subgraph2vec: Learning Distributed Representations of Rooted Sub-graphs from Large Graphs

Annamalai Narayanan[†], Mahinthan Chandramohan[†], Lihui Chen[†], Yang Liu[†] and Santhoshkumar Saminathan[§]
[†]Nanyang Technological University, Singapore
[§]BigCommerce, California, USA
annamala002@e.ntu.edu.sg, {mahinthan,elhchen,yangliu}@ntu.edu.sg, santhosh.kumar@yahoo.com

ABSTRACT

In this paper, we present subgraph2vec, a novel approach for learning latent representations of rooted subgraphs from large graphs inspired by recent advancements in Deep Learning and Graph Kernels. These latent representations encode semantic substructure dependencies in a continuous vector space, which is easily exploited by statistical models for tasks such as graph classification, clustering, link prediction and community detection. subgraph2vec leverages on local information obtained from neighbourhoods of nodes to *learn* their latent representations in an *unsupervised* fashion. We demonstrate that subgraph vectors learnt by our approach could be used in conjunction with classifiers such as CNNs, SVMs and relational data clustering algorithms to achieve significantly superior accuracies. Also, we show that the subgraph vectors could be used for building a deep learning variant of Weisfeiler-Lehman graph kernel. Our experiments on several benchmark and large-scale real-world datasets reveal that subgraph2vec achieves significant improvements in accuracies over existing graph kernels on both supervised and unsupervised learning tasks. Specifically, on two real-world program analysis tasks, namely, code clone and malware detection, subgraph2vec outperforms state-of-the-art kernels by more than 17% and 4%, respectively.

Keywords

Graph Kernels, Deep Learning, Representation Learning

1. INTRODUCTION

Graphs offer a rich, generic and natural way for representing structured data. In domains such as computational biology, chemoinformatics, social network analysis and program analysis, we are often interested in computing similarities between graphs to cater domain-specific applications such as protein function prediction, drug toxicity prediction and malware detection.

Graph Kernels. Graph Kernels are one of the popular and widely adopted approaches to measure similarities among graphs [3, 4, 6, 7, 14]. A Graph kernel measures the similarity between a pair of graphs by recursively decomposing them into *atomic* substructures (e.g., walk [3], shortest paths [4], graphlets [7] etc.) and defining a similarity function over the substructures (e.g., number of common substructures across both graphs). This makes the kernel function correspond to an inner product over substructures in reproducing kernel Hilbert space (RKHS). Formally, for a given graph G, let $\Phi(G)$ denote a vector which contains counts of atomic substructures, and $\langle \cdot, \cdot \rangle_H$ denote a dot product in a RKHS H. Then, the kernel between two graphs G and G' is given by

$$K(G, G') = \langle \Phi(G), \Phi(G') \rangle_H \quad (1)$$

From an application standpoint, the kernel matrix $\mathfrak{K}$ that represents the pairwise similarity of graphs in the dataset (calculated using eq. (1)) could be used in conjunction with kernel classifiers (e.g., Support Vector Machine (SVM)) and relational data clustering algorithms to perform graph classification and clustering tasks, respectively.

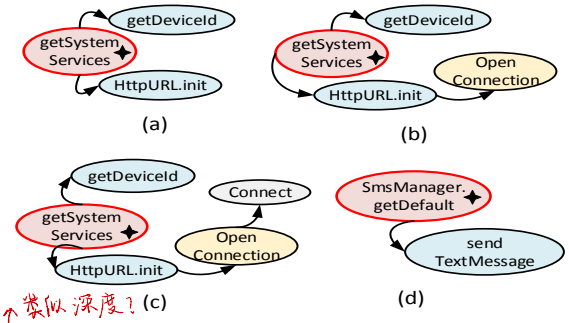

Figure 1: Dependency schema of a set of rooted subgraphs of degree 1 ((a), (d)), 2 ((b)) and 3 ((c)) in an Android malware's API dependency graph. The root nodes are marked with a star. Graph (b) can be derived from (a) by adding a node and an edge. Graph (c) can be derived from (b) in a similar fashion. Graph (d) is highly dissimilar from all the other graphs and is not readily derivable from any of them.

1.1 Limitations of Existing Graph Kernels

However, as noted in [7, 14], the representation in eq. (1) does not take two important observations into account.

- **(L1) Substructure Similarity.** Substructures that are used to compute the kernel matrix are not independent. To illustrate this, lets consider the Weisfeiler-Lehman (WL)

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

kernel [6] which decomposes graphs into rooted subgraphs[1]. These subgraphs encompass the neighbourhood of certain degree around the root node. Understandably, these subgraphs exhibit strong relationships among them. That is, a subgraph with second degree neighbours of the root node could be arrived at by adding a few nodes and edges to its first degree counterpart. We explain this with an example presented in Fig. 1. The figure illustrates API-dependency subgraphs from a well-known Android malware called DroidKungFu (DKF) [18]. These subgraph portions of DKF involves in leaking users' private information (i.e., IMEI number) over the internet and sending premium-rates SMS without her consent. Sub-figures (a), (b) and (c) represent subgraphs of degree 1, 2 and 3 around the root node `getSystemServices`, respectively. Evidently, these subgraphs exhibit high similarity among one another. For instance, subgraph (c) could be derived from subgraph (b) by adding a node and an edge, which in turn could be derived from subgraph (a) in a similar fashion. However, the WL kernel, by design ignores these subgraph similarities and considers each of the subgraphs as individual features. Other kernels such as random walk and shortest path kernels also make similar assumptions on their respective substructures' similarities.

- **(L2) Diagonal Dominance.** Since graph kernels regard these substructures as separate features, the dimensionality of the feature space often grows exponentially with the number of substructures. Consequently, only a few substructures will be common across graphs. This leads to diagonal dominance, that is, a given graph is similar to itself but not to any other graph in the dataset. This leads to poor classification/clustering accuracy.

1.2 Existing Solution: Deep Graph Kernels

To alleviate these problems Yanardag and Vishwanathan [7], recently proposed an alternative kernel formulation termed as *Deep Graph Kernel* (DGK). Unlike eq. (1), DGK captures the similarities among the substructures with the following formulation:

$$\mathcal{K}(G, G') = \Phi(G)^T \mathcal{M} \Phi(G') \tag{2}$$

where $\mathcal{M}$ represents a $|\mathcal{V}| \times |\mathcal{V}|$ positive semi-definite matrix that encodes the relationship between substructures and $\mathcal{V}$ represents the vocabulary of substructures obtained from the training data. Therefore, one can design a $\mathcal{M}$ matrix that respects the similarity of the substructure space.

Learning representation of substructures. In DGK [7], the authors used representation learning (deep learning) techniques inspired by the work of Mikolov et al. [15] to learn vector representations (*aka* embeddings) of substructures. Subsequently, these substructure embeddings were used to compute $\mathcal{M}$ and the same is used in eq (2) to arrive at the deep learning variants of several well-known kernels such as WL, graphlet and shortest path kernels.

Context. In order to facilitate unsupervised representation learning on graph substructures, the authors of [7] defined a notion of *context* among these substructures. Substructures that co-occur in the same context tend to have high similarity. For instance, in the case of rooted subgraphs, all the subgraphs that encompass same degree of neighbourhood around the root node are considered as co-occurring in the same context (e.g., all degree-1 subgraphs are considered to be in the same context). Subsequently, embedding learning task's objective is designed to make the embeddings of substructures that occur in the same context similar to one another. Thus defining the correct context is of paramount importance to build high quality embeddings.

Deep WL Kernel. Through their experiments the authors demonstrated that the deep learning variant of WL kernel constructed using the above-said procedure achieved state-of-the-art performances on several datasets. However, we observe that, in their approach to learn subgraph embeddings, the authors make three novice assumptions that lead to three critical problems:

- (**A1**) *Only rooted subgraphs of same degree are considered as co-occurring in the same context.* That is, if $D_G^{(d)} = \{sg_1^{(d)}, sg_2^{(d)}, ...\}$ is a multi-set of all degree d subgraphs in graph G, [7] assumes that any two subgraphs $sg_i^{(d)}, sg_j^{(d)} \in D_G^{(d)}$ co-occur in the same context irrespective of the length (or number) of path(s) connecting them or whether they share the same nodes/edges. For instance, in the case of Android malware subgraphs in Fig. 1, [7] assumes that only subgraphs (a) and (d) are in the same context and are possibly similar as they both are degree-1 subgraphs. However in reality, they share nothing in common and are highly dissimilar. This assumption makes subgraphs that do not co-occur in the same graph neighbourhood to be in the same context and thus similar (problem 1).
- (**A2**) *Any two rooted subgraphs of different degrees never co-occur in the same context.* That is, two subgraphs $sg_i^{(d)} \in D_G^{(d)}$ and $sg_j^{(d')} \in D_G^{(d')}$ (where $d \neq d'$) never co-occur in the same context irrespective of the length (or number) of path(s) connecting them or whether they share the same nodes/edges. For instance, in Fig. 1, subgraphs (a), (b) and (c) are considered not co-occurring in the same context as they belong to different degree neighbourhood around the root node. Hence, [7] incorrectly biases them to be dissimilar. This assumption makes subgraphs that co-occur in the same neighbourhood not to be in the same context and thus dissimilar (problem 2).
- (**A3**) *Every subgraph ($sg_r^{(d)}$) in any given graph has exactly same number of subgraphs in its context.* This assumption clearly violates the topological neighbourhood structure in graphs (problem 3).

Through our thorough analysis and experiments we observe that these assumptions led [7] to building relatively low quality subgraph embeddings. Consequently, this reduces the classification and clustering accuracies when [7]'s deep WL kernel is deployed. This motivates us to address these limitations and build better subgraph embeddings, in order to achieve higher accuracy.

1.3 Our Approach

In order to learn accurate subgraph embeddings, we address each of the three problems introduced in the previous subsection. We make two main contributions through our subgraph2vec framework to solve these problems:

- We extend the WL relabeling strategy [6] (used to relabel the nodes in a graph encompassing its breadth-first neigh-

[1]The WL kernel models the subgraph around a root node as a tree (i.e., without cycles) and hence is referred as WL subtree kernel. However since the tree represents a rooted subgraph, we refer to the rooted subgraph as the substructure being modeled in WL kernel, in this work.

bourhood) to define a proper context for a given subgraph. For a given subgraph $sg_r^{(d)}$ in G with root r, subgraph2vec considers all the rooted subgraphs (up to a certain degree) of neighbours of r as the context of $sg_r^{(d)}$. This solves problems 1 and 2.

- However this context formation procedure yields radial contexts of different sizes for different subgraphs. This renders the existing representation learning models such as the skipgram model [15] (which captures fixed-length linear contexts) unusable in a straight-forward manner to learn the representations of subgraphs using its context, thus formed. To address this we propose a modification to the skipgram model enabling it to capture varying length radial contexts. This solves problem 3.

Experiments. We determine subgraph2vec's accuracy and efficiency in both supervised and unsupervised learning tasks with several benchmark and large-scale real-world datasets. Also, we perform comparative analysis against several state-of-the-art graph kernels. Our experiments reveal that subgraph2vec achieves significant improvements in classification/clustering accuracy over existing kernels. Specifically, on two real-world program analysis tasks, namely, code clone and malware detection, subgraph2vec outperforms state-of-the-art kernels by more than 17% and 4%, respectively.

Contributions. We make the following contributions:

- We propose subgraph2vec, an unsupervised representation learning technique to learn latent representations of rooted subgraphs present in large graphs (§5).
- We develop a modified version of the skipgram language model [15] which is capable of modeling varying length radial contexts (rather than fixed-length linear contexts) around target subgraphs (§5.2).
- We discuss how subgraph2vec's representation learning technique would help to build the deep learning variant of WL kernel (§5.3).
- Through our large-scale experiments on several benchmark and real-world datasets, we demonstrate that subgraph2vec could significantly outperform state-of-the-art graph kernels (incl. [7]) on graph classification and clustering tasks (§6).

2. RELATED WORK

The closest work to our paper is Deep Graph Kernels [7]. Since we have discussed it elaborately in §1, we refrain from discussing it here. Recently, there has been significant interest from the research community on learning representations of nodes and other substructures from graphs. We list the prominent such works in Table 1 and show how our work compares to them in-principle. Deep Walk [8] and node2vec [10] intend to learn node embeddings by generating random walks in a single graph. Both these works rely on existence of node labels for at least a small portion of nodes and take a semi-supervised approach to learn node embeddings. Recently proposed Patchy-san [9] learns node and subgraph embeddings using a supervised convolutional neural network (CNN) based approach. In contrast to these three works, subgraph2vec learns subgraph embeddings (which includes node embeddings) in an unsupervised manner.

Table 1: Representation Learning from Graphs

Solution	Learning Paradigm	node vector	subgraph vector	Context used for rep. learning
Deep Walk [8]	Semi-sup	●	○	Fixed-length random walks
node2vec [10]	Semi-sup	●	○	Fixed-Length biased random walks
Patchy-san [9]	Sup	●	●	Receptive field of sequence of neighbours of nodes
Deep Graph Kernels [7]	Unsup	●	◓	Subgraphs occurring at same degree
subgraph2vec	Unsup	●	●	Subgraphs of different degrees occurring in the same local neighbourhoods

In general, from a substructure analysis point of view, research on graph kernel could be grouped into three major categories: kernels for limited-size subgraphs [12], kernels based on subtree patterns [6] and kernels based on walks [3] and paths [4]. subgraph2vec is complementary to these existing graph kernels where the substructures exhibit reasonable similarities among them.

3. PROBLEM STATEMENT

We consider the problem of learning distributed representations of rooted subgraphs from a given set of graphs. More formally, let $G = (V, E, \lambda)$, represent a graph, where V is a set of nodes and $E \subseteq (V \times V)$ be a set of edges. Graph G is labeled[2] if there exists a function λ such that $\lambda : V \rightarrow \ell$, which assigns a unique label from alphabet ℓ to every node $v \in V$. Given $G = (V, E, \lambda)$ and $sg = (V_{sg}, E_{sg}, \lambda_{sg})$, sg is a sub-graph of G iff there exists an injective mapping $\mu : V_{sg} \rightarrow V$ such that $(v_1, v_2) \in E_{sg}$ iff $(\mu(v_1), \mu(v_2)) \in E$.

Given a set of graphs $\mathcal{G} = \{G_1, G_2, ..., G_n\}$ and a positive integer D, we intend to extract a vocabulary of all (rooted) subgraphs around every node in every graph $G_i \in \mathcal{G}$ encompassing neighbourhoods of degree $0 \leq d \leq D$, such that $SG_{vocab} = \{sg_1, sg_2, ...\}$. Subsequently, we intend to learn distributed representations with δ dimensions for every subgraph $sg_i \in SG_{vocab}$. The matrix of representations (embeddings) of all subgraphs is denoted as $\Phi \in \mathbb{R}^{|SG_{vocab}| \times \delta}$.

Once the subgraph embeddings are learnt, they could be used to cater applications such as graph classification, clustering, node classification, link prediction and community detection. They could be readily used with classifiers such as CNNs and Recurrent Neural Networks. Besides this, these embeddings could be used to make a graph kernel (as in eq(2)) and subsequently used along with kernel classifiers such as SVMs and relational data clustering algorithms. These use cases are elaborated later in §5.4 after introducing the representation learning methodology.

4. BACKGROUND: LANGUAGE MODELS

Our goal is to learn the distributed representations of subgraphs extending the recently proposed representation learning and language modeling techniques for multi-relational data. In this section, we review the related background in language modeling.

Traditional language models. Given a corpus, the traditional language models determine the likelihood of a sequence of words appearing in it. For instance, given a sequence of words $\{w_1, w_2, ..., w_T\}$, n-gram language model

[2]For graphs without node labels, we follow the procedure mentioned in [6] and label nodes with their degree.

targets to maximize the following probability:

$$Pr(w_t|w_1, ..., w_{t-1}) \quad (3)$$

Meaning, they estimate the likelihood of observing the target word w_t given n previous words $(w_1, ..., w_{t-1})$ observed thus far.

Neural language models. The recently developed neural language models focus on learning distributed vector representation of words. These models improve traditional n-gram models by using vector embeddings for words. Unlike n-gram models, neural language models exploit the of the notion of context where a *context is defined as a fixed number of words surrounding the target word.* To this end, the objective of these word embedding models is to maximize the following log-likelihood:

$$\sum_{t=1}^{T} logPr(w_t|w_{t-c}, ..., w_{t+c}) \quad (4)$$

where $(w_t|w_{t-c}, ..., w_{t+c})$ are the context of the target word w_t. Several methods are proposed to approximate eq. (4). Next, we discuss one such a method that we extend in our subgraph2vec framework, namely Skipgram models [15].

4.1 Skip Gram

The skipgram model maximizes co-occurrence probability among the words that appear within a given context window. Give a context window of size c and the target word w_t, skipgram model attempts to predict the words that appear in the context of the target word, $(w_{t-c}, ..., w_{t-c})$. More precisely, the objective of the skipgram model is to maximize the following loglikelihood,

$$\sum_{t=1}^{T} log\ Pr(w_{t-c}, ..., w_{t+c}|w_t) \quad (5)$$

where the probability $Pr(w_{t-c}, ..., w_{t+c})$ is computed as

$$\Pi_{-c \leq j \leq c, j \neq 0} Pr(w_{t+j}|w_t) \quad (6)$$

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

$$\frac{exp(\Phi_{w_t}^T \Phi'_{w_{t+j}})}{\sum_{w=1}^{\mathcal{V}} exp(\Phi_{w_t}^T \Phi'_w)} \quad (7)$$

where Φ_w and $\Phi_w{}'$ are the input and output vectors of word w.

4.2 Negative Sampling

The posterior probability in eq. (6) could be learnt in several ways. For instance, a novice approach is to use a classifier like logistic regression. This is prohibitively expensive if the vocabulary of words is very large.

Negative sampling is an efficient algorithm that is used to alleviate this problem and train the skipgram model. Negative sampling selects the words that are not in the context at random instead of considering all words in the vocabulary. In other words, 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. An important intuition we extend in subgraph2vec is to view subgraphs in large graphs as words that are generated from a special language. In other words, different subgraphs compose graphs in a similar way that different words form sentences when used together. With this analogy, one can utilize word embedding models to learn dimensions of similarity between subgraphs. The main expectation here is that similar subgraphs will be close to each other in the embedding space.

Algorithm 1: SUBGRAPH2VEC $(\mathcal{G}, D, \delta, \mathfrak{e})$

input : $\mathcal{G} = \{G_1, G_2, ..., G_n\}$: set of graphs such that each graph $G_i = (V_i, E_i, \lambda_i)$ from which embeddings are learnt
D: Maximum degree of subgraphs to be considered for learning representations. This will produce a vocabulary of subgraphs, $SG_{vocab} = \{sg_1, sg_2, ...\}$ from all the graphs in $\mathcal{G}$
δ: number of dimensions (embedding size)
$\mathfrak{e}$: number of epochs
output: Matrix of vector representations of subgraphs $\Phi \in \mathbb{R}^{|SG_{vocab}| \times \delta}$

```
1  begin
2      SG_vocab = BuildSubgraphVocab(G) //use Algorithm 2
3      Initialization: Sample Φ from U^{|SGVocab| × δ}
4      for e = 0 to 𝔢 do
5          𝔊 = Shuffle (G)
6          for each G_i ∈ 𝔊 do
7              for each v ∈ V_i do
8                  for d = 0 to D do
9                      sg_v^(d) := GetWLSubgraph(v, G_i, d)
10                     RadialSkipGram (Φ, sg_v^(d), G_i, D)
11     return Φ
```

5. METHOD: LEARNING SUB-GRAPH REPRESENTATIONS

In this section we discuss the main components of our subgraph2vec algorithm (§5.2), how it enables making a deep learning variant of WL kernel (§5.3) and some of its usecases in detail (§5.4).

5.1 Overview

Similar to the language modeling convention, the only required input is a corpus and a vocabulary of subgraphs for subgraph2vec to learn representations. Given a dataset of graphs, subgraph2vec considers all the neighbourhoods of rooted subgraphs around every rooted subgraph (up to a certain degree) as its *corpus*, and set of all rooted subgraphs around every node in every graph as its *vocabulary*. Subsequently, following the language model training process with the subgraphs and their contexts, subgraph2vec learns the intended subgraph embeddings.

5.2 Algorithm: subgraph2vec

The algorithm consists of two main components; first a procedure to generate rooted subgraphs around every node in a given graph (§5.2.1) and second the procedure to learn embeddings of those subgraphs (§5.2.2).

As presented in Algorithm 1 we intend to learn δ dimensional embeddings of subgraphs (up to degree D) from all the graphs in dataset $\mathcal{G}$ in $\mathfrak{e}$ epochs. We begin by building a vocabulary of all the subgraphs, SG_{vocab} (line 2) (using

Algorithm 2: GetWLSubgraph (v, G, d)

input : v: Node which is the root of the subgraph
$G = (V, E, \lambda)$: Graph from which subgraph has to be extracted
d: Degree of neighbours to be considered for extracting subgraph
output: $sg_v^{(d)}$: rooted subgraph of degree d around node v

1 **begin**
2 $sg_v^{(d)} = \{\}$
if $d = 0$ **then**
3 $sg_v^{(d)} := \lambda(v$
4 **else**
5 $\mathcal{N}_v := \{v' \mid (v, v') \in E\}$
6 $M_v^{(d)} := \{\text{GetWLSubgraph}(v', G, d-1) \mid v' \in \mathcal{N}_v\}$
7 $sg_v^{(d)} := sg_v^{(d)} \cup$ GetWLSubgraph
$(v, G, d-1) \oplus sort(M_v^{(d)})$
8 **return** $sg_v^{(d)}$

Algorithm 3: RadialSkipGram $(\Phi, sg_v^{(d)}, G, D)$

1 **begin**
2 $context_v^{(d)} = \{\}$
3 **for** $v' \in$ Neighbours(G, v) **do**
4 **for** $\partial \in \{d-1,\ d,\ d+1\}$ **do**
5 **if** $(\partial \geq 0$ ***and*** $\partial \leq D)$ **then**
6 $context_v^{(d)} = context_v^{(d)} \cup$
GetWLSubgraph(v', G, ∂)
7 **for** ***each*** $sg_{cont} \in context_v^{(d)}$ **do**
8 $J(\Phi) = -\log \Pr\,(sg_{cont} | \Phi(sg_v^{(d)}))$
9 $\Phi = \Phi - \alpha \frac{\partial J}{\partial \Phi}$

Algorithm 2). Then the embeddings for all subgraphs in the vocabulary (Φ) is initialized randomly (line 3). Subsequently, we proceed with learning the embeddings in several epochs (lines 4 to 10) iterating over the graphs in $\mathcal{G}$. These steps represent the core of our approach and are explained in detail in the two following subsections.

5.2.1 Extracting Rooted Subgraphs

To facilitate learning its embeddings, a rooted subgraph $sg_v^{(d)}$ around every node v of graph G_i is extracted (line 9). This is a fundamentally important task in our approach. To extract these subgraphs, we follow the well-known WL relabeling process [6] which lays the basis for the WL kernel and WL test of graph isomorphism [6, 7]. The subgraph extraction process is explained separately in Algorithm 2. The algorithm takes the root node v, graph G from which the subgraph has to be extracted and degree of the intended subgraph d as inputs and returns the intended subgraph $sg_v^{(d)}$. When $d = 0$, no subgraph needs to be extracted and hence the label of node v is returned (line 3). For cases where $d > 0$, we get all the (breadth-first) neighbours of v in $\mathcal{N}_v$ (line 5). Then for each neighbouring node, v', we get its degree $d-1$ subgraph and save the same in list $M_v^{(d)}$ (line 6). Finally, we get the degree $d-1$ subgraph around the root node v and concatenate the same with sorted list $M_v^{(d)}$ to obtain the intended subgraph $sg_v^{(d)}$ (line 7).

Example. To illustrate the subgraph extraction process, lets consider the examples in Fig. 1. Lets consider the graph 1(c) as the complete graph from which we intend to get the degree 0, 1 and 2 subgraph around the root node `HttpURL.init`. Subjecting these inputs to Algorithm 2, we get subgraphs {`HttpURL.init`}, {`HttpURL.init -> OpenConnection`} and {`HttpURL.init -> OpenConnection -> Connect`} for degrees 0, 1 and 2, respectively.

5.2.2 Radial Skipgram

Once the subgraph $sg_v^{(d)}$, around the root node v is extracted, Algorithm 1 proceeds to learn its embeddings with the radial skip gram model (line 10). Similar to the vanilla skipgram algorithm which learns the embeddings of a target word from its surrounding linear context in a given document, our approach learns the embeddings of a target subgraph using its surrounding radial context in a given graph. The radial skipgram procedure is presented in Algorithm 3.

Modeling the radial context. The radial context around a target subgraph is obtained using the process explained below. As discussed previously in §4.1, natural language text have linear co-occurrence relationships. For instance, skipgram model iterates over all possible collocations of words in a given sentence and in each iteration it considers one word in the sentence as the *target word* and the words occurring in its context window as *context words*. This is directly usable on graphs if we model linear substructures such as walks or paths with the view of building node representations. For instance, Deep Walk [8] uses a similar approach to learn a target node's representation by generating random walks around it. However, unlike words in a traditional text corpora, subgraphs do not have a linear co-occurrence relationship. Therefore, we intend to consider the breadth-first neighbours of the root node as its context as it directly follows from the definition of WL relabeling process.

To this end, we define the context of a degree-d subgraph $sg_v^{(d)}$ rooted at v, as the multiset of subgraphs of degrees $d-1, d$ and $d+1$ rooted at each of the neighbours of v (lines 2-6 in Algorithm 3). Clearly this models a radial context rather than a linear one. Note that we consider subgraphs of degrees $d-1, d$ and $d+1$ to be in the context of a subgraph of degree d. This is because, as explained with example earlier in §1.1, a degree-d subgraph is likely to be rather similar to subgraphs of degrees that are closer to d (e.g., $d-1, d+1$) and not just degree-d subgraphs only.

Vanilla Skip Gram. As explained previously in §4.1, the vanilla skipgram language model captures fixed-length linear contexts over the words in a given sentence. However, for learning a subgraph's radial context arrived at line 6 in Algorithm 3, the vanilla skipgram model could not be used. Hence we propose a minor modification to consider a radial context as explained below.

Modification. The embedding of a target subgraph, $sg_v^{(d)}$, with context $context_v^{(d)}$ is learnt using lines 7 - 9 in Algorithm 3. Given the current representation of target subgraph $\Phi(sg_v^{(d)})$, we would like to maximize the probability of every subgraph in its context sg_{cont} (lines 8 and 9). We can learn such posterior distribution using several choices of classifiers. For example, modeling it using logistic regression would result in a huge number of labels that is equal to $|SG_{vocab}|$. This could be in several thousands/millions in the case of large graphs. Training such models would require large amount of computational resources. To alleviate this bottleneck, we approximate the probability distribution using the negative sampling approach.

5.2.3 Negative Sampling

Given that $sg_{cont} \in SG_{vocab}$ and $|SG_{vocab}|$ is very large, calculating $Pr(sg_{cont}|\Phi(sg_v^{(d)}))$ in line 8 is prohibitively expensive. Hence we follow the negative sampling strategy (introduced in §4.2) to calculate above mentioned posterior probability. In our negative sampling phase for every training cycle of Algorithm 3, we choose a fixed number of subgraphs (denoted as *negsamples*) as negative samples and update their embeddings as well. Negative samples adhere to the following conditions: if $negsamples = \{sgneg_1, sgneg_2, ...\}$, then $negsamples \subset SG_{vocab}$, $|negsamples| << |SG_{vocab}|$ and $negsamples \cap context_v^{(d)} = \{\}$. This makes $\Phi(sg_v^{(d)})$ closer to the embeddings of all the subgraphs its context (i.e.$\Phi(sg_{cont}) \mid \forall sg_{cont} \in context_v^{(d)}$) and at the same time distances the same from the embeddings of a fixed number of subgraphs that are not its context (i.e.$\Phi(sgneg_i) \mid \forall sgneg_i \in negsamples$).

5.2.4 Optimization

Stochastic gradient descent (SGD) optimizer is used to optimize these parameters (line 9, Algorithm 3). The derivatives are estimated using the back-propagation algorithm. The learning rate α is empirically tuned.

5.3 Relation to Deep WL kernel

As mentioned before, each of the subgraph in SG_{vocab} is obtained using the WL re-labelling strategy, and hence represents the WL neighbourhood labels of a node. Hence learning latent representations of such subgraphs amounts to learning representations of WL neighbourhood labels. Therefore, once the embeddings of all the subgraph in SG_{vocab} are learnt using Algorithm 1, one could use it to build the deep learning variant of the WL kernel among the graphs in $\mathcal{G}$. For instance, we could compute $\mathcal{M}$ matrix such that each entry $\mathcal{M}_{ij}$ computed as $\langle \Phi_i, \Phi_j \rangle$ where Φ_i corresponds to learned δ-dimensional embedding of subgraph i (resp. Φ_j). Thus, matrix $\mathcal{M}$ represents nothing but the pairwise similarities of all the substructures used by the WL kernel. Hence, matrix $\mathcal{M}$ could directly be plugged into eq. (2) to arrive at the deep WL kernel across all the graphs in $\mathcal{G}$.

5.4 Use cases

Once we compute the subgraph embeddings, they could be used in several practical applications. We list some prominent use cases here:

(1) Graph Classification. Given $\mathcal{G}$, 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 $\mathcal{H} : \mathcal{G} \rightarrow Y$. To this end, one could feed subgraph2vec's embeddings to a deep learning classifier such as CNN (as in [9]) to learn $\mathcal{H}$. Alternatively, one could follow a kernel based classification. That is, one could arrive at a deep WL kernel using the subgraph embeddings as discussed in §5.3, and use kernelized learning algorithm such as SVM to perform classification.

(2) Graph Clustering. Given $\mathcal{G}$, in graph clustering, the task is to group similar graphs together. Here, a graph kernel could be used to calculate the pairwise similarity among graphs in $\mathcal{G}$. Subsequently, relational data clustering algorithms such as Affinity Propagation (AP) [16] and Hierarchical Clustering could be used to cluster the graphs.

It is noted that subgraph2vec's use cases are not confined only to the aforementioned tasks. Since subgraph2vec could be used to learn node representations (i.e., when subgraph of degree 0 are considered, subgraph2vec provides node embeddings similar to Deep Walk [8] and node2vec [10]). Hence other tasks such as node classification, community detection and link prediction could also performed using subgraph2vec's embeddings. However, in our evaluations in this work we consider only graph classification and clustering as they are more prominent.

Table 2: 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

6. EVALUATION

We evaluate subgraph2vec's accuracy and efficiency both in supervised and unsupervised learning tasks. Besides experimenting with benchmark datasets, we also evaluate subgraph2vec on with real-world program analysis tasks such as malware and code clone detection on large-scale Android malware and clone datasets. Specifically, we intend to address the following research questions: (1) How does subgraph2vec compare to existing graph kernels for graph classification tasks in terms of accuracy and efficiency on benchmark datasets, (2) How does subgraph2vec compare to state-of-the-art graph kernels on a real-world unsupervised learning task, namely, code clone detection (3) How does subgraph2vec compare to state-of-the-art graph kernels on a real-world supervised learning task, namely, malware detection.

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

6.1 Classification on 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 statistics on the same are reported in Table 2. MUTAG dataset consists 188 chemical compounds where class label indicates whether or not the compound has a mutagenic effect on a bacterium. PTC dataset comprises of 344 compounds and the classes indicate carcinogenicity on female/male rats. PROTEINS is a graph collection where nodes are secondary structure elements and edges indicate neighborhood in the amino-acid sequence or in 3D space. NCI1 and NCI109 datasets contain compounds screened for activity against non-small cell lung cancer and ovarian cancer cell lines. Graphs are classified as enzyme or non-enzyme. All these datasets are made available in [6, 7].

Comparative Analysis. For classification tasks on each of the datasets, we use the embeddings learnt using subgraph2vec and build the Deep WL kernel as explained in §5.3. We compare subgraph2vec against the WL kernel [6] and Yanardag and Vishwanathan's formulation of deep WL kernel [7] (denoted as Deep WL_{YV}).

Configurations. For all the datasets, 90% of samples are chosen at random for training and the remaining 10% samples are used for testing. The hyper-parameters of the classifiers are tuned based on 5-fold cross validation on the training set.

Table 3: Average Accuracy (± std dev.) for subgraph2vec and state-of-the-art graph kernels on benchmark graph classification datasets

Dataset	MUTAG	PTC	PROTEINS	NCI1	NCI109
WL [6]	80.63 ± 3.07	56.91 ± 2.79	72.92 ± 0.56	80.01 ± 0.50	80.12 ± 0.34
Deep WL_{YV} [7]	82.95 ± 1.96	59.04 ± 1.09	73.30 ± 0.82	**80.31** ± 0.46	**80.32** ± 0.33
subgraph2vec	**87.17** ± 1.72	**60.11** ± 1.21	**73.38** ± 1.09	78.05 ± 1.15	78.39 ± 1.89

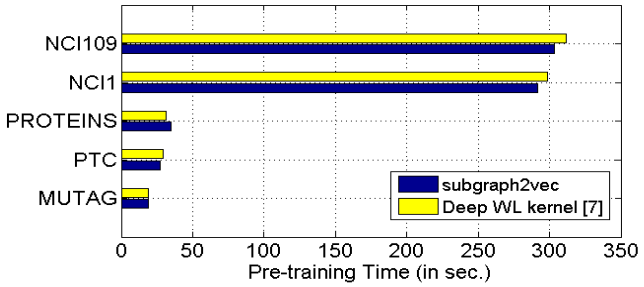

Figure 2: Deep WL kernel Vs subgraph2vec Pre-training Durations.

Table 4: Clone Detection Dataset Statistics

Dataset	# samples	# clusters	# nodes (avg.)	# edges (avg.)
$Clone_{260}$ [17]	260	100	9829.15	31026.30

Evaluation Metric. The experiment is repeated 5 times and the average accuracy (along with std. dev.) is used to determine the effectiveness of classification. Efficiency is determined in terms of time consumed for learning subgraph embeddings (*aka* pre-training duration).

6.1.1 *Results and Discussion.*

Accuracy. Table 3 lists the results of the experiments. It is clear that SVMs with subgraph2vec's embeddings achieve better accuracy on 3 datasets (MUTAG, PTC and PROTEINS) and comparable accuracy on the remaining 2 datasets (NCI1 and NCI109).

Efficiency. Out of the methods compared, only Deep WL_{YV} kernel and subgraph2vec involve pre-training to compute vectors of subgraphs. Evidently, pre-training helps them capture latent similarities between the substructures in graphs and thus aids them to outperform traditional graph kernels. Therefore, it is important to study the cost of pre-training. To this end, we report the pre-training durations of these two methods in Fig. 2. Being similar in terms of pre-training, both methods require very similar durations to build the pre-trained vectors. However, for the datasets under consideration, subgraph2vec requires lesser time than Deep WL_{YV} kernel as its radial skipgram involves slightly lesser computations than the vanilla skipgram used in Deep WL_{YV} kernel.

However it is important to note that classification on these benchmark datasets are much simpler than real-world classification tasks. In fact, by using trivial features such as number of nodes in the graph, [13] achieved comparable accuracies to the state-of-the-art graph kernels. It would be incomplete if we evaluate subgraph2vec only on these benchmark datasets. Hence in the two subsequent experiments, we involve real-world datasets on practical graph clustering and classification tasks.

6.2 Clone Detection

Android apps are cloned across different markets by unscrupulous developers for reasons such as stealing advertisement revenue [17]. Detecting and removing such cloned apps is an important task for app market curators that helps maintaining quality of markets and app ecosystem. In this experiment, we consider a set of Android apps and our goal is to cluster them such that clone (semantically similar) apps are grouped together. Hence, this amounts to unsupervised code similarity detection.

Table 5: Clone Detection - Results

Kernel	WL [6]	Deep WL_{YV} [7]	subgraph2vec
Pre-training duration	-	421.7 s	409.28 s
ARI	0.67	0.71	**0.88**

Dataset. We acquired a dataset of 260 apps collected from the authors of a recent clone detection work, 3D-CFG [17]. We refer to this dataset as $Clone_{260}$. All the apps in $Clone_{260}$ are manually analyzed and 100 clone sets (i.e. ground truth clusters) are identified by the authors of [17]. The details on this dataset are furnished in Table 4. As it could be seen from the table, this problem involves graphs that are much larger/denser than the benchmark datasets used in §6.1.

Our objective is to reverse engineer these apps, obtain their bytecode and represent the same as graphs. Subsequently, we cluster similar graphs that represent cloned apps together. To achieve this, we begin by representing reverse engineered apps as Inter-procedural Control Flow Graphs (ICFGs). Nodes of the ICFGs are labeled with Android APIs that they access[3]. Subsequently, we use subgraph2vec to learn the vector representations of subgraphs from these ICFGs and build a deep kernel matrix (using eq. (2)). Finally, we use AP clustering algorithm [16] over the kernel matrix to obtain clusters of similar ICFGs representing clone apps.

Comparative Analysis. We compare subgraph2vec's accuracy on the clone detection task against the WL [6] and Deep WL_{YV} [7] kernels.

Evaluation Metric. A standard clustering evaluation metric, namely, Adjusted Rand Index (ARI) is used to determine clone detection accuracy. The ARI values lies in the range [-1, 1]. A higher ARI means a higher correspondence to ground-truth clone sets.

6.2.1 *Results and Discussion.*

Accuracy. The results of clone detection using the three kernels under discussion are presented in Table 5. Following observations are drawn from the table:

- subgraph2vec outperform WL and Deep WL_{YV} kernels by more than 21% and 17% , respectively. The difference between using Deep WL kernel and subgraph2vec embeddings is more pronounced in the unsupervised learning task.
- WL kernel perform poorly in clone detection task as it, by design, fails to identify the subgraph similarities, which is essential to precisely captures the latent program semantics. On the other hand, Deep WL_{YV} kernel performs reasonable well as it captures similarities among subgraphs of same degree. However, it fails to capture the complete

[3]For more details on app representations, we refer to [11].

Table 6: Malware Detection Dataset Statistics

Dataset	Class	Source	# apps	# nodes (avg.)	# edges (avg.)
$\textbf{Train}_{10K}$	Malware	Drebin [18]	5600	9590.23	19377.96
	Benign	Google Play [2]	5000	20873.71	38081.24
$\textbf{Test}_{10K}$	Malware	Virus Share [1]	5000	13082.40	25661.93
	Benign	Google Play [2]	5000	27032.03	42855.41

semantics of the program due to its strong assumptions (see §1.2). Whereas, subgraph2vec was able to precisely capture subgraph similarities spanning across multiple degrees.

Efficiency. From Table 5, it can be seen that the pre-training duration for subgraph2vec is slightly better than Deep WL_{YV} kernel. This observation is inline with the pretraining durations of benchmark datasets. WL kernel involves no pre-training and deep kernel computation and hence much more efficient than the other two methods.

6.3 Malware Detection

Malware detection is a challenging task in the field of cyber-security as the attackers continuously enhance the sophistication of malware to evade novel detection techniques. In the case of Android platform, many existing works such as [11], represent benign and malware apps as ICFGS and cast malware detection as a graph classification problem. Similar to clone detection, this task typically involves large graphs as well.

Datasets. DREBIN [18] provides a collection of 5,560 Android malware apps collected from 2010 to 2012. We collected 5000 benign top-selling apps from Google Play [2] that were released around the same time and use them along with the Drebin apps to train the malware detection model. We refer to this dataset as TRAIN_{10K}. To evaluate the performance of the model, we use a more recent set of 5000 malware samples (i.e., collected from 2010 to 2014) provided by Virus share [1] and an equal number of benign apps from Google Play that were released around the same time. We refer to this dataset as TEST_{10K}. Hence, in total, our malware detection experiments involve 20,600 apps. The statistics of this dataset is presented in Table 6.

Comparative Analysis and Evaluation Metrics. The same type of comparative analysis and evaluation metrics against WL and Deep WL_{YV} kernels used in experiments with benchmark datasets in §6.1 are used here as well.

6.3.1 *Results & Discussion.*

Accuracy. The results of malware detection using the three kernels under discussion are presented in Table 7. Following observations are drawn from the table:

- SVM built using subgraph2vec embeddings outperform WL and Deep WL_{YV} kernels by more than 12% and 4%, respectively. This improvement could be attributed to subgraph2vec's high quality embeddings learnt from apps' ICFGs.
- On this classification task, both Deep WL_{YV} and subgraph2vec outperform WL kernel by a significant margin (unlike the experiments on benchmark datasets). Clearly, this is due to the fact that the former methods capture the latent subgraph similarities from ICFGs which helps them learn semantically similar but syntactically different malware features.

Efficiency. The inferences on pre-training efficiency discussed in §6.1 and §6.2 hold for this experiment as well.

Table 7: Malware Detection - Results

Classifier	WL [6]	Deep WL_{YV} [7]	subgraph2vec
Pre-training duration	-	2631.17 s	2219.28 s
Accuracy	66.15	71.03	**74.48**

7. CONCLUSION

In this paper, we presented subgraph2vec, an unsupervised representation learning technique to learn embedding of rooted subgraphs that exist in large graphs. Through our large-scale experiments involving benchmark and real-world graph classification and clustering datasets, we demonstrate that subgraph embeddings learnt by our approach could be used in conjunction with classifiers such as CNNs, SVMs and relational data clustering algorithms to achieve significantly superior accuracies. On real-world application involving large graphs, subgraph2vec outperforms state-of-the-art graph kernels significantly without compromising efficiency of the overall performance. We make all the code and data used within this work available at: https://sites.google.com/site/subgraph2vec

8. REFERENCES

[1] Virus Share malware dataset: http://virusshare.com
[2] Google Play Store: https://play.google.com/store
[3] Vishwanathan, S. V. N., et al. "Graph kernels." The Journal of Machine Learning Research 11 (2010): 1201-1242.
[4] Borgwardt, Karsten M., and Hans-Peter Kriegel. "Shortest-path kernels on graphs." Data Mining, Fifth IEEE International Conference on. IEEE, 2005.
[5] Shervashidze, Nino, et al. "Efficient graphlet kernels for large graph comparison." International conference on artificial intelligence and statistics. 2009.
[6] Shervashidze, Nino, et al. "Weisfeiler-lehman graph kernels." The Journal of Machine Learning Research 12 (2011): 2539-2561.
[7] Yanardag, Pinar, & S. V. N. Vishwanathan. "Deep graph kernels." Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, 2015.
[8] Perozzi, Bryan, et al. "Deepwalk: Online learning of social representations." Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2014.
[9] Niepert, Mathias, et al. "Learning Convolutional Neural Networks for Graphs." Proceedings of the 33rd annual international conference on machine learning. ACM, 2016.
[10] Grover, Aditya, & Leskovec, Jure. "node2vec: Scalable Feature Learning for Networks." Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, 2016.
[11] Narayanan, Annamalai, et al. "Contextual Weisfeiler-Lehman Graph Kernel For Malware Detection." The 2016 International Joint Conference on Neural Networks (IJCNN). IEEE, 2016.
[12] Horváth, Tamás, et al. Cyclic pattern kernels for predictive graph mining. In Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD), pages 158–167, 2004.
[13] Orlova, Yuliia, et al. "Graph kernel benchmark data sets are trivial!" ICML Workshop on Features and Structures FEATS 2015.
[14] Yanardag, Pinar, and S. V. N. Vishwanathan. "A Structural Smoothing Framework For Robust Graph Comparison." Advances in Neural Information Processing Systems. 2015.
[15] Mikolov, Tomas, et al. "Efficient estimation of word representations in vector space." ICLR Workshop, 2013.
[16] Frey, Brendan J., and Delbert Dueck. "Clustering by passing messages between data points." science 315.5814 (2007): 972-976.
[17] Chen, Kai, et al. "Achieving accuracy and scalability simultaneously in detecting application clones on android markets." Proceedings of the 36th International Conference on Software Engineering. ACM, 2014.
[18] Arp, Daniel, et al. "Drebin: Effective and explainable detection of android malware in your pocket." Proceedings of the Annual Symposium on Network and Distributed System Security (NDSS). 2014.