

---

# Learning Word Representations by Embedding the WordNet Graph

---

**Thibault Cordier**

ENS Paris Saclay - Master MVA

thibault.cordier@ens-paris-saclay.fr

**Antoine Tadros**

ENS Paris Saclay - Master MVA

antoine.tadros@ens-paris-saclay.fr

**Pascal Denis, Rémi Gilleron and Nathalie Vauquier**

INRIA Lille–Nord Europe

59650 Villeneuve d’Ascq, France

pascal.denis@inria.fr

remi.gilleron@univ-lille.fr

nathalie.vauquier@inria.fr

## Abstract

The overall objective of this project consists to produce word representations with the help of existing lexical databases like WordNet. We proceed by constructing similarity graphs over the 80k WordNet noun synsets - using various synset similarity measures - and then extracting noun synset representations by embedding these graphs - applying recent node embedding algorithms - in order to map these representations into word representations. We evaluate these against standard word similarity datasets.

## 1 Introduction

How to adequately represent words as vectors is a long-standing and crucial problem in the fields of Text Mining and Natural Language Processing (NLP). This question has recently re-surfaced due to the recent surge of research in “deep” neural networks, and the development of algorithms for learning distributed word representations – or “word embeddings” – (the best known of which is probably word2vec [6, 7]). Typically, these approaches directly construct word representations from large amounts of unannotated texts, and don’t make use of any linguistic resource.

Due to the ubiquity of networks in many real world applications, and the need for better graph analytics tools, another recent trend of research has been in developing graph embedding techniques [3, 5]. One specific problem is node embedding, where the goal is to encode the graph nodes as low-dimensional vectors that faithfully summarize their graph position and the topology of their local neighborhood. Several new deep learning algorithms have been proposed for node embedding (e.g., node2vec [4], deepwalk [8]), in addition to already well-established matrix factorization-based approaches like Local Linear Embedding (LLE) [9] or Laplacian Eigenmaps [1]. These approaches have been used for different types of graphs such as knowledge graphs and semantic graphs.

### 1.1 Goals of the Paper

The overall objective is to improve word representations with the help of existing lexical databases like WordNet. For this, our supervisors aim to combine word embedding techniques from texts with node embedding techniques over WordNet. This paper is a preliminary step in this direction. The goal is to explore recent node embedding algorithms, in particular node2vec and deepwalk, to learn synset embeddings from the WordNet lexical database and derive word representations.

The tentative work-plan is as follows:

- Review the relevant literature on word and graph embedding methods.
- Construct similarity graphs over the 80k WordNet noun synsets, using various synset similarity algorithms [2].
- Apply node2vec and deepwalk on these similarity graphs to derive noun synset representations.
- Map these synset representations into word representations, and evaluate these against standard word similarity datasets.

## 2 WordNet database

WordNet®<sup>1</sup> is a broad lexical database of English words. Nouns, verbs, adjectives and adverbs are grouped into synonym sets (**synsets**), each expressing an unique lexical concept and interlinked by a variety of relations (conceptual-semantic and lexical relations).

WordNet can be compared to a **thesaurus**, in the sense that it groups words together based on their meanings - here words that denote the same concept and are interchangeable in many contexts.

However, some differences are worth noting. First, WordNet interlinks not just word forms - strings of letters - but specific senses of words. (That is why words that are found in close proximity to one another in the network are semantically disambiguated.) Second, WordNet labels the semantic relations among words, whereas the groupings of words in a thesaurus does not follow any explicit pattern other than meaning similarity. Third, WordNet represents word forms with several distinct meanings in as many distinct synsets. (Thus, each form-meaning pair in WordNet is unique.)

The main relation among words in WordNet is **synonymy**. The backbone of the noun network is the subsumption hierarchy (hyponymy/hypernymy), which accounts for close to 80% of the relations. The nine types of relations defined on the noun subnetwork, in addition to the synonymy relation that is implicit in each node are: the **hyponymy** (IS-A) relation, and its inverse, **hypernymy**; six **meronymic** (PART-OF) relations - COMPONENT-OF, MEMBER-OF and SUBSTANCE-OF - and their inverses; and **antonymy**, the COMPLEMENT-OF relation.

The entire WordNet network consists of 117,000 synsets and the WordNet noun network is represented by 82,125 synsets (which accounts for more than 70% of all WordNet). Subsequently, we propose to limit our study to the WordNet noun network in this project to produce word representations for each word in this network.

## 3 Word Embedding: Problem Formalization

The problem of "how to adequately represent words as vectors" - that we propose to solve with the help of existing lexical databases like WordNet - is usually a **word embedding** problem with graph analytics method.

Graph is naturally adapted to represent WordNet. According to the terminology of [3], WordNet can be defined as a *Knowledge Graph with Auxiliary Information* (potentially based on an information content). The first challenge consists to know "how to explore global consistency between different types of relations in WordNet" and/or "how to incorporate the rich and unstructured information in the embedding". One possibility consists to determine a semantic relatedness (a more general concept than similarity) between all synsets in order to construct a **similarity graph** - a *Homogeneous Graph* which both nodes and edges belong to a single type respectively - over the WordNet noun synsets.

Given a similarity graph over noun synsets, **graph embedding** is an efficient way to represent synsets as vectors. More particularly, **node embedding** represents each node as a vector in a low dimensional space in which the graph structural information and graph properties are maximumly preserved. (Nodes that are "close" in the graph are embedded to have similar vector representations.) The second challenge consists to know "how to encode the proximity in the learnt embeddings". One solution

---

<sup>1</sup>Christiane Fellbaum (1998, ed.) WordNet: An Electronic Lexical Database. Cambridge, MA: MIT Press.

Similarity Measures	
Name	Formula
Path Similarity	$\text{sim}_{\text{path}}(c_1, c_2) = \frac{1}{1 + \text{len}(c_1, c_2)}$
Leacock-Chodorow	$\text{sim}_{\text{LC}}(c_1, c_2) = -\log \frac{\text{len}(c_1, c_2)}{2 \times \max_{c \in \text{WordNet}} \text{depth}(c)}$
Wu-Palmer	$\text{sim}_{\text{WP}}(c_1, c_2) = \frac{\text{len}(c_1, \text{lso}_{1,2}) + \text{len}(c_2, \text{lso}_{1,2})}{\text{len}(c_1, \text{lso}_{1,2}) + \text{len}(c_2, \text{lso}_{1,2}) + 2 \times \text{depth}(\text{lso}_{1,2})}$
Resnik	$\text{sim}_{\text{R}}(c_1, c_2) = -\log p(\text{lso}_{1,2})$
Jiang-Conrath	$\text{sim}_{\text{JC}}(c_1, c_2) = \frac{1}{2 \log p(\text{lso}_{1,2}) - (\log p(c_1) + \log p(c_2))}$
Lin	$\text{sim}_{\text{L}}(c_1, c_2) = \frac{2 \log p(\text{lso}_{1,2})}{\log p(c_1) + \log p(c_2)}$

With  $\text{lso}_{1,2} = \text{lso}(c_1, c_2)$  the lowest super-ordinate of  $c_1$  and  $c_2$   
Table 1: Synset similarity measures studied in the paper.

is to consider this problem with an encoder-decoder perspective as suggested by [5]. Several deep learning algorithms - like node2vec and deepwalk - have been proposed for node embedding.

Then because we derive noun synset embeddings and we search to obtain word embeddings, we must propose **word sense disambiguation** method. The third challenge consists to know "how to map these synset representations into word representations".

From the perspective of problem setting, we propose to clearly define all the procedures of our problem previously stated, namely:

- **Similarity graph construction:** *from* WordNet graph *to* similarity graph
- **Node embedding:** *from* similarity graph *to* noun synset embeddings
- **Word sense disambiguation:** *from* noun synset embeddings *to* word embeddings

## 4 Similarity graph construction

Measures of **lexical semantic relatedness** are important tools in the field of Natural Language Processing (NLP) that can evaluate not only the similarity, but also the semantic relatedness between two concepts.

These measures are used in such applications as word sense disambiguation, determining the structure of texts, text summarization and annotation, information extraction and retrieval, automatic indexing, lexical selection, and the automatic correction of word errors in text.

More particularly, lexical semantic relatedness is a powerful proximity measure of concepts in WordNet - that can take into account the diversity of its conceptual-semantic and lexical relations - with which it is possible to derive a **similarity graph** over the 80k WordNet noun synsets.

The advantage brought by the use of similarity graphs - a *Homogeneous Graph* which both nodes and edges belong to a single type respectively unlike the WordNet network - is that its format is suitable for modelling a **graph embedding** problem. So construct similarity graph over the WordNet noun synsets is a preliminary step for synset embedding.

We propose to construct similarity graphs using various synset similarity algorithms as proposed and evaluated by [2]. We summarize each of the five approaches studied in this paper with short descriptions below as well as the similarity measures in the Table 1.

#### 4.1 Similarity by Computing Taxonomic Path Length

A simple way to compute semantic relatedness in a taxonomy such as WordNet is to view it as a graph and identify relatedness with path length between the concepts.

**Path Similarity** approach returns a score denoting how similar two word senses are, based on the shortest path that connects the senses in the IS-A (hypernym/hyponym) taxonomy. The score is in the range 0 to 1. A score of 1 represents identity i.e. comparing a sense with itself will return 1.

#### 4.2 Similarity by Scaling the Network

Despite its apparent simplicity, a widely acknowledged problem with the edge-counting approach is that it typically “relies on the notion that links in the taxonomy represent uniform distances”, which is typically not true. One solution consists to compute similarities by scaling the network.

**Leacock-Chodorow Similarity** approach [10] returns a score denoting how similar two word senses are, based on the shortest path that connects the senses (as above) and the maximum depth of the taxonomy in which the senses occur.

**Wu-Palmer Similarity** approach [11] returns a score denoting how similar two word senses are, based on the depth of the two senses in the taxonomy and that of their Least Common Subsumer (most specific ancestor node).

The LCS does not necessarily feature in the shortest path connecting the two senses, as it is by definition the common ancestor deepest in the taxonomy, not closest to the two senses. Typically, however, it will so feature. Where multiple candidates for the LCS exist, that whose shortest path to the root node is the longest will be selected. Where the LCS has multiple paths to the root, the longer path is used for the purposes of the calculation.

#### 4.3 Similarity with Information-based and Integrated Approaches

The final group of approaches that we present attempt to counter problems inherent in a general ontology by incorporating an additional, and qualitatively different, knowledge source, namely information from a corpus.

**Resnik Similarity** approach [12] returns a score denoting how similar two word senses are, based on the Information Content (IC) of the Least Common Subsumer (most specific ancestor node). This approach postulates that the semantic similarity is proportional to the probability of encountering an instance of the LCS concept.

**Jiang-Conrath Similarity** approach [13] returns a score denoting how similar two word senses are, based on the Information Content (IC) of the Least Common Subsumer (most specific ancestor node) and that of the two input Synsets. This approach postulates that the semantic distance of the link connecting a child-concept  $c$  to its parent-concept  $par(c)$  is proportional to the conditional probability  $p(c|par(c))$  of encountering an instance of  $c$  given an instance of  $par(c)$ .

**Lin Similarity** approach [14] returns a score denoting how similar two word senses are, based on the Information Content (IC) of the Least Common Subsumer (most specific ancestor node) and that of the two input Synsets. This approach derives from the *Similarity Theorem* that says: "The similarity between A and B is measured by the ratio between the amount of information needed to state their commonality and the information needed to fully describe what they are".

Note that for any similarity measure that uses information content, the result is dependent on the corpus used to generate the information content and the specifics of how the information content was created.

### 5 Node Embedding

Graph embedding techniques are methods that try to map a graph - the *Graph Embedding Input* - into a set of low-dimensional vectors - the *Graph Embedding Output* - while preserving its structure and the information it contains. By doing so, we get another representation of the graph which is more efficient when it comes to graph processing.

There is different type of embedding, depending on the needs of the specific application task. We can embed nodes, edges, sub-graph or the graph itself and represent it as a vector.

In our case and based on the taxonomies presented in [3], we are looking for a **node embedding** of the synsets which are the nodes of our graph.

As part of our study, we will limit our analysis to **Deep Learning based Graph Embedding with Random Walk** methods, namely the two most used algorithms node2vec [4] and deepwalk [8].

### 5.1 Deep Learning for Node Embedding with Random Walk

Instead of hand crafted features or expensive matrix factorisation approaches that doesn't scale well to real-life networks, these methods look for an embedding that maximize the probability of observing a network neighbourhood  $N_S(u)$  for a node  $u$  generated through a neighbourhood sampling strategy  $S$  conditioned on its embedding given by  $f$ .

The interest of DL Graph Embedding with Random Walk methods is that they can automatically exploit the neighbourhood structure through sampled paths on the graph. In fact, a graph is represented as a set of random walk paths sampled from it. The deep learning methods are then applied to the sampled paths for graph embedding which preserves graph properties carried by the paths.

**Formalization of the problem:** Let  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$  be a given graph. Let  $f : \mathcal{V} \rightarrow \mathbb{R}^d$  be the mapping function from the nodes space to a low dimension space of real value features we aim to learn -  $d$  is a parameter specifying the number of dimensions of our feature representation. We seek to optimize the following objective function:

$$\max_f \sum_{u \in \mathcal{V}} \log P(N_S(u)|f(u)) \quad (1)$$

In order to make the optimization problem tractable, two standard assumptions are made:

- Conditional independence:  $\forall u \in \mathcal{V}, P(N_S(u)|f(u)) = \prod_{n \in N_S(u)} P(n|f(u))$
- Symmetry in feature space:  $\forall n, u \in \mathcal{V}, P(n|f(u)) = \frac{\exp(f(n) \cdot f(u))}{\sum_{v \in \mathcal{V}} \exp(f(v) \cdot f(u))}$

Adopting an **encoder-decoder perspective** as suggested by [5], we can define our graph embedding problem in light of four methodological components:

- a *pairwise similarity* function  $s_{\mathcal{G}}$  defined over the graph  $\mathcal{G}$  (depending on the similarity measure  $\text{sim}_X$  of the graph  $\mathcal{G}$  and on the sampling strategy  $S$ ):

$$s_{\mathcal{G}} : \mathcal{V} \times \mathcal{V} \rightarrow \mathbb{R}^+, s_{\mathcal{G}}(n, u) = P_{\mathcal{G}, S}(n|u) \quad (2)$$

- an *encoder* function that maps nodes to vector embeddings:

$$\text{ENC} : \mathcal{V} \rightarrow \mathbb{R}^d, \text{ENC}(u) = f(u) \quad (3)$$

- a *decoder* function that maps pairs of node embeddings to a real-valued node similarity measure, which quantifies the similarity of the two nodes in the original graph:

$$\text{DEC} : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}, \text{DEC}(f(n), f(u)) = P(n|f(u)) \quad (4)$$

- a *loss function*  $l$  which determines how the quality of the pairwise reconstructions is evaluated in order to train the model:

$$l : \mathbb{R}^+ \times \mathbb{R}^+ \rightarrow \mathbb{R}, l(\text{DEC}(f(n), f(u)), s_{\mathcal{G}}(n, u)) = -\log(\text{DEC}(f(n), f(u))) \quad (5)$$

With this approach, we seek to minimize the following cross-entropy loss (with  $\mathcal{D}$  the training set generated by sampling random walks starting from each node):

$$\mathcal{L} = \sum_{(n, u) \in \mathcal{D}} -\log(\text{DEC}(f(n), f(u))) \quad (6)$$

Considering the optimisation problem of minimizing the cross-entropy loss force the algorithm to preserve the nodes similarities - because each sampling random walks starting from each node will favor the most similar neighbor nodes depending on the sampling strategy  $S$ .

In this type of algorithm, we first sample random walk paths on the graph. Those paths will serve as data for a deep learning method that will aim at preserving the properties and the information that it contains.

Both of the algorithm - node2vec and deepwalk that we will present - are expressed as likelihood maximization problem or cross-entropy loss minimization problem as precised lately. Their differences lie in their way of sampling paths on the graph and so in their way of exploiting the neighborhood structure.

## 5.2 DeepWalk

The algorithm of DeepWalk[8] consists of two main components: first a random walk generator, and second an update procedure.

**Random Walk Generator:** DeepWalk first samples paths from the input graph using **truncated random walks**. A walk samples uniformly from the neighbors of the last node visited - or proportionally to the edge weight from the last node to its neighbors - until the maximum length is reached. The algorithm specifies a number of random walks  $W$  of length  $L$  to start at each node.

**Update Procedure:** Then, it tries to maximize the co-occurrence probability among the nodes that appear within a window of size  $w$  in each path using **SkipGram** model. Optimization is done by stochastic gradient descent.

But calculate this probability (i.e the *decoder* function which is the softmax function) is not feasible because of the expensive calculation of the normalization factor with a high number of nodes (80k in our case).

There are usually two solutions to approximate the full softmax: hierarchical softmax and negative sampling[3]. DeepWalk algorithm associates his SkipGram model with hierarchical softmax.

The **hierarchical softmax** method gives to each node a path in a binary tree  $\mathcal{T}$  - each node is assigned to a leaf - that will help factorizing the expression of the likelihood and reduce it output dimension. Instead of enumerating all nodes, only the path from the root to the corresponding leaf needs to be evaluated. The optimization problem becomes maximizing the probability of a specific path in the tree.

Given two nodes  $u \in \mathcal{V}$ ,  $n \in \mathcal{V}$  and its relative path ( $b_0 = \text{ROOT}$ ,  $b_1, \dots, b_{|\log(\mathcal{V})|} = n$ ) in the binary tree  $\mathcal{T}$ , the hierarchical softmax consists to compute:

$$P(n|f(u)) = \prod_{i=1}^{|\log(\mathcal{V})|} P(b_i|f(u)) \quad (7)$$

where the probability to pass by  $b_i$  in  $\mathcal{T}$  conditioned on the embedding  $f(u)$  could be modeled by a binary classifier:

$$P(b_i|f(u)) = 1/(1 + \exp(-g(b_i).f(u))) \quad (8)$$

where  $g : \mathcal{V}(\mathcal{T}) \rightarrow \mathbb{R}^d$  is the mapping function that return the representation assigned to tree node  $b_i$ 's parent.

In this case, the hierarchical softmax reduces time complexity of SkipGram from  $\mathcal{O}(|\mathcal{V}|^2)$  to  $\mathcal{O}(|\mathcal{V}|\log(|\mathcal{V}|))$ . Moreover, optimization is operated by stochastic gradient descent on the model parameter set  $(f, g)$  where the size of each is  $\mathcal{O}(d|\mathcal{V}|)$ .

## 5.3 node2vec

The algorithm of node2vec[4] also consists of two main components: first a random walk generator, and second an update procedure

**Random Walk Generator:** node2vec first samples paths from the input graph using random walks. But here, the method of sampling the neighbourhood of nodes use a **biased random walk** strategy that combine:

- **Breadth-First Sampling** (BFS, neighbourhood is restricted to directly linked nodes)
- and **Depth-First Sampling** (DFS, neighbourhood is a sequence of node with increasing distances from source node, like DeepWalk path).

In the biased random walk strategy, consider a random walk that just traversed edge  $(t, u)$  and now resides at node  $u$ . Consider all the possible next nodes that can be taken  $\{v, v \in N_G(u)\}$ . Let  $d_{t,v}$  be the shortest path in terms of number of nodes between  $t$  and  $v$ . We introduce a bias :

$$\alpha_{p,q}(t, v) = \begin{cases} \frac{1}{p} & \text{if } d_{t,v} = 0 \\ 1 & \text{if } d_{t,v} = 1 \\ \frac{1}{q} & \text{if } d_{t,v} = 2 \end{cases}$$

The weights of the probability transition are updated to take into account this bias :  $\omega_{u,v} \leftarrow \omega_{u,v} * \alpha_{p,q}(t, v)$ , where  $\omega_{u,v}$  is the probability transition from a node  $u_i$  to  $u_j$ .

Intuitively, parameters  $p$  and  $q$  control how fast the walk explores and leaves the neighborhood of starting node  $u_0$ :

- If  $q > 1$  we tend to stay around node  $u_0$  as in the BFS strategy.
- If  $q < 1$  we tend to leave node  $u_0$  as in the DFS strategy.
- If  $p > \max(q, 1)$  we are less likely to sample a node that have been already visited in last two walks.
- If  $p < \min(q, 1)$  the walk will be close the initial node  $u_0$ .

**Update Procedure:** As in DeepWalk, node2vec tries to maximize the co-occurrence probability among the nodes that appear within a window of size  $w$  in each path with a **SkipGram** architecture using stochastic gradient decent - and meets the same difficulties as DeepWalk.

However, the node2vec algorithm considers the objective function in another form:

$$\max_f \sum_{u \in \mathcal{V}} -\log(Z_u) + \sum_{n \in N_S(u)} f(n) \cdot f(u) \quad (9)$$

where  $Z_u = \sum_{v \in \mathcal{V}} \exp(f(u) \cdot f(v))$  per-node partition function.

The latter is expensive to compute for large networks like WordNet and can be approximate using negative sampling. The key idea of **negative sampling** is to distinguish the target node from noises using logistic regression.

In this case, the negative sampling reduces time complexity of SkipGram from  $\mathcal{O}(|\mathcal{V}|^2)$  to  $\mathcal{O}(K|\mathcal{V}|)$  where  $K$  is the number of negative nodes that are sampled.

## 6 Word Embedding

Now that we have embedded the noun synsets of WordNet, we want to use it to represent words involved in the definitions of these noun synsets in the same real values space and so operate the final step of **Word Embedding**.

This problem can be assimilate as a **Word-Sense Disambiguation** (WSD) problem whose purpose is to identify which sense of a word (i.e. meaning) is used in a sentence, when the word has multiple meanings.

## 6.1 Word Sense Disambiguation

In WordNet, a word can belong to several synsets. We want to use this information to compute the low-dimension vector to represent this word as well as possible.

The most naive way is to take the **average vector** of all the synsets associated to the word. But we can imagine that a word is more often used for some of its meanings than for others.

From this remark, a representation of a word strongly depends on the field in which we wish to represent it. An idea would be to use word sense disambiguation (WSD) algorithm to estimate which sense represents the best the word depending on its context.<sup>2</sup>

**WSD algorithm** estimates which sense is the better suited for a word in a sentence by looking at the context (the meaning of the other words in the sentence).

By running a WSD algorithm on a corpus, we can derive the frequency at which each sense of a word appears. Then we can compute the word vector as the **weighted average vector** of all its senses and then propose one representation for the word.

## 6.2 Lesk Algorithms

We propose to use the **Lesk algorithm** which is a classical algorithm for word sense disambiguation introduced in [15]. The Lesk algorithm is based on the assumption that words in a given "neighborhood" (section of text) will tend to share a common topic.

There are different variants of this algorithm as *Original Lesk* (Lesk, 1986), *Adapted/Extended Lesk* (Banerjee and Pederson, 2002/2003), *Simple Lesk* (with definition, example(s) and hyper+hyponyms), *Cosine Lesk* (use cosines to calculate overlaps instead of using raw counts) and *Enhanced Lesk* (Basile et al. 2014) (in wishlist).

To give an example, the **Simple Lesk** algorithm is to compare the dictionary definition of an ambiguous word with the terms contained in its neighborhood and take the most significant meaning of the ambiguous word.

An implementation might look like this:

- for every sense of the word being disambiguated one should count the amount of words that are in both neighborhood of that word and in the dictionary definition of that sense
- the sense that is to be chosen is the sense which has the biggest number of this count

## 7 Experiments

### 7.1 Limitations

Let  $\mathcal{G} = (\mathcal{V}, \mathcal{E})$  be the similarity graph over the 80k noun synsets of WordNet. Let  $\mathcal{C}$  be a corpus used for add information content. We mention the limits encountered during our work.

We encountered constraints in computing and storage capacity due to the high number of synsets and relations in WordNet - and so due to the high number of nodes and edges in similarity graph.

**Similarity graph construction step:** Path, Leacock-Chodorow and Wu-Palmer Similarity are very greedy approaches to calculations - because they must explore WordNet to compute paths length - compared to Resnik, Jiang-Conrath and Lin Similarity -  $\mathcal{O}(\mathcal{C})$  to compute frequencies once and  $\mathcal{O}(1)$  to call them.

**Node embedding step:** node2vec and deepwalk algorithms must realize:

- Pre-process modified weights:  $\mathcal{O}(|\mathcal{E}|)$ , in the worst case  $\mathcal{O}(|\mathcal{V}|^2)$  if  $\mathcal{G}$  is full-connected.
- Random walk generation:  $\mathcal{O}(W.L.|\mathcal{V}|)$  with  $W$  the number of random walk of length  $L$ .
- SkipGram approximation:  $\mathcal{O}(|\mathcal{V}| \log(|\mathcal{V}|))$ .
- Update procedure:  $\mathcal{O}(d|\mathcal{V}|)$  with  $d$  the number of dimensions of our feature representation.

---

<sup>2</sup>So, it is natural for a word to have several embeddings because its representation depends on its context. But our task is to represent word with a unique representation taking into account each of its meanings.



Detail Results				
	Naive	Resnik	Jiang-Conrath	Lin
Similarity	$-0.044 (1.18 \times 10^{-2})$	$-0.045 (9.77 \times 10^{-3})$	<b>0.12</b> ( $1.03 \times 10^{-11}$ )	$0.039 (2.71 \times 10^{-2})$
ws353_relatedness	$-0.0098 (8.82 \times 10^{-1})$	$-0.081 (2.19 \times 10^{-1})$	$0.0028 (9.66 \times 10^{-1})$	$-0.066 (3.17 \times 10^{-1})$
rareword	$0.32 (8.73 \times 10^{-15})$	$0.25 (8.77 \times 10^{-10})$	$0.14 (6.77 \times 10^{-4})$	$0.25 (2.27 \times 10^{-9})$
rg56	$0.76 (1.56 \times 10^{-13})$	$0.69 (1.91 \times 10^{-10})$	$0.78 (2.22 \times 10^{-14})$	$0.71 (4.65 \times 10^{-11})$
ws353_similarity	$0.53 (5.87 \times 10^{-15})$	$0.39 (1.59 \times 10^{-8})$	$0.47 (5.98 \times 10^{-12})$	$0.42 (1.29 \times 10^{-9})$
mturk	$0.32 (7.58 \times 10^{-5})$	$0.31 (1.98 \times 10^{-4})$	$0.4 (9.42 \times 10^{-7})$	$0.35 (1.33 \times 10^{-5})$
wordsim353	$0.32 (3.30 \times 10^{-9})$	$0.2 (2.02 \times 10^{-4})$	$0.27 (5.92 \times 10^{-7})$	$0.23 (1.72 \times 10^{-5})$
men	$0.26 (7.05 \times 10^{-34})$	$0.22 (1.29 \times 10^{-24})$	$0.26 (2.00 \times 10^{-33})$	$0.29 (3.31 \times 10^{-42})$

Table 2: Pearson coefficient score (and p-value) with DeepWalk

Detail Results				
	Naive	Resnik	Jiang-Conrath	Lin
Similarity	$-0.068 (1.08 \times 10^{-4})$	$-0.044 (1.22 \times 10^{-2})$	<b>0.13</b> ( $5.75 \times 10^{-14}$ )	$0.042 (1.73 \times 10^{-2})$
ws353_relatedness	$-0.023 (7.31 \times 10^{-1})$	$-0.043 (5.16 \times 10^{-1})$	$-0.019 (7.77 \times 10^{-1})$	$-0.0045 (9.46 \times 10^{-1})$
rareword	$0.31 (2.29 \times 10^{-14})$	$0.27 (7.32 \times 10^{-11})$	$0.14 (7.22 \times 10^{-4})$	$0.27 (3.72 \times 10^{-11})$
rg56	$0.82 (1.43 \times 10^{-16})$	$0.7 (5.63 \times 10^{-11})$	$0.76 (2.64 \times 10^{-13})$	$0.75 (6.30 \times 10^{-13})$
ws353_similarity	$0.51 (4.90 \times 10^{-14})$	$0.43 (4.24 \times 10^{-10})$	$0.52 (1.05 \times 10^{-14})$	$0.51 (6.84 \times 10^{-14})$
mturk	$0.29 (3.45 \times 10^{-4})$	$0.22 (9.27 \times 10^{-3})$	$0.32 (1.18 \times 10^{-4})$	$0.46 (8.96 \times 10^{-9})$
wordsim353	$0.3 (3.76 \times 10^{-8})$	$0.23 (2.59 \times 10^{-5})$	$0.29 (1.38 \times 10^{-7})$	$0.3 (4.20 \times 10^{-8})$
men	$0.22 (1.44 \times 10^{-25})$	$0.23 (8.13 \times 10^{-28})$	$0.31 (9.63 \times 10^{-50})$	$0.35 (7.14 \times 10^{-61})$

Table 3: Pearson coefficient score (and p-value) with Node2Vec ( $p = 0.5$ ,  $q = 0.75$ )

## 7.2 Experimental setup

Based on previous considerations, we define our experimental setup:

**Similarity graph construction:** We choose to use various synset similarity measures as Resnik, Jiang-Conrath and Lin Similarity with Information Content derived from *brown* corpus. We use also a naive measure which is an alternative of Path Similarity measure where we just take the closer neighbours. Then in order to construct not full-connected graph, we opt for the k-nn graph construction method with  $k = 10$ .

**Node embedding algorithms:** We use deepwalk algorithm (with  $W = 20$ ,  $L = 80$ ,  $w = 10$ ,  $n = 10$ ) and node2vec algorithm (with  $W = 20$ ,  $L = 80$ ,  $w = 10$ ,  $n = 10$ ,  $p = 0.5$ ,  $q = 0.75$ ) where  $W$  is the number of random walk of length  $L$ ,  $w$  the window length of SkipGram model and  $n$  the number of iteration in SGD.  $p$  and  $q$  correspond to hyperparameters of node2vec.

**Word Embedding:** We choose to use the Simple Lesk algorithm for word sense disambiguation step with *brown* corpus to compute frequencies - to compute weighted average.

We carry out our experiments with the python environment and the necessary libraries:

- NLTK library: for WordNet interface, for Information Content derived from *brown* corpus and for similarity measures.
- node2vec library: from GitHub (eliorc/node2vec), for node2vec and deepwalk algorithms.
- pywsd library: from GitHub (alvations/pywsd), for WSD functions.
- mangoes library: from GitLab of INRIA (magnet/mangoes), for evaluation against standard word similarity datasets.

## 8 Results and Evaluation

To evaluate our embedding, we use *mangoes* built-in embedding evaluation module. It compares the similarities between words that our embedding gives to the one in several dataset with human-assigned similarity judgements. From the result presented in Table 2 and Table 3, we can notice that the best result on all the datasets is reached when using the graph built with the Jiang-Conrath similarity.

Nevertheless the results doesn't seem that good. The limitation in our hardware set up and the lack of time forces us to reduce the complexity of the graph that we had to process. Thus, we got less links between our synsets which unfortunately end up in a huge loss of information.

By looking at the graph properties, we saw that "pruning" the graph to get a shallow network made some synsets loose most of their connections. These adverse effects involve that some noun synsets in WordNet can not obtain embedding by deepwalk and node2vec. For example the synset "entity.n.01" have all its similarities so small that they underflow and are approximated to 0.

We can also investigate the choice of the hyper-parameters for the node embedding algorithms. But here again, our hardware limited our scope of action.

However we can still compare deepwalk to node2vec. And from the results, node2vec seems to give on most similarities and dataset a better embedding (with a higher Pearson correlation and a lower p-value associated).

## 9 Conclusion

The overall objective of this project consisted to produce word representations with the help of existing lexical databases like WordNet. We proceeded by constructing similarity graphs over the 80k WordNet noun synsets and then extracting noun synset representations by embedding these graphs. At the end we tried to map word representations in the low-dimension space obtained by the graph embedding.

We evaluated these against standard word similarity datasets. We can notice that the best result on all the datasets is reached when using the graph built with the node2vec algorithm with Jiang-Conrath similarity.

Our method is constrained by computing and storage capacity due to the high number of synsets and relations in WordNet - and so due to the high number of nodes and edges in similarity graph.

Due to are limited set up, we didn't have time to train as much as we wished deepwalk and node2vec to get more relevant results.

To continue this project, we propose to test all the similarity measures - if the computing and storage capacities allow it - for the similarity graphs construction and to test also the word sens disambiguation with more different methods.

For a future work, we propose to try the path2vec algorithm proposed in [16]. The authors claim that their algorithm outdo node2vec and deepwalk for synset graph embedding.

## References

- [1] M. Belkin and P. Niyogi. Laplacian Eigenmaps for Dimensionality Reduction and Data Representation. *Neural Computation*, 15(6):1373–1396, 2003.
- [2] A. Budanitsky and G. Hirst. Evaluating wordnet-based measures of lexical semantic relatedness. *Computational Linguistics*, 32(1):13–47, 2006.
- [3] H. Cai, V. W. Zheng, and K. C. Chang. A comprehensive survey of graph embedding: Problems, techniques and applications. *CoRR*, abs/1709.07604, 2017.
- [4] A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. abs/1607.00653, 2016.
- [5] W. L. Hamilton, R. Ying, and J. Leskovec. Representation learning on graphs: Methods and applications. *CoRR*, abs/1709.05584, 2017.
- [6] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. Technical report, arXiv:1301.3781, 2013.
- [7] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed Representations of Words and Phrases and their Compositionality. In *NIPS*, 2013.
- [8] B. Perozzi, R. Al-Rfou, and S. Skiena. Deepwalk: Online learning of social representations. *CoRR*, abs/1403.6652, 2014.
- [9] S. Roweis and L. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290:2323–2326, 2000.
- [10] Leacock, Claudia and Martin Chodorow. 1998. Combining local context and WordNet similarity for word sense identification. In Christiane Fellbaum, editor, *WordNet: An Electronic Lexical Database*. The MIT Press, Cambridge, MA, pages 265–283.
- [11] Wu, Zhibiao 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, June.
- [12] Resnik, Philip. 1995. Using information content to evaluate semantic similarity. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 448–453, Montreal, Canada, August.
- [13] Jiang, Jay J. and David W. Conrath. 1997. Semantic similarity based on corpus statistics and lexical taxonomy. In *Proceedings of International Conference on Research in Computational Linguistics (ROCLING X)*, Taiwan, pages 19–33.
- [14] Lin, Dekang. 1998. An information-theoretic definition of similarity. In *Proceedings of the 15th International Conference on Machine Learning*, Madison, WI, pages 296–304, July.
- [15] Lesk, Michael. “Automatic sense disambiguation using machine readable dictionaries: how to tell a pine cone from an ice cream cone.” *Proceedings of the 5th Annual International Conference on Systems Documentation*. ACM, 1986.
- [16] Kutuzov, Andrey & al., "Learning Graph Embeddings from WordNet-based Similarity Measure" arXiv:1808.05611