all the models are tested on exactly the same set of synset pairs (but the similarities themselves are taken from SimLex999, not from the WordNet).

2. Model-based synset selection (dynamic synsets): in this setup, lemmas are converted to synsets dynamically as a part of the evaluation workflow. We choose the synsets that maximize word pair similarity *using the vectors from the model itself*, not similarity functions on the WordNet. Then the resulting ranking is evaluated against the original SimLex999 ranking.

The second (dynamic) setup in principle allows the models to find better lemma-to-synset mappings than those provided by the WordNet similarity functions. This setup essentially evaluates two abilities of the model: 1) to find the best pair of synsets for a given pair of lemmas (sort of a disambiguation task), and 2) to produce the similarity score for the chosen synsets. We are not aware of any 'gold' lemma-to-synset mapping for SimLex999, thus we directly evaluate only the second part, but implicitly the first one still influences the resulting scores. Models often choose different synsets. For example, for the word pair '*atom, carbon*', the synset pair maximizing the *JCN-S* similarity calculated on the 'raw' WordNet would be 'atom.n.02 (*'a tiny piece of anything'*), carbon.n.01 (*'an abundant nonmetallic tetravalent element'*)' with the similarity 0.11. However, in a *path2vec* model trained on the same gold similarities, the synset pair with the highest similarity 0.14 has a different first element: 'atom.n.01 (*'the smallest component of an element having the chemical properties of the element'*)', which seems to be at least as good a decision as the one from the raw WordNet.

Baselines *path2vec* is compared against five baselines (more on them in Section 2): *raw WordNet similarities* by respective measures; *Deepwalk* (Perozzi et al., 2014); *node2vec* (Grover and Leskovec, 2016); *FSE* (Subercaze et al., 2015); and *TransR* (Lin et al., 2015).

DeepWalk, *node2vec*, and *TransR* models were trained on the same WordNet graph. We used all 82,115 noun synsets as vertices and hypernym/hyponym relations between them as edges. Since the *node2vec* C++ implementation accepts

[4]Note, however, that it does not mean testing on the training data: for example, 75% of synset pairs from the SimLex999 are **not present** in our pruned *JCN-S* training dataset; for the *LCH* dataset it is 82%. Evaluating these absent pairs only does not substantially change the results.

Model	*Selection of synsets* JCN-S	LCH	ShP	WuP
WordNet	1.0	1.0	1.0	1.0
TransR	0.568	0.776	0.776	0.725
node2vec	0.726	0.759	0.759	0.787
Deepwalk	0.775	0.868	0.868	0.850
FSE	0.830	0.900	0.900	0.890
path2vec	**0.931**	**0.935**	**0.952**	**0.931**

Table 1: Spearman correlation scores with WordNet similarities on the 666 noun pairs in SimLex999.

Model	*Selection of synsets* JCN-S	LCH	ShP	WuP
WordNet	0.487	**0.513**	**0.513**	0.474
TransR	0.394	0.395	0.395	0.379
node2vec	0.426	0.434	0.434	0.400
Deepwalk	0.468	0.468	0.468	0.450
FSE	0.490	0.502	0.502	0.483
path2vec	**0.501**	0.470	0.512	**0.491**

Table 2: Spearman correlations with human SimLex999 noun similarities (WordNet synset selection).

an edge list as input, we had to add a self-connection for all nodes (synsets) that lack edges in WordNet. During the training of *DeepWalk* and *node2vec* models, we tested different values for the number of random walks (in the range from 10 to 100), and the vector size (100 to 600). For *DeepWalk*, we additionally experimented with the window size (5 to 100). All other hyperparameters were left at their default values. *FSE* embeddings of the WordNet noun synsets were provided to us by the authors, and consist of 128-bit vectors.

Discussion of Results Table 1 presents the comparison of *path2vec* and the baselines with regards to how well they approximate the WordNet similarity functions output (the raw WordNet similarities always get the perfect correlation in this evaluation setup). All the reported rank correlation value differences in this and other tables are statistically significant based on the standard two-sided p-value. We report the results for the best models for each method, all of them (except *FSE*) using vector size 300 for comparability.

Path2vec outperform other baseline embeddings, achieving high correlation with the raw WordNet similarities. This shows that our simple model can approximate different graph measures. Figure 2 shows the similarities' distributions in the resulting models, reflecting the original measures' distributions in Figure 1.

As expected, vector dimensionality greatly influences the performance of all graph embedding models. As an example, Figure 3 plots the performance of the *path2vec* models trained on *JCN-S* and *WuP* datasets, when using 'dynamic synset selection' evaluation setup (that is, each model can decide for itself how to map SimLex999 lemmas to WordNet synsets). The red horizontal line is the correlation of WordNet similarities with SimLex999 human scores. For the *path2vec* models, there is a tendency to improve the performance when the vector size is increased, until a plateau is reached beyond 600 dimensions. Note that *Deepwalk*[5] does not benefit much from increased vector size, while *node2vec*[6] yields strangely low scores for 200 dimensions. Interestingly, *path2vec* and *Deepwalk* models consistently *outperform* the raw WordNet (this is also true for *FSE*). This means these embeddings are in some sense 'regularized', leading to better 'disambiguation' of senses behind SimLex999 word pairs and eventually to better similarities ranking.

In Tables 2 and 3, we select the best 300D *path2vec* models from the experiments described above and compare them against the best 300D baseline models and 128D FSE embeddings in static and dynamic evaluation setups. When WordNet-defined lemma-to-synset mappings are used (Table 2), the raw WordNet similarities are non-surprisingly the best, although *FSE* and *path2vec* embeddings achieve nearly the same performance (even slightly better for the *JCN-S* and *WuP* mappings). Following them are the *Deepwalk* models, which in turn outperform *node2vec* and *TransR*. In the dynamic synset selection setup (see Table 3), all the models except *node2vec* and *TransR* are superior to raw WordNet, and the best models are *FSE* and *path2vec ShP/WuP*, significantly outperforming the others. *Path2vec* models trained on *JCN-S* and *LCH* are on par with *Deepwalk* and much better than *node2vec* and *TransR*. We believe it to interesting, considering that it does not use random walks on graphs and is conceptually simpler than *FSE*.

Note that word embedding models trained on text perform *worse* than the WordNet-based em-

[5]The reported best *Deepwalk* models were trained with the number of walks 10 and window size 70.

[6]The reported best *node2vec* models were trained with the number of walks 25.

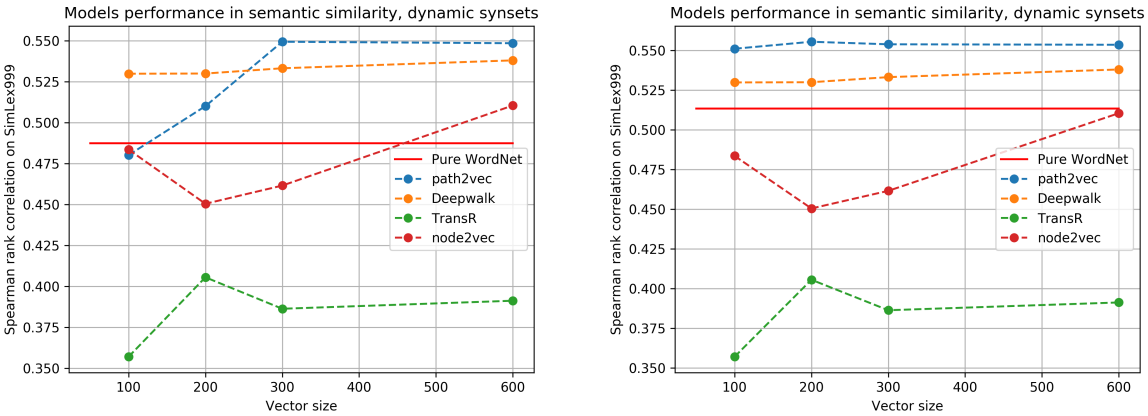

Figure 3: Evaluation on SimLex999 noun pairs, model-based synset selection: *JCN-S* (left) and *WuP* (right).

Model	*Correlation*
TransR (Lin et al., 2015)	0.386
node2vec (Grover and Leskovec, 2016)	0.462
Deepwalk (Perozzi et al., 2014)	0.533
FSE (Subercaze et al., 2015)	**0.556**
Raw WordNet JCN-S	0.487
Raw WordNet LCH	0.513
Raw WordNet ShP	0.513
Raw WordNet WuP	0.474
path2vec JCN-S	0.533
path2vec LCH	0.532
path2vec ShP	**0.555**
path2vec WuP	**0.555**

Table 3: Spearman correlations with human SimLex999 noun similarities (model synset selection).

beddings (including *path2vec*) on the semantic similarity task. For example, the *word2vec* model of vector size 300 trained on the Google News corpus (Mikolov et al., 2013) achieves Spearman correlation of only 0.449 with SimLex999, when testing only on nouns. The *GloVe* embeddings (Pennington et al., 2014) of the same vector size trained on the Common Crawl corpus achieve 0.404.

6 Experiment 2: Extrinsic Evaluation based on Word Sense Disambiguation

Experimental Setting As an additional extrinsic evaluation, we turned to word sense disambiguation task, reproducing the WSD approach from (Sinha and Mihalcea, 2007). The original algorithm uses WordNet similarities; we tested how using dot products and the learned embeddings instead will influence the WSD performance.

The employed WSD algorithm starts with building a graph where the nodes are the WordNet synsets of the words in the input sentence. The nodes are then connected by edges weighted with the similarity values between the synset pairs (only if the similarity exceeds a threshold, which is a hyperparameter; we set it to 0.95). The final step is selecting the most likely sense for each word based on the weighted in-degree centrality score for each synset (in case of ties, the first synset is chosen). Figure 4 shows a graph generated for the sentence '*More often than not, ringers think of the church as something stuck on the bottom of the belfry*'. Note that we disambiguate nouns only.

Discussion of Results Table 4 presents the WSD micro-F1 scores using raw WordNet similarities, 300D *path2vec*, *Deepwalk* and *node2vec* models, and the 128D *FSE* model. We evaluate on the following all-words English WSD test sets: Senseval-2 (Palmer et al., 2001), Senseval-3 (Mihalcea et al., 2004), and SemEval-15 Task 13 (Raganato et al., 2017). Raw WordNet similarities are still the best, but the *path2vec* models are consistently the second after them (and orders of magnitude faster), outperforming other graph embedding baselines. The largest drop between the original and vector-based measures is for *JCN-S*, which is also the only one which relies not only on graph but also on external information from a corpus, making it more difficult to approximate (see also Figure 2, where this measure distribution seems to be the most difficult to reproduce). Note that both the original graph-based measures and graph embeddings do not outperform the most frequent sense (MFS) baseline, which is in line with the original algorithm (Sinha and Mihalcea, 2007).

Here our aim was not to improve WSD systems

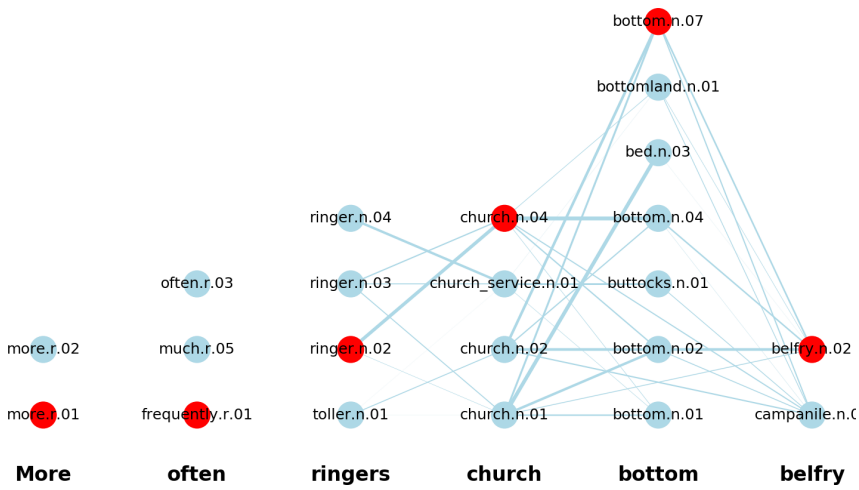

Figure 4: A sentence graph for WSD, where a column lists all the possible synsets of a corresponding word.

Model	Senseval2	Senseval3	SemEval-15
Random sense	0.381	0.312	0.393
Baselines (various graph embeddings)			
TransR	0.540	0.466	0.536
node2vec	0.503	0.467	0.489
Deepwalk	0.528	0.476	0.552
FSE	0.536	0.476	0.523
WordNet (graph-based measures)			
JCN-S	**0.620**	**0.558**	**0.597**
LCH	0.547	0.494	0.550
ShP	0.548	0.495	0.550
WuP	0.547	0.487	0.542
path2vec (vector-based measures)			
JCN-S	0.511	0.463	0.508
LCH	0.527	0.472	0.536
ShP	0.534	**0.489**	**0.563**
WuP	**0.543**	**0.489**	0.545

Table 4: F1 scores on all-words WSD tasks.

but to compare *path2vec* against other graph embedding methods in an extrinsic, task-based evaluation. This is also the reason why we do not compare against many other existing WordNet-based WSD systems: we are interested only in the approaches which learn dense representations of graph nodes, as *path2vec* does.

7 Computational Efficiency Evaluation

Pairwise Similarity Computation One of the reasons to use *path2vec* embeddings is computational efficiency. Directly employing the WordNet graph to find semantic similarities between synsets is expensive. The dot product computation is much faster as compared to shortest path computation (and other complex walks) on a large graph. Also, dense low-dimensional vector representations of nodes take much less space than the pairwise similarities between all the nodes.

The time complexity of calculating the shortest path between graph nodes (as in *ShP* or *LCH*) is in the best case linear in the number of nodes and edges (Leacock and Chodorow, 1998). *JCN-S* compares favorably since it is linear in the height of the taxonomy tree (Jiang and Conrath, 1997); however, it still cannot leverage highly-optimized routines and hardware capabilities, which makes the use of vectorized representations so efficient. Calculating Hamming distance between binary strings (as in the *FSE* algorithm) is linear in the sum of string lengths, which are equivalent of vector sizes (Hamming, 1950). At the same time, the complexity of calculating dot product between float vectors (as in *path2vec*) is linear in the vector size by the definition of the dot product and is easily and routinely parallelized.

As an example, let us consider the popular problem of ranking the graph nodes by their similarity to one particular node of interest (finding the 'nearest neighbors'). Table 5 shows the time for computing similarities of one node to all other WordNet noun nodes, using either standard graph similarity functions from NLTK, Hamming distance between 128D binary embeddings, or dot product between a 300D float vector (representing this node) and all rows of a 82115×300 matrix. Using float vectors is 4 orders of magnitude faster than *LCH*, 3 orders faster than *JCN*, and 2 orders faster than Hamming distance.

Construction of the Training Set Despite its computational efficiency at test time, construct-

Model	*Running time*
LCH in NLTK	30 sec.
JCN-S in NLTK	6.7 sec.
FSE embeddings	0.713 sec.
path2vec and other float vectors	**0.007** sec.

Table 5: Computation of 82,115 similarities between one noun synset and all other noun synsets in WordNet.

ing a training dataset for *path2vec* (following the workflow described in Section 4) requires calculating pairwise similarities between all possible pairs of graph nodes. This leads to a number of similarity calculations quadratic in the number of nodes, which can be prohibitive in case of very large graphs. However, instead of this, the training datasets for *path2vec* can be constructed much faster by taking the graph structure into account. In essence, this implies finding for each node v the set of other nodes directly connected to it or to its direct graph neighbors (set of second order graph neighbors, V_2). Then, graph similarity is calculated only for the pairs consisting of each v and the nodes in their respective V_2; these pairs constitute the training dataset (the same thresholds and normalization procedures apply).

The amount of pairwise similarity calculations is then linear in the number of nodes times the average number of neighbors in V_2, which is much better. Particularly, in the case of WordNet, each node (synset) has 36 synsets in its V_2 on average, and half of the nodes do not have any neighbors at all. Thus, only 2,935,829 pairwise similarity calculations are needed, 1,000 times less than when calculating similarities between all synset pairs.

Following that, e.g., the training dataset for *JCN-S* can be constructed in 3 minutes, instead of 5 hours, with similar speedups for other graph distance measures. The training datasets constructed in this 'fast' way showed negligible performance decrease compared to the 'full' datasets ($0.07...0.03$ drop in the semantic similarity experiments, and < 0.03 drop in the WSD experiments). It means that when using *path2vec* in practical tasks, one can construct the training dataset very quickly, preserving embeddings performance.

8 Discussion and Conclusion

We presented *path2vec*, a simple, effective, and efficient model for embedding graph similarity measures. It can be used to learn vector representations of graph nodes, approximating shortest path distances or other node similarity measures of interest. Additionally, if the similarity function is based on the shortest path, this paves the way to a quick and efficient calculation of the shortest distance between two nodes in large graphs.

Our model allow for much more efficient graph distances calculations (3 or 4 orders of magnitude faster depending on a similarity measure). In applications one could replace path-based measures with dot product between *path2vec* embeddings, gaining significant speedup in distance computation between nodes. Thus, our model could be used to speed up various other graph-based algorithms that make use of node distance computations, such as Floyd (1962) algorithm, Dijkstra (1959) algorithm, or algorithms for computing node betweenness centrality (Brandes, 2001).

In this paper, we used our model to learn embeddings of WordNet synsets and showed that in the semantic similarity task, the resulting representations perform better than the state-of-the-art graph embedding approaches based on random walks. Interestingly, the learned embeddings can outperform the original WordNet similarities on which they were trained. *path2vec* was also evaluated on the WSD task (it has not been done before for graph embeddings, to our knowledge), again outperforming other approaches.

However, *path2vec* can be trained on arbitrary graph measures and is not restricted to the shortest path or to only tree-structured graphs. In the future, we plan to explore the possibility of training embeddings able to approximate multiple similarity metrics at once. Another direction of further research is to apply our model to other types of data, such as social networks or graph of roads.

Acknowledgements

This work has been partially supported by Deutsche Forschungsgemeinschaft (DFG) within the JOIN-T (grant BI 1544/4-1 and SP 1999/1-1)) project and the ACQuA project (grant BI 1544/7-1 and HA 5851/2-1), which is part of the Priority Program Robust Argumentation Machines (RATIO) (SPP-1999), and Young Scientist Mobility Grant from the Faculty of Mathematics and Natural Sciences, University of Oslo. We thank three anonymous reviewers for their most useful feedback. Last but not least, we are grateful to Sarah Kohail who helped with computing the first version of the *node2vec* baselines.

References

Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural language processing with Python: analyzing text with the natural language toolkit*. O'Reilly Media, Inc.

Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 2787–2795. Curran Associates, Inc.

Antoine Bordes, Jason Weston, Ronan Collobert, and Yoshua Bengio. 2011. Learning structured embeddings of knowledge bases. In *Twenty-Fifth AAAI Conference on Artificial Intelligence*, pages 301–306, San Francisco, CA, USA. AAAI Press.

Ulrik Brandes. 2001. A faster algorithm for betweenness centrality. *Journal of mathematical sociology*, 25(2):163–177.

Alexander Budanitsky and Graeme Hirst. 2006. Evaluating WordNet-based measures of lexical semantic relatedness. *Computational Linguistics*, 32(1):13–47.

Shaosheng Cao, Wei Lu, and Qiongkai Xu. 2015. Grarep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pages 891–900. ACM.

Edsger W Dijkstra. 1959. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271.

Robert W Floyd. 1962. Algorithm 97: shortest path. *Communications of the ACM*, 5(6):345.

Francois Fouss, Alain Pirotte, Jean-Michel Renders, and Marco Saerens. 2007. Random-walk computation of similarities between nodes of a graph with application to collaborative recommendation. *IEEE Transactions on knowledge and data engineering*, 19(3):355–369.

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. ACM.

William Hamilton, Rex Ying, and Jure Leskovec. 2017a. Representation learning on graphs: Methods and applications. *IEEE Data Engineering Bulletin*, 40(3):52–74.

William Hamilton, Zhitao Ying, and Jure Leskovec. 2017b. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1024–1034.

Richard Hamming. 1950. Error detecting and error correcting codes. *Bell System technical journal*, 29(2):147–160.

Felix Hill, Roi Reichart, and Anna Korhonen. 2015. SimLex-999: Evaluating Semantic Models With (Genuine) Similarity Estimation. *Computational Linguistics*, 41(4):665–695.

Jay J. Jiang and David W. Conrath. 1997. Semantic similarity based on corpus statistics and lexical taxonomy. In *Proceedings of the 10th Research on Computational Linguistics International Conference*, pages 19–33, Taipei, Taiwan. The Association for Computational Linguistics and Chinese Language Processing (ACLCLP).

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

Henry Kucera and Nelson Francis. 1982. *Frequency analysis of English usage: Lexicon and grammar*. Boston: Houghton Mifflin.

Claudia Leacock and Martin Chodorow. 1998. Combining local context and WordNet similarity for word sense identification. *WordNet: An electronic lexical database*, 49(2):265–283.

Bertrand Lebichot, Guillaume Guex, Ilkka Kivimäki, and Marco Saerens. 2018. A constrained randomized shortest-paths framework for optimal exploration. *arXiv preprint arXiv:1807.04551*.

Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. 2015. Learning entity and relation embeddings for knowledge graph completion. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, volume 15, pages 2181–2187, Austin, TX, USA. AAAI Press.

Rada Mihalcea, Timothy Chklovski, and Adam Kilgarriff. 2004. The Senseval-3 English lexical sample task. In *Senseval-3: Third International Workshop on the Evaluation of Systems for the Semantic Analysis of Text*, pages 25–28, Barcelona, Spain. Association for Computational Linguistics.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S. Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems 26*, pages 3111–3119, Lake Tahoe, NV, USA. Curran Associates, Inc.

George A. Miller. 1995. WordNet: A lexical database for English. *Communications of the ACM*, 38(11):39–41.

Nikola Mrkšić, Ivan Vulić, Diarmuid Ó Séaghdha, Ira Leviant, Roi Reichart, Milica Gašić, Anna Korhonen, and Steve Young. 2017. Semantic specialization of distributional word vector spaces using monolingual and cross-lingual constraints. *Transactions of the Association for Computational Linguistics*, 5:309–324.

Roberto Navigli. 2009. Word sense disambiguation: A survey. *ACM Computing Surveys (CSUR)*, 41(2):10.

Diarmuid Ó Séaghdha. 2009. Semantic classification with wordnet kernels. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers*, pages 237–240, Boulder, CO, USA. Association for Computational Linguistics.

Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. 2016. Asymmetric transitivity preserving graph embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1105–1114. ACM.

Martha Palmer, Christiane Fellbaum, Scott Cotton, Lauren Delfs, and Hoa Trang Dang. 2001. English tasks: All-words and verb lexical sample. In *Proceedings of SENSEVAL-2 Second International Workshop on Evaluating Word Sense Disambiguation Systems*, pages 21–24, Toulouse, France. Association for Computational Linguistics.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710, New York, NY, USA. ACM.

Mohammad Taher Pilehvar and Nigel Collier. 2016. De-conflated semantic representations. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1680–1690, Austin, TX, USA. Association for Computational Linguistics.

Mohammad Taher Pilehvar and Roberto Navigli. 2015. From senses to texts: An all-in-one graph-based approach for measuring semantic similarity. *Artificial Intelligence*, 228:95–128.

Alessandro Raganato, Jose Camacho-Collados, and Roberto Navigli. 2017. Word sense disambiguation: A unified evaluation framework and empirical comparison. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 99–110, Valencia, Spain. Association for Computational Linguistics.

Delip Rao, David Yarowsky, and Chris Callison-Burch. 2008. Affinity measures based on the graph Laplacian. In *Coling 2008: Proceedings of the 3rd Textgraphs workshop on Graph-based Algorithms for Natural Language Processing*, pages 41–48, Manchester, UK. Coling 2008 Organizing Committee.

Philip Resnik. 1999. Semantic similarity in a taxonomy: An information-based measure and its application to problems of ambiguity in natural language. *Journal of Artificial Intelligence Research*, 11(1):95–130.

Sascha Rothe and Hinrich Schütze. 2015. Autoextend: Extending word embeddings to embeddings for synsets and lexemes. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1793–1803, Beijing, China. Association for Computational Linguistics.

Michael Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. 2018. Modeling relational data with graph convolutional networks. In *European Semantic Web Conference*, pages 593–607, Heraklion, Greece. Springer.

Ravi Sinha and Rada Mihalcea. 2007. Unsupervised graph-based word sense disambiguation using measures of word semantic similarity. In *International Conference on Semantic Computing (ICSC)*, pages 363–369, Irvine, CA, USA. IEEE.

Mark Steyvers and Joshua B. Tenenbaum. 2005. The large-scale structure of semantic networks: statistical analyses and a model of semantic growth. *Cognitive science*, 29(1):41–78.

Julien Subercaze, Christophe Gravier, and Frédérique Laforest. 2015. On metric embedding for boosting semantic similarity computations. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 8–14, Beijing, China. Association for Computational Linguistics.

Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. 2014. Knowledge graph embedding by translating on hyperplanes. In *AAAI Conference on Artificial Intelligence*, pages 1112–1119, Québec City, QC, Canada.

Zhibiao Wu and Martha Palmer. 1994. Verb semantics and lexical selection. In *Proceedings of the 32nd Annual Meeting of the Association for Computational Linguistics*, pages 133–138, Las Cruces, NM, USA. Association for Computational Linguistics.