graph2vec: Learning Distributed Representations of Graphs

Annamalai Narayanan, Mahinthan Chandramohan, Rajasekar Venkatesan, Lihui Chen, Yang Liu and Shantanu Jaiswal
Nanyang Technological University, Singapore
annamala002@e.ntu.edu.sg,
{mahinthan,rajasekarv,elhchen,yangliu}@ntu.edu.sg,shantanu004@e.ntu.edu.sg

ABSTRACT

Recent works on representation learning for graph structured data predominantly focus on learning distributed representations of graph substructures such as nodes and subgraphs. However, many graph analytics tasks such as graph classification and clustering require representing entire graphs as fixed length feature vectors. While the aforementioned approaches are naturally unequipped to learn such representations, graph kernels remain as the most effective way of obtaining them. However, these graph kernels use handcrafted features (e.g., shortest paths, graphlets, etc.) and hence are hampered by problems such as poor generalization. To address this limitation, in this work, we propose a neural embedding framework named `graph2vec` to learn data-driven distributed representations of arbitrary sized graphs. `graph2vec`'s embeddings are learnt in an unsupervised manner and are task agnostic. Hence, they could be used for any downstream task such as graph classification, clustering and even seeding supervised representation learning approaches. Our experiments on several benchmark and large real-world datasets show that `graph2vec` achieves significant improvements in classification and clustering accuracies over substructure representation learning approaches and are competitive with state-of-the-art graph kernels.

Keywords

Graph Kernels, Deep Learning, Representation Learning

1. INTRODUCTION

Graph-structured data are ubiquitous nowadays in many domains such as social networks, cybersecurity, bio- and chemo-informatics. Many analytics tasks in these domains such as graph classification, clustering and regression require representing graphs as fixed-length feature vectors to facilitate applying appropriate Machine Learning (ML) algorithms. For instance, vectorial representations (*aka* embeddings) of programs' call graphs could be used to detect malware [6] and those of chemical compounds could be used to predict their properties such as solubility and anti-cancer activity [7].

Graph Kernels and handcrafted features. Graph kernels are one of the most prominent ways of catering the aforementioned graph analytics tasks. Graph kernels evaluate the similarity (*aka* kernel value) between a pair of graphs G and G' by recursively decomposing them into atomic substructures (e.g., random walks, shortest paths, graphlets etc.) and defining a similarity (*aka* kernel) function over the substructures (e.g., counting the number of common substructures across G and G'). Subsequently, kernel methods (e.g., Support Vector Machines (SVMs)) could be used for performing classification/clustering. However, these kernels exhibit two critical limitations: (1) Many of them do not provide explicit graph embeddings. This renders using general purpose ML algorithms which operate on vector embeddings (e.g., Random Forests (RFs), Neural Networks, etc.) unusable with graph data. (2) The substructures (i.e., walk, paths, etc.) leveraged by these kernels are 'handcrafted' i.e., determined manually with specific well-defined functions that help extracting such substructures from graphs. However, as noted by Yanardag and Vishwanathan [7], when such handcrafted features are used on large datasets of graphs, it leads to building very high dimensional, sparse and non-smooth representations and thus yield poor generalization. We note that replacing handcrafted features with ones that are learnt automatically from data could help to fix both the aforementioned limitations. In fact, in related domains such as text mining and computer vision, feature learning based approaches have outperformed handcrafted ones significantly across many tasks [2, 9].

Learning substructure embeddings. Recently, several approaches have been proposed to learn embeddings of graph substructures such as nodes [4], paths [7] and subgraphs [5, 6]. These embeddings can then be used directly in substructure based analytics tasks such as node classification, community detection and link prediction. However, these substructure representation learning approaches are incapable of learning representations of entire graphs and hence cannot be used for tasks such as graph classification. As we show through our experiments in §5, obtaining graph embeddings through trivial extensions such as averaging or max pooling over substructure embeddings leads to suboptimal results.

Learning task-specific graph embeddings. On the other hand, a few supervised approaches (i.e., ones that require class labels of graphs) to learn embeddings of entire

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

graphs such as PATCHY-SAN [9] have been proposed very recently. While they offer excellent performances in supervised learning tasks (e.g., graph classification) they pose two critical limitations that reduce their usability: (1) Being deep neural network based representation learning approaches, they require large volume of labeled data to learn meaningful representations. Obviously, obtaining such datasets is a challenge in itself as it requires mammoth labeling effort. (2) The representations thus learnt are specific to one particular ML task and cannot be used or transferred to other tasks or problems. For instance, the graph embeddings for the chemical compounds in the MUTAG dataset (see [7] for details) learnt using [9] are specifically designed to predict whether or not a compound has mutagenic effect on a bacterium. Hence, the same embeddings could not be used for any other tasks such as predicting the properties of the compounds. To circumvent these limitations, similar to the above mentioned substructure representation learning approaches, we need a completely unsupervised approach that can succinctly capture the generic characteristics of entire graphs in the form of their embeddings. To the best of our knowledge, there are no such techniques available. Hence driven by this motivation, in this work, we take the first steps towards learning task-agnostic representations of arbitrary sized graphs in an unsupervised fashion.

Our approach. To this end, we propose and develop a neural embedding framework named `graph2vec`. Inspired by the success of recently proposed neural document embedding models, we extend the same to learn graph embeddings. These document embedding models exploit the way how words/word sequences compose documents to learn their embeddings. Analogically, in `graph2vec`, we propose to view an entire graph as a document and the rooted subgraphs around every node in the graph as words that compose the document and extend document embedding neural networks to learn representations of entire graphs.

To the best of our knowledge, `graph2vec` is the first neural embedding approach that learns representations of whole graphs and it offers the following key advantages over graph kernels and other substructure embedding approaches:

1. **Unsupervised representation learning:** `graph2vec` learns graph embeddings in a completely unsupervised manner i.e., class labels of graphs are not required for learning their embeddings. This allows us to readily use `graph2vec` embeddings in a plethora of applications where labeled data is difficult to obtain.

2. **Task-agnostic embeddings:** Since `graph2vec` does not leverage on any task-specific information (e.g., class labels) for its representation learning process, the embeddings it provides are generic. This allows us to use these embeddings across all analytics tasks involving whole graphs. In fact, `graph2vec` embeddings could be used to seed supervised representation learning approaches such as [9].

3. **Data-driven embeddings:** Unlike graph kernels, `graph2vec` learns graph embeddings from a large corpus of graph data. This enables `graph2vec` to circumvent the aforementioned disadvantages of handcrafted feature based embedding approaches.

4. **Captures structural equivalence:** Unlike approaches such as `sub2vec` [5] which sample linear substructures (e.g., fixed length random walks) in a graph and learns graph embeddings from them, our framework samples and considers non-linear substructures, namely, rooted subgraphs for learning embeddings. Considering such non-linear substructures are known to preserve *structural equivalence*[1] and hence this ensures `graph2vec`'s representation learning process yields similar embeddings for structurally similar graphs.

Experiments. We determine `graph2vec`'s accuracy and efficiency in both supervised and unsupervised learning tasks with several benchmark and large real-world graph datasets. Also, we perform comparative analysis against several state-of-the-art substructure (e.g., node) representation learning approaches and graph kernels. Our experiments reveal that `graph2vec` achieves significant improvements in classification and clustering accuracies over substructure embedding methods and are highly competitive to state-of-the-art kernels. Specifically, on two real-world program analysis tasks, namely, malware detection and malware familial clustering, `graph2vec` outperforms state-of-the-art substructure embedding approaches by more than 17% and 39%, respectively.

Contributions. We make the following contributions:

- We propose `graph2vec`, an unsupervised representation learning technique to learn distributed representations of arbitrary sized graphs.
- Through our large-scale experiments on several benchmark and real-world datasets, we demonstrate that `graph2vec` could significantly outperform substructure representation learning algorithms and highly competitive to state-of-the-art graph kernels on graph classification and clustering tasks.
- We make an efficient implementation of `graph2vec` and the embeddings of all the datasets used in this work publicly available at [15].

The remainder of the paper is organized as follows: In §2 the problem of learning graph embeddings is formally defined. In §3, preliminaries on word and document representation learning approaches that `graph2vec` relies on are presented. The proposed method and its evaluation results and discussions are presented in §4 and §5, respectively. Conclusions are discussed in §6.

2. PROBLEM STATEMENT

Given a set of graphs $\mathbb{G} = \{G_1, G_2, ...\}$ and a positive integer δ (i.e., expected embedding size), we intend to learn δ-dimensional distributed representations for every graph $G_i \in \mathbb{G}$. The matrix of representations of all graphs is denoted as $\Phi \in \mathbb{R}^{|\mathbb{G}| \times \delta}$.

More specifically, let $G = (N, E, \lambda)$, represent a graph, where N is a set of nodes and $E \subseteq (N \times N)$ be a set of edges. Graph G is labeled if there exists a function λ such that $\lambda : N \rightarrow \ell$, which assigns a unique label from alphabet ℓ to every node $n \in N$. Otherwise, G is considered as unlabeled[2].

[1] For instance, Weisfeiler-Lehman kernel uses non-linear substructures for computing kernels across graphs and is demonstrated to outperform linear substructure kernels such as random walk kernel and shortest path kernel in many tasks [7, 10].

[2] Since our procedure requires node labels, in the case of unlabeled graphs, we follow the procedure mentioned in [10] and label nodes with their degree.

Additionally, the edges may also be labeled in which case we also have an edge labeling function, $\eta : E \rightarrow \mathfrak{e}$.

Given $G = (N, E, \lambda)$ and $sg = (N_{sg}, E_{sg}, \lambda_{sg})$, sg is a subgraph of G iff there exists an injective mapping $\mu : N_{sg} \rightarrow N$ such that $(n_1, n_2) \in E_{sg}$ iff $(\mu(n_1), \mu(n_2)) \in E$. In this work, by subgraph, we strictly refer to a specific class of subgraphs, namely, rooted subgraphs. In a given graph G, a rooted subgraph of degree d around node $n \in N$ encompasses all the nodes (and corresponding edges) that are reachable in d hops from n.

3. BACKGROUND: SKIPGRAM WORD & DOCUMENT EMBEDDING MODELS

Our goal is to learn the distributed representations of graphs by extending the recently proposed document embedding techniques in NLP for multi-relational data. Hence, in this section, we review the related background in language modeling, word and document embedding techniques.

3.1 Skipgram model for learning word embeddings

Modern neural embedding methods such as `word2vec` [2] use a simple and efficient feed forward neural network architecture called ”skipgram” to learn distributed representations of words. `word2vec` works based on the rationale that *the words appearing in similar contexts tend to have similar meanings and hence should have similar vector representations*. To learn a target word's representation, this model exploits the notion of context, where a context is defined as a fixed number of words surrounding the target word. To this end, given a sequence of words $\{w_1, w_2, ..., w_t, ..., w_T\}$, the target word w_t whose representation has to be learnt and the length of the context window c, the objective of skipgram model is to maximize the following log-likelihood:

$$\sum_{t=1}^{T} log\ Pr(w_{t-c}, ..., w_{t+c}|w_t) \tag{1}$$

where $w_{t-c}, ..., w_{t+c}$ are the context of the target word w_t. The probability $Pr(w_{t-c}, ..., w_{t+c})$ is computed as

$$\Pi_{-c \leq j \leq c, j \neq 0} Pr(w_{t+j}|w_t) \tag{2}$$

Here, the context words and the target word are assumed to be independent. Furthermore, $Pr(w_{t+j}|w_t)$ is defined as:

$$\frac{exp(\vec{w_t} \cdot \vec{w'}_{t+j})}{\sum_{w \in \mathcal{V}} exp(\vec{w_t} \cdot \vec{w})} \tag{3}$$

where $\vec{w}$ and $\vec{w}'$ are the input and output vectors of word w and $\mathcal{V}$ is the vocabulary of all the words.

3.2 Negative Sampling

The posterior probability in eq. (2) could be learnt in several ways. For instance, a naive approach would be to use a classifier like logistic regression. However, this is prohibitively expensive if the vocabulary $\mathcal{V}$ is very large.

Negative sampling is a simple yet efficient algorithm that helps to alleviate this problem and train the skipgram model. Negative sampling selects a small subset of words at random that are not in the target word's context and updates their embeddings in every iteration instead of considering all words in the vocabulary. Training this way ensures the following: *if a word w appears in the context of another word w', then the vector embedding of w is closer to that of w' compared to any other randomly chosen word from the vocabulary.*

Once skipgram training converges, semantically similar words are mapped to closer positions in the embedding space revealing that the learned word embeddings preserve semantics.

3.3 Neural document embedding models

Recently, `doc2vec`, a straight forward extension to `word2vec` from learning embeddings of words to those of word sequences was proposed by Le and Mikolov [3]. `doc2vec` uses an instance of the skipgram model called paragraph vector-distributed bag of words (PV-DBOW) (interchangeably referred as `doc2vec` skipgram) which is capable of learning representations of arbitrary length word sequences such as sentences, paragraphs and even whole large documents[3]. More specifically, given a set of documents $D = \{d_1, d_2, ...d_N\}$ and a sequence of words $c(d_i) = \{w_1, w_2, ..., w_{l_i}\}$ sampled from document $d_i \in D$, `doc2vec` skipgram learns a δ dimensional embeddings of the document $d_i \in D$ and each word w_j sampled from $c(d_i)$ i.e., $\vec{d_i} \in R^\delta$ and $\vec{w_j} \in R^\delta$, respectively. The model works by considering a word $w_j \in c(d_i)$ to be occurring in the context of document d_i and tries to maximize the following log likelihood:

$$\sum_{j=1}^{l_i} log\ Pr(w_j|d_i) \tag{4}$$

where, the probability $Pr(w_j|d)$ is defined as,

$$\frac{exp(\vec{d} \cdot \vec{w_j})}{\sum_{w \in \mathcal{V}} exp(\vec{d} \cdot \vec{w})} \tag{5}$$

where $\mathcal{V}$ is the vocabulary of all the words across all documents in D. Understandably, eq. (4) could be trained efficiently using negative sampling.

In `graph2vec`, we consider graphs analogical to documents that are composed of rooted subgraphs which, in turn, are analogical words from a special language and extend document embedding models to learn graph embeddings.

4. METHOD: LEARNING GRAPH REPRESENTATIONS

In this section we discuss the intuition (§4.1), overview (§4.2) and main components of our `graph2vec` algorithm (§4.3) in detail and explain how it learns embeddings of arbitrary sized graphs in an unsupervised manner.

4.1 Intuition

With the background on word and document embeddings presented in the previous section, an important intuition we extend in `graph2vec` is to view an entire graph as a document and the rooted subgraphs (that encompass a neighborhood of certain degree) around every node in the graph as

[3] To be precise, there are two versions of `doc2vec`, namely, PV-distributed memory (PV-DM) and PV-DBOW. PV-DM is not an instance of skipgram model. It learns document embeddings by combining and optimizing them with those of words that occur within a fixed length context window (similar to `word2vec`) in the corresponding documents. Evidently, PV-DM is not related to our proposed technique and hence we refrain from discussing it further.

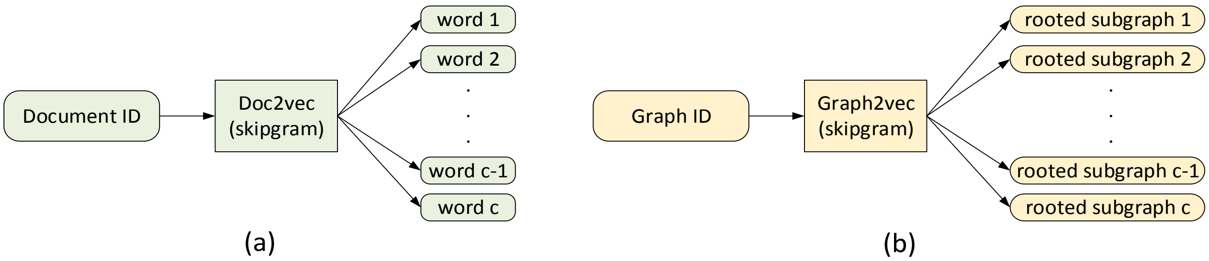

Figure 1: (a) `doc2vec`'s skipgram model - Given a document d, it samples c words from d and considers them as co-occuring in the same context (i.e., context of the document d) uses them to learn d's representation. (b) `graph2vec` - Given a graph G, it samples c rooted subgraphs around different nodes that occur in G and uses them analogous to `doc2vec`'s context words and thus learns G's representation.

words that compose the document. In other words, different subgraphs compose graphs in a similar way that different words compose sentences/documents when used together.

At this juncture, it is important to note that other substructures such as nodes, walks and paths could also be considered as atomic entities that compose a graph, instead of rooted subgraphs. However, there are two reasons that make rooted subgraphs more amenable for learning graph embeddings:

1. Higher order substructure. Compared to simpler lower order substructures such as nodes, rooted subgraphs encompass higher order neighborhoods which offers a richer representation of composition of the graphs. Hence, the embeddings learnt through sampling such higher order substructures would reflect the compositions of the graphs better.

2. Non-linear substructure. Compared to linear substructures such as walks and paths, rooted subgraphs capture the inherent non-linearity in the graphs better. This fact is evident while considering the graph kernels, as well. For instance, Weisfeiler-Lehman (WL) kernel which are based on non-linear substructures offer significantly better performance on many tasks than the linear substructure based kernels such as random walk and shortest path kernels [7, 10].

Through establishing the above mentioned analogy of documents and words to graphs and rooted subgraphs, respectively, one can utilize document embedding models to learn graph embeddings. The main expectation here is that structurally similar graphs will be close to each other in the embedding space. In this sense, similar to the Deep Graph Kernels [7], `graph2vec`'s embeddings provide means to arrive a data-driven graph kernel.

4.2 Algorithm overview

Similar to the document convention, the only required input is a corpus of graphs for `graph2vec` to learn their representations. Given a dataset of graphs, `graph2vec` considers the set of all rooted subgraphs (i.e., neighbourhoods) around every node (up to a certain degree) as its vocabulary. Subsequently, following the `doc2vec` skipgram training process, we learn the representations of each graph in the dataset.

To train the skipgram model in the above mentioned fashion we need to extract rooted subgraphs and assign a unique label for all the rooted subgraphs in the vocabulary. To this end, we deploy the WL relabeling strategy (which is also used by the WL kernel).

4.3 `graph2vec`: Algorithm

Algorithm 1: GRAPH2VEC $(\mathbb{G}, D, \delta, \mathfrak{e}, \alpha)$

```
input : $\mathbb{G} = \{G_1, G_2, ..., G_n\}$: Set of graphs such that each
        graph $G_i = (N_i, E_i, \lambda_i)$ for which embeddings have to
        be learnt
        $D$: Maximum degree of rooted subgraphs to be
        considered for learning embeddings. This will produce
        a vocabulary of subgraphs, $SG_{vocab} = \{sg_1, sg_2, ...\}$
        from all the graphs in $\mathbb{G}$
        $\delta$: number of dimensions (embedding size)
        $\mathfrak{e}$: number of epochs
        $\alpha$: Learning rate
output: Matrix of vector representations of graphs $\Phi \in \mathbb{R}^{|\mathbb{G}| \times \delta}$
1  begin
2      Initialization: Sample $\Phi$ from $R^{|\mathbb{G}| \times \delta}$
3      for $e = 1$ to $\mathfrak{e}$ do
4          $\mathfrak{G} = $ SHUFFLE $(\mathbb{G})$
5          for each $G_i \in \mathfrak{G}$ do
6              for each $n \in N_i$ do
7                  for $d = 0$ to $D$ do
8                      $sg_n^{(d)} :=$ GETWLSUBGRAPH$(n, G_i, d)$
9                      $J(\Phi) = -\log \Pr\ (sg_n^{(d)} | \Phi(G))$
10                     $\Phi = \Phi - \alpha \frac{\partial J}{\partial \Phi}$
11     return $\Phi$
```

The algorithm consists of two main components; first, a procedure to generate rooted subgraphs around every node in a given graph (§4.3.1) and second, the procedure to learn embeddings of the given graphs (§4.3.2).

As presented in Algorithm 1 we intend to learn δ dimensional embeddings of all the graphs in dataset $\mathbb{G}$ in $\mathfrak{e}$ epochs. We begin by randomly initializing the embeddings for all graphs in the dataset (line 2). Subsequently, we proceed with extracting rooted subgraphs around every node in each of the graphs (line 8) and iteratively learn (i.e., refine) the corresponding graph's embedding in several epochs (lines 3 to 10). These steps represent the core of our approach and are explained in detail in the two following subsections.

4.3.1 Extracting Rooted Subgraphs

To facilitate learning graph embeddings, a rooted subgraph $sg_n^{(d)}$ around every node n of graph G_i is extracted (line 8). This is a fundamentally important task in our approach. To extract these subgraphs, we follow the well-known WL relabeling process [10] which lays the basis for the WL kernel [7, 10]. The subgraph extraction process is explained separately in Algorithm 2. The algorithm takes the root node n, graph G from which the subgraph has to be extracted and degree of the intended subgraph d as inputs

Algorithm 2: GETWLSUBGRAPH (n, G, d)

```
input  : $n$: Node which acts as the root of the subgraph
         $G = (N, E, \lambda)$: Graph from which subgraph has to be extracted
         $d$: Degree of neighbours to be considered for extracting subgraph
output: $sg_n^{(d)}$: Rooted subgraph of degree $d$ around node $n$
1 begin
2     $sg_n^{(d)} = \{\}$
      if $d = 0$ then
3         $sg_n^{(d)} := \lambda(n)$
4     else
5         $\mathcal{N}_n := \{n' \mid (n, n') \in E\}$
6         $M_n^{(d)} := \{\text{GETWLSUBGRAPH}(n', G, d-1) \mid n' \in \mathcal{N}_n\}$
7         $sg_n^{(d)} := sg_n^{(d)} \cup$ GETWLSUBGRAPH
              $(n, G, d-1) \oplus sort(M_n^{(d)})$
8     return $sg_n^{(d)}$
```

and returns the intended subgraph $sg_n^{(d)}$. When $d = 0$, no subgraph needs to be extracted and hence the label of node n is returned (line 3). For cases where $d > 0$, we get all the (breadth-first) neighbours of n in $\mathcal{N}_n$ (line 5). Then for each neighbouring node, n', we get its degree $d-1$ subgraph and save the same in list $M_n^{(d)}$ (line 6). Finally, we get the degree $d-1$ subgraph around the root node n and concatenate the same with sorted list $M_n^{(d)}$ to obtain the intended subgraph $sg_n^{(d)}$ (line 7).

4.3.2 Skipgram with Negative Sampling

Given that $sg_n^{(d)} \in SG_{vocab}$ and $|SG_{vocab}|$ is very large, calculating $Pr(sg_n^{(d)}|\Phi(G))$ in line 9 of Algorithm 1 is prohibitively expensive. Hence we follow the negative sampling strategy (introduced in §3.2) to calculate this posterior probability.

In our negative sampling phase, for every training cycle of Algorithm 1, given a graph $G_i \in \mathbb{G}$ and a set of rooted subgraphs in its context, $c(G_i) = c = \{sg_1, sg_2, ...\}$, we select a set of fixed number of randomly chosen subgraphs as negative samples $c' = \{sgn_1, sgn_2, ...sgn_k\}$ such that $c' \subset SG_{vocab}, k << |SG_{vocab}|$ and $c \cap c' = \{\}$. Intuitively, negative samples (c') is a set of k subgraphs, each of which is not present in the graph whose embedding has to be learnt (G_i), but in the vocabulary of subgraphs. The number of negative samples (k) is a hyper-parameter that could be empirically tuned. For efficient training, for every graph $G_i \in \mathbb{G}$, first, the target-context pairs $(G_i,\ c)$ are trained and the embeddings of the corresponding subgraphs are updated. Subsequently, we update only the embeddings of the negative samples c', instead of the whole vocabulary.

Given a pair of graphs G_i and G_j, this training makes their embeddings $\Phi(G_i)$ and $\Phi(G_j)$ closer if they are composed of similar rooted subgraphs (i.e., $c(G_i)$ is similar to $c(G_j)$) and at the same time distances them from the embeddings of all the graphs which are not composed of similar subgraphs.

4.3.3 Optimization

Stochastic gradient descent (SGD) optimizer is used to optimize the parameters in line 9 and 10 of Algorithm 1. The derivatives are estimated using the back-propagation algorithm. The learning rate α is empirically tuned.

4.4 Use cases

Once the embeddings of graphs are computed using `graph2vec`, they could be used for a variety of downstream graph analytics tasks. The prominent ones are reviewed below.

Graph Classification. Given $\mathbb{G}$, a set of graphs and Y, the set of corresponding class labels, graph classification is the task where we learn a model $\mathcal{H}$ such that $\mathcal{H} : \mathbb{G} \rightarrow Y$. To this end, one could obtain the embeddings of all the graphs in $\mathbb{G}$ and feed them to general purpose classifiers such as RFs, Nueral Networks and SVMs to perform classification. At this juncture, it is important to note that other graph embedding procedure such as graph kernels and substructure embeddings do not offer this flexibility. More specifically, in the case of such methods, the kernel matrices computed using them[4] could be used only in conjunction with kernel classifiers (e.g., SVMs) and general purpose classifiers could not be used.

Graph Clustering. Given $\mathbb{G}$, in graph clustering, the goal is to group similar graphs together. `graph2vec`'s embeddings could be used along with general purpose clustering algorithms such as K-means and relational clustering algorithms such as Affinity Propagation (AP) [14] to achieve this. Again, due to the aforementioned limitations of graph kernels and substructure embeddings, they could be used only with relational clustering algorithms.

5. EVALUATION

We evaluate `graph2vec`'s accuracy and efficiency both in graph classification and clustering tasks. Besides experimenting with benchmark datasets, we also evaluate our approach on two real-world graph analytics tasks from the field of program analysis, namely, malware detection and malware familial clustering on large malware datasets.

Research Questions. Specifically, we intend to address the following research questions: (1) How does `graph2vec` compare to state-of-the-art substructure representation learning approaches and graph kernels for graph classification tasks in terms of accuracy and efficiency on benchmark datasets, (2) How does `graph2vec` compare to the aforementioned state-of-the-art approaches on a real-world graph classification task, namely, malware detection detection, and (3) How does `graph2vec` compare to the aforementioned state-of-the-art approaches on a real-world graph clustering task, namely, malware familial clustering.

Comparative Analysis. The proposed approach is compared with two representation learning techniques, namely, `node2vec` [4] and `sub2vec` [5] and two graph kernel techniques, namely, WL kernel [10] and Deep WL kernel [7]. `node2vec` is a neural embedding framework which learns feature representation of individual nodes in graphs. In our experiments, to obtain embeddings of entire graphs using `node2vec`, we average those of all the nodes in the graph. `sub2vec` [5] is a framework that learns representations of any arbitrary subgraphs. Therefore, obtaining representation of whole graphs using `sub2vec` is a straightforward procedure. WL kernel [10] is handcrafted feature based kernel that decomposes graphs into rooted subgraphs and computes the kernel values based on them. Besides kernel values, it also yields explicit vector representations of graphs. Deep WL

[4] see [7] for the explanations on obtaining kernel matrix with substructure embedding approaches

Table 1: Benchmark dataset statistics

Dataset	# samples	# nodes (avg.)	# distinct node labels
MUTAG	188	17.9	7
PTC	344	25.5	19
PROTEINS	1113	39.1	3
NCI1	4110	29.8	37
NCI109	4127	29.6	38

kernel [7] is a representation learning variant of WL kernel which learns embeddings of rooted subgraphs such that similar subgraphs have similar embeddings. Thus, the kernel values obtained using subgraph embeddings would be unaffected by the limitations of handcrafted features such as diagonal dominance.

Evaluation Setup. All the experiments were conducted on a server with 36 CPU cores (Intel E5-2699 2.30GHz processor) and 200 GB RAM running Ubuntu 14.04.

5.1 Graph Classification with Benchmark Datasets

Datasets. Five benchmark graph classification datasets namely MUTAG, PTC, PROTEINS, NCI1 and NCI109 are used in this experiment. These datasets belong to chemo- and bio-informatics domains and the specifications of the datasets used are given in Table 1. MUTAG is a data set of 188 chemical compounds labeled according to whether or not they have a mutagenic effect on a specific bacteria. PTC dataset comprises of 344 compounds and their classes indicate carcinogenicity on rats. PROTEINS is a collection of graphs whose nodes represent secondary structure elements and edges indicate neighborhood in the amino-acid sequence or in 3D space. NCI1 and NCI109 datasets consist of 4,110 and 4,127 graphs respectively, representing two balanced subsets of datasets of chemical compounds screened for activity against non-small cell lung cancer and ovarian cancer cell lines, respectively.

Experiment & Configurations. In this experiment, for each of the datasets, we train a SVM classifier with 90% of the samples chosen at random and evaluate its performance on the test set of remaining 10% samples. The hyper-parameters of the classifiers are tuned based on 5-fold cross validation on the training set. For all the representation learning methods, we used a common embedding dimensions of 1024, which was arrived empirically[5].

Evaluation Metrics. The experiment is repeated 5 times and the average accuracy is used to determine the effectiveness of classification. Efficiency is determined in terms of time consumed for building graph embeddings (*aka* pre-training duration). The training and testing durations are not reported as they are not directly related to the proposed method.

5.1.1 Results and Discussion

Accuracy. The results obtained by the graph2vec on benchmark datasets are summarized in Table 2. From the results, it is evident that the proposed approach outperforms other representation learning and kernel based techniques on 3 datasets (MUTAG, PTC and PROTEINS) and has comparable accuracy on the remaining 2 datasets (NCI1 and NCI109). The following inferences are made from the table.

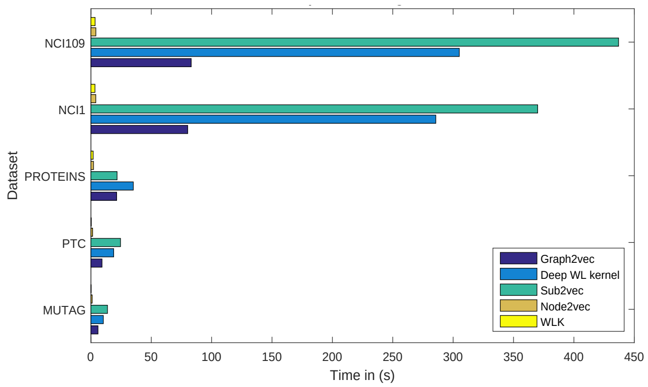

Figure 2: Pre-training durations of graph embedding techniques

- node2vec being a lower order substructure embedding technique, it could only model local similarity within a confined neighborhood and fails to learn global structural similarities that helps to classify similar graphs together. This is especially evident from the results on larger datasets, PROTEINS, NCI1 and NCI109 where node2vec's accuracy is just above 50% (i.e., only marginally better than random classification). In general, from these results, one could conclude that while the substructure embeddings techniques excel in substructure based analytics tasks (see [4] for node2vec's node classification and link prediction performances), extending them directly for tasks involving whole graphs yields sub-par accuracies.

- sub2vec performs predominantly poorly across all datasets. This is mainly because of the fact that its strategy to sample graph substructures and learn their embeddings is particularly ill-suited for obtaining embedding of large graphs. That is, sub2vec samples only one random walk (of fixed length) from the given graph and subsequently learns its representations using fixed length linear context skipgrams (with several iterations) over the sampled walk. This prevents sub2vec from learning meaningful representations of an entire graph, as sampling only random walk may not be enough to cover all the neighborhoods in the graph. This ultimately prevents the method from appropriately modeling the structural similarities across graphs which reflects in its poor performance. Also, this inference is reinforced by the fact that sub2vec accuracies decrease with the increase in the size of the graphs (see the difference in accuracies for MUTAG and NCI109 datasets).

- WL kernel, being a technique particularly designed to cater tasks such as graph classification, consistently provides good results on all datasets. Deep WL Kernel performs better than WL kernel on all datasets, as it addresses the limitations of the latter kernel's handcrafted features and achieves better generalization.

- Finally, graph2vec's structure-preserving, data-driven embedding which appropriately models both local and global similarities among graphs, consistently yields good results on all datasets. In particular, it outperforms all the state-of-the-art methods in MUTAG, PTC and PROTEINS dataset and obtains slightly lesser accuracies on NCI1 and NCI109 datasets than the kernels.

[5] Embedding dimensions of {16, 32, 64,..., 4096} were experimented with and 1024 was found produce best results predominantly with reasonable efficiency, across all datasets

Table 2: Average Accuracy (± std dev.) for `graph2vec` and state-of-the-art graph kernels on benchmark graph classification datasets

Dataset	MUTAG	PTC	PROTEINS	NCI1	NCI109
`node2vec` [4]	72.63 ± 10.20	58.85 ± 8.00	57.49 ± 3.57	54.89 ± 1.61	52.68 ± 1.56
`sub2vec` [5]	61.05 ± 15.79	59.99 ± 6.38	53.03 ± 5.55	52.84 ± 1.47	50.67 ± 1.50
WL kernel [10]	80.63 ± 3.07	56.91 ± 2.79	72.92 ± 0.56	80.01 ± 0.50	80.12 ± 0.34
Deep WL kernel [7]	82.95 ± 1.96	59.04 ± 1.09	**73.30** ± 0.82	**80.31** ± 0.46	**80.32** ± 0.33
`graph2vec`	**83.15** ± 9.25	**60.17** ± 6.86	**73.30** ± 2.05	73.22 ± 1.81	74.26 ± 1.47

Table 3: Large real-world datasets used in graph classifications and clustering tasks

Dataset	source	# samples	# classes	# nodes (avg.)	# edges (avg.)	# distinct node labels
Classification	[1, 12]	10,560	2	2637.12	927.31	4271
Clustering	[13]	24,650	71	1071.33	544.83	3828

Table 4: Malware Detection - Results

Method	Accuracy (avg. ± std.)
`node2vec` [4]	81.25 ± (1.04)
`sub2vec` [5]	76.83 ± (2.83)
WL kernel [10]	97.12 ± (0.44)
Deep WL kernel [7]	98.16 ± (0.20)
`graph2vec`	**99.03 ± (0.17)**

Efficiency. A pre-training phase to compute vectors of substructures and graphs is required for all the aforementioned methods except WL kernel. On the other hand, WL kernel requires a phase to extract rooted subgraph features and build handcrafted embeddings. In the case of former approaches, pre-training is the crucial step that helps in capturing latent substructure similarities in graphs and thus aids them to outperform handcrafted feature techniques. Therefore, it is important to study the cost of pre-training. The results of pre-training/feature extraction durations for all the methods under study are shown in Figure 2.

Understandably, WL kernel is the most scalable method for obtaining graph embeddings as it does not involve learning representations. `node2vec` learns embeddings of lower order entities (i.e., nodes) through confined explorations of neighborhoods around them and hence takes very less time for pretraining. `sub2vec` learns graph embeddings by sampling linear substructures and running several iterations of skipgram algorithm over them. This results in significantly high pretraining durations. DeepWL kernel learns rooted subgraph embeddings using skipgram. It takes much lesser duration than `sub2vec` as the latter's length of sampled random walk is much longer than the number of samples rooted subgraphs in the former. Finally, our approach, which learns embeddings of higher order structures remains less scalable than `node2vec`, but much more scalable than Deep WL kernel and `sub2vec`. This is due to the fact that, our approach runs skipgram training only a limited number times (which is equal to the number of rooted subgraphs sampled form the given graph), while the other two approaches run it several times over a fixed length linear context window.

The efficiency results in our experiments with real-world datasets discussed in subsequent subsections follow the same pattern as the one discussed above. Hence, we refrain from discussing efficiency results here after.

5.2 Graph Classification with Real-world Dataset

The performances of graph embedding approaches on large real-world datasets may be different from the benchmark ones as they are more complex. Furthermore, benchmark datasets used in §5.1 are too small for the data-driven embedding approaches to reap considerable leverage by exploiting the volume of data over the handcrafted approaches. Therefore, it is highly essential to evaluate the performance of the proposed method on large real-world datasets to showcase its true potentials.

Experiment & Configurations. To this end, we consider a large-scale Android malware detection problem where we are given a dataset of API Dependency Graphs (ADGs) of malicious and benign Android apps, and the task is to represent each of them as vectors and train ML classifiers to identify malicious ones. The datasets statistics are presented in Table 3. Evidently, these ADGs are much larger than the benchmark graphs. The dataset comprises of 10,560 ADGs, each of which contain more than 2,600 nodes (i.e., instructions), 920 edges (i.e., control flows among instructions) and 4200 unique node labels (i.e., APIs invoked in instructions) on average. The training set comprises of 70% of samples chosen at random and the remaining 30% samples are used as test set to evaluate the models. The experiment is repeated 5 times and the results are averaged.

Evaluation Metrics. The same evaluation metrics that are used in experiments with benchmark datasets (see §5.1) are used here as well.

5.2.1 Results & Discussion

The malware detection results of the proposed and compared state-of-the-art approaches are presented in Table 4. From the results obtained, the following inference is drawn:

- Averaging `node2vec` embeddings and using `sub2vec` to obtain graph representations perform poorly in this experiment as well. In particular, the proposed approach outperforms these two techniques by more than 17% and 22%, respectively.
- Both WL and Deep WL kernels perform significantly better than the two substructure representation learning approaches. However, `graph2vec` still outperforms these techniques by 1.91% and 0.87%, respectively.
- Evidently, being data-driven approaches, both `graph2vec` and Deep WL kernel exhibit excellent performance on this large-scale dataset. Especially in this experiment, the range in which they outperform other techniques under comparison is more pronounced than the experiments with benchmark datasets. Again, the two representation learning approaches, `node2vec` and `sub2vec` perform worse as they are ill-suited for learning embeddings of entire graphs.

5.3 Graph Clustering

The goal of this experiment is to demonstrate the efficacy of `graph2vec`'s embedding in a downstream analytics task where we do not have class labels for graphs. This task would be most appropriate for evaluating and comparing the methods that do not leverage on any task-specific information in the process of learning representations.

Experiment & Configurations. In this experiment, we are given with ADGs of malware samples and the name

Table 5: Malware Clustering - Results

Method	ARI (as %)
node2vec [4]	16.39
sub2vec [5]	14.55
WL kernel [10]	48.93
Deep WL kernel [7]	50.41
graph2vec	**56.28**

of families[6] to which they belong and the task is to group samples belonging to the same family into the same cluster. To this end, we consider the AMD dataset [13] which comprises of more than 24,000 Android malware apps belonging to 71 families. The statistics of this dataset is presented in Table 3.

From this dataset, only the large families that have more than 100 corresponding malware samples are considered for clustering as this helps to mitigate imbalance in the cluster sizes. The embeddings and kernels of ADGs belonging to these families are built and a relational clustering algorithm, namely, AP [14] is used to cluster them.

Evaluation Metric. In order to quantitatively measure malware familial clustering accuracy, a standard clustering evaluation metric, namely, Adjusted Rand Index (ARI) is used. The ARI values lie in the range [-1, 1]. For the ease of understanding, we report the ARI as a percentage value. A higher ARI means a higher correspondence to the ground-truth data.

5.3.1 Results and Discussion

The results of malware clustering using graph2vec and other state-of-the-art methods are presented in Table 5. From the table, the following inferences are drawn:

- At the outset, we observe all the method obtain lesser ARIs in this experiment, as the malware clustering task is inherently more complex than two classification tasks considered previously.
- Similar to the classification tasks, both node2vec and sub2vec perform poorer than the kernels and graph2vec. This reinforces the inference that adopting node2vec and sub2vec for graph embedding will yield subpar results.
- Both WL and Deep WL kernel perform much better than the two aforementioned embedding approaches. However, different from the classification tasks, in this task, the former methods outperform the latter methods by more than 2 folds.
- In this experiment, graph2vec outperforms all the other compared approaches highly significantly. In particular, it outperforms the substructure embedding techniques by more than 39% and the kernels by more than 5%. This reinforces the findings inferred from the classification experiments.

Summary. Summarizing the inferences from all the three experiments, one could see: (1) trivially extending node and subgraph representation learning approaches to build graph embeddings yield sub-par results, and (2) learning graph embedding from data leads to more accurate results than building the same using handcrafted features. Since graph2vec is appropriately designed, it achieves excellent accuracies in graph analytics tasks with reasonably good efficiency.

[6] Samples belonging to same families perform similar malicious activities

6. CONCLUSIONS

In this paper, we presented graph2vec, an unsupervised representation learning technique to learn embedding of graphs of arbitrary sizes. Through our large-scale experiments involving benchmark graph classification datasets, we demonstrate that graph embeddings learnt by our approach outperform substructure embedding approaches significantly and are comparable to graph kernels. Since graph2vec is a data-driven representation learning approach, its true potentials are revealed when trained on large volumes of graphs. To this end, when evaluated on two real-world applications involving large graph datasets, graph2vec outperforms state-of-the-art graph kernels without compromising efficiency of the overall performance. We make all the code and data used within this work available at: [15].

7. ACKNOWLEDGMENTS

We thank the authors of [4], [7] and [10] for making the source code of their approaches publicly available. We thank the authors of [5] for sharing their approach's source code with us.

8. REFERENCES

[1] Google Play Market. https://play.google.com/store (accessed May 2017).

[2] Mikolov, Tomas, et al. "Efficient estimation of word representations in vector space." arXiv preprint arXiv:1301.3781 (2013).

[3] Le, Quoc, and Tomas Mikolov. "Distributed representations of sentences and documents." Proceedings of the 31st International Conference on Machine Learning (ICML-14). 2014.

[4] Grover, Aditya, and Jure Leskovec. "node2vec: Scalable feature learning for networks." Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, 2016.

[5] Adhikari, Bijaya, et al. "Distributed Representation of Subgraphs." arXiv preprint arXiv:1702.06921 (2017).

[6] Narayanan, Annamalai, et al. "subgraph2vec: Learning distributed representations of rooted sub-graphs from large graphs." International Workshop on Mining and Learning with Graphs. (2016).

[7] Yanardag, Pinar, and S. V. N. Vishwanathan. "Deep graph kernels." Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, 2015.

[8] Goyal, Palash, and Emilio Ferrara. "Graph Embedding Techniques, Applications, and Performance: A Survey." arXiv preprint arXiv:1705.02801 (2017).

[9] Niepert, Mathias, Mohamed Ahmed, and Konstantin Kutzkov. "Learning convolutional neural networks for graphs." Proceedings of the 33rd annual international conference on machine learning. ACM. 2016.

[10] Shervashidze, Nino, et al. "Weisfeiler-lehman graph kernels." Journal of Machine Learning Research 12.Sep (2011): 2539-2561.

[11] Wang, Daixin, et al. "Structural deep network embedding." Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, 2016.

[12] Arp, Daniel, et al. "DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket." NDSS. 2014.

[13] Wei, Fengguo, et al. "Deep Ground Truth Analysis of Current Android Malware." Proceedings of the 14th Conference on Detection of Intrusions and Malware & Vulnerability Assessment. 2017.

[14] Frey, Brendan J., and Delbert Dueck. "Clustering by passing messages between data points." science 315.5814 (2007): 972-976.

[15] Graph2vec website: https://sites.google.com/view/graph2vec