Graphs are topological and not geometrical, so cannot be used directly in polygonal transormations

Lee and Kong *Applied Network Science* (2019) 4:50
https://doi.org/10.1007/s41109-019-0157-9

Applied Network Science

RESEARCH Open Access

Learning compact graph representations via an encoder-decoder network

John Boaz Lee* and Xiangnan Kong

*Correspondence: jtlee@wpi.edu
Worcester Polytechnic Institute, Worcester, USA

Abstract

Feature representation learning for classification of multiple graphs is a problem with practical applications in many domains. For instance, in chemoinformatics, the learned feature representations of molecular graphs can be used to classify molecules which exhibit anti-cancer properties. In many previous work, including discriminative subgraph mining and graphlet-based approaches, a graph representation is derived by counting the occurrence of various graph sub-structures. However, these representations fail to capture the co-occurrence patterns that are inherently present among different sub-structures. Recently, various methods (*e.g.*, DeepWalk, node2vec) have been proposed to learn representations for nodes in a graph. These methods use node co-occurrence to learn node embeddings. However, these methods fail to capture the co-occurrence relationship between more complex sub-structures in the graph since they were designed primarily for node representation learning. In this work, we study the problem of learning graph representations that can capture the structural and functional similarity of sub-structures (as evidenced by their co-occurrence relationship) in graphs. This is particularly useful when classifying graphs that have very few sub-structures in common. The proposed method uses an encoder-decoder model to predict the random walk sequence along neighboring regions (or sub-structures) in a graph given a random walk along a particular region. The method is unsupervised and can be used to obtain generic feature representations of graphs making it applicable with various types of graphs. We evaluate the learned representations using several real-world datasets on the binary graph classification task. The proposed model is able to achieve superior results against multiple state-of-the-art techniques.

Keywords: Deep learning, Encoder-decoder, Graph classification, Graph representation, RNN

Introduction

Graph-structured data can be found in many different domains including biology, chemistry, and the study of social networks (Duvenaud et al. 2015; Hwang and Kuang 2010; Yanardag and Vishwanathan 2015). For instance, in chemistry, chemical compounds can be represented as molecular graphs where nodes represent atoms and edges signify the presence of a chemical bond between atoms (Duvenaud et al. 2015). In social network analysis, the interaction among different entities of a community can be represented as a graph with individuals as nodes and edges denoting social interactions

between individuals (Yanardag and Vishwanathan 2015). A natural question that arises in these scenarios is what the structure of a graph tells us about the properties of the graph (*e.g.,* what does the structure of a molecule tell us about the compound's aqueous solubility, or its anti-cancer activity?). In other words, we are often interested in classifying graph-structured data. Many techniques have been proposed to solve this problem. These include learning graph kernels (Vishwanathan et al. 2010), identifying discriminative subgraphs (Jin and Wang 2011; Kong et al. 2011), using specially designed neural network models such as the graph neural network (Scarselli et al. 2009), and learning graph fingerprints (Duvenaud et al. 2015).

In this paper, we study the graph representation learning problem, where the task is to learn feature representations of graphs for classification. An illustration of this is shown in Fig. 1. In particular, we investigate the use of an unsupervised model that can be used to learn compact graph representations for a large number of labeled or unlabeled graphs. The learned representations can be used directly with off-the-shelf classification methods like support vector machines (SVM), logistic regression, or neural networks.

Many existing work in the literature for calculating graph representations, like techniques based on discriminative subgraph mining (Kong et al. 2011; Natarajan and Ranu 2018; Wang et al. 2017) and ones involving graphlet counts (Ahmed et al. 2017; Shervashidze et al. 2009), make the assumption that different graph sub-structures are independent. Representations calculated using these approaches inevitably grow in size as more sub-structures are considered. Because of this, usually only a limited number of sub-structures are considered. Furthermore, previous work focus on the problem of graph representation learning without considering the co-occurrence relationships among different graph sub-structures. However, in many applications it is beneficial to calculate graph representations that capture the co-occurrence among sub-structures. This is particularly useful when there are a large number of structures and only a few are shared

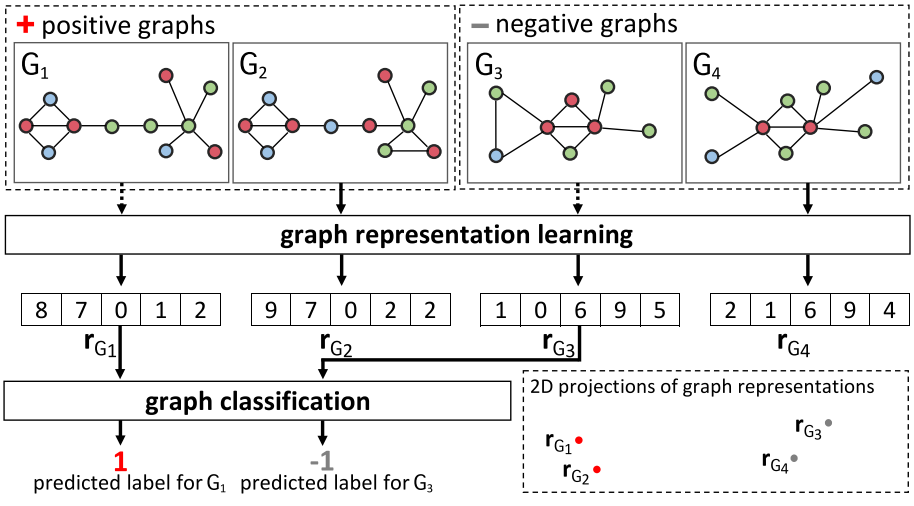


Fig. 1 An example of the representation learning problem for graph classification. Here, the graph representation model is trained using the graphs attached to solid lines (G_2 and G_4). The corresponding representations (for G_2 and G_4) are then used to train the classifier. The trained model can then be used to generate representations for unseen samples, shown with dashed lines here (G_1 and G_3)

between graphs. For instance, two molecules may not have many sub-structures in common but if they both have sub-structures that co-occur frequently in other molecules they may be considered functionally similar as their respective sub-structures seem to serve the same function. An example of this is shown in Fig. 2.

Inspired by the recent success of encoder-decoder models (Cho et al. 2014; Kalchbrenner and Blunsom 2013; Kiros et al. 2015) for modeling co-occurrence of text data, we propose a model to capture graph representations that reflect structural and functional similarity in graphs. This is done by considering the co-occurrence relationship of graph sub-structures. The proposed method has an advantage over existing techniques since the model can learn similar representations for graphs that do not necessarily have to be very structurally similar.

We propose a novel solution based upon the Skip-thought encoder-decoder model (Kiros et al. 2015). Skip-thought is a generalization of the skip-gram model (Mikolov et al. 2013) which was originally introduced in the natural language processing (NLP) domain for learning vector representations of words. Recently, the skip-gram model has been adapted successfully to solve the problem of learning node representations for graph-structured data (Grover and Leskovec 2016; Perozzi et al. 2014; Tang et al. 2015). These

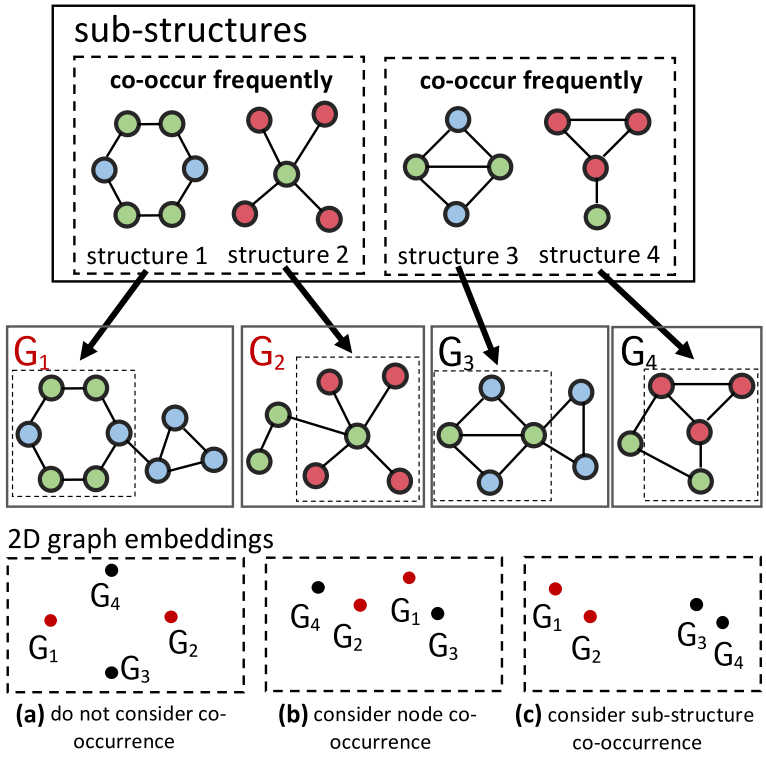


Fig. 2 (**a**) If we simply count or consider the sub-structures independently while ignoring their co-occurrence relationship (Kong et al. 2011; Natarajan and Ranu 2018; Vishwanathan et al. 2010; Wang et al. 2017), the representations will not be very similar. (**b**) On the other hand, if only node co-occurrence is considered (Grover and Leskovec 2016; Perozzi et al. 2014; Tang et al. 2015), graphs $\mathcal{G}_2$ and $\mathcal{G}_4$ (similarly, $\mathcal{G}_1$ and $\mathcal{G}_3$) end up having more similar representations (due to similar nodes) even though they do not share sub-structures that co-occur often. In (**c**), the approach considered in this paper, the representations for graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ (similarly, $\mathcal{G}_3$ and $\mathcal{G}_4$) end up being more similar because both graphs have sub-structures that co-occur frequently in the dataset

embeddings work well on node-level tasks such as link prediction. However, in many real-world applications we also need to learn feature representations for graphs and not just node embeddings.

In (Kiros et al. 2015), an encoder-decoder model is trained on a large text corpus and the final output of the encoder is used as the input sentence's representation. It has been shown that the model learns a function that maps semantically and syntactically similar sentences close to one another in representation space. In this work, the idea is to take instead a sequence generated by a random walk along a labeled graph and to divide it into three parts, feeding these into the encoder-decoder model. Since the structure of the graph determines the random walk sequences that can be generated, we can treat each sub-sequence as a representation of a particular subgraph in the graph. We argue that by training an encoder-decoder model on a large number of random walk sequences, we can learn a feature representation that groups structurally and functionally similar subgraphs together. We further expound on this point when we introduce the proposed encoder-decoder model. Fig. 3 shows an example of how we can train the model using a random walk over a graph.

After the model is trained on a large sample of random walks generated from a dataset of labeled graphs, we can then freeze the model and use the encoder as a feature extractor. In particular, we obtain a representation of a graph by sampling multiple short random walks and aggregating the information encoded in the feature representations of these short walks. We borrow an analogy from the NLP domain to highlight the idea. In order to obtain a good feature representation for a text document, short of sampling all the words in the document one may sample a set of sentences from the document and use these to construct the features for the document. Similarly, to obtain a feature representation for a graph, we sample a set of subgraphs (as represented by the short walks) and use the aggregate subgraph features to construct the final graph feature vector. Since we use the trained encoder as our feature extractor, graphs whose sub-structures share structural and functional properties will tend to have more similar feature vectors.

Problem formulation

Here, we briefly define the problem of representation learning for graphs. We are given a set of labeled graphs $\mathcal{D} = \{\mathcal{G}_1, \mathcal{G}_2, \cdots, \mathcal{G}_n\}$. Each graph $\mathcal{G}_i = (\mathcal{V}_i, \mathcal{E}_i, \ell_v)$ is comprised of a vertex set $\mathcal{V}_i$, an edge set $\mathcal{E}_i \subseteq \mathcal{V}_i \times \mathcal{V}_i$, and a node labeling function $\ell_v : V \rightarrow \mathcal{L}_V$ which

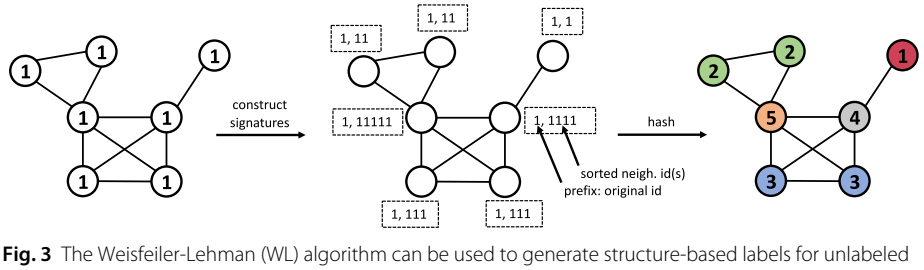


Fig. 3 The Weisfeiler-Lehman (WL) algorithm can be used to generate structure-based labels for unlabeled graphs. We initially assign the same id/label to each node in the graph. Each iteration of the algorithm is comprised of two steps: node signature construction and hashing. The illustration above shows one iteration. Each node's signature is prefixed by their current label followed by a sorted list of their neighbor's labels. The signatures are then hashed to derive new node labels

assigns each node to a label in $\mathcal{L}_V$. Here, $V = \bigcup_{i=1}^{n} \mathcal{V}_i$ and $\mathcal{L}_V$ is the set of all node labels. Additionally, the edges may also be labeled in which case we also have an edge labeling function $\ell_e : E \rightarrow \mathcal{L}_E$. Finally, each node $v \in V$ (similarly, an edge $e \in E$) can also have an associated attribute vector, $\mathbf{f}_v \in \mathbb{R}^X$ (similarly, $\mathbf{f}_e \in \mathbb{R}^Y$). X is the number of node attributes and Y is the number of edge attributes.

The task is then to learn a function $f_r : G \rightarrow \mathbb{R}^D$ that maps a graph to a D-dimensional representation, here G is the input space of graphs. The set of learned representations $\mathcal{R}$ can then be used to train a classifier $f_c : \mathcal{R} \rightarrow \mathcal{Y}$ that learns to predict the class label y_i of a graph $\mathcal{G}_i$ given its representation. Here, $\mathcal{Y}$ is the set of all class labels.

Unlabeled graphs

Although we will be working primarily with labeled graphs, our method can be easily extended to support unlabeled graphs by including an additional pre-processing step. Algorithms like the Weisfeiler-Lehman algorithm (Weisfeiler and Lehman 1968; Shervashidze et al. 2011) or the Morgan algorithm (Rogers and Hahn 2010) for calculating molecular fingerprints are iterative algorithms that work by repeatedly calculating the attribute for a node via hashing of the attributes of its neighboring nodes. The final node attributes capture the local structure or topology of the graph. For unlabeled graphs, all node attributes can be initialized to a constant value and after the algorithm is run, we can treat the node attributes as the labels for the nodes in the graph. We show an example of the Weisfeiler-Lehman algorithm in Fig. 4.

Furthermore, for more flexibility, we can also used the approach proposed in role2vec (Ahmed et al. 2018). The intuition is to allow users to construct structural feature vectors

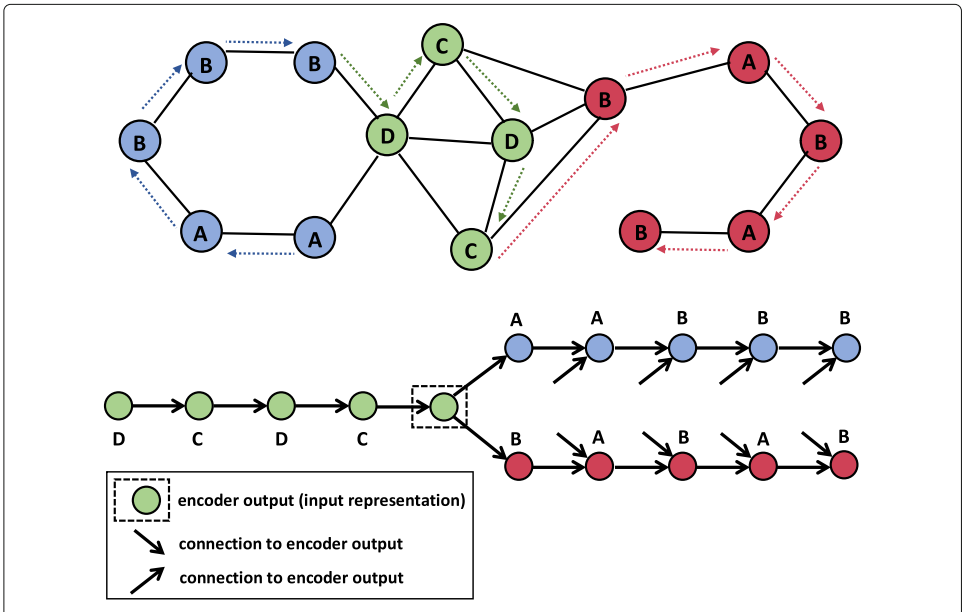


Fig. 4 A random walk over a graph can be split into three sub-sequences (s_1, s_2, s_3). The middle sequence is fed as input into the encoder and the decoders attempt to reconstruct the other sub-sequences. Note that the unattached arrows are connected to the encoder output to condition the decoder prediction at each step. Since the model processes entire sequences, it can distinguish the structural difference between the two neighboring sub-structures, even though they share the same types of nodes

for each node (*e.g.*, counts of various network motifs in a node's neighborhood). We can then use logarithmic binning to map nodes with similar features to the same label.

Proposed method

Skip-thought

We begin by introducing the general architecture of the particular encoder-decoder (Kiros et al. 2015) design we decided to employ. Here, a recurrent neural network (RNN), in particular using the Gated Recurrent Unit (GRU) architecture (Chung et al. 2014), is used as the encoder while a pair of RNNs with conditional GRU are used as decoders. The model is trained using the Adam stochastic optimization algorithm (Kingma and Ba 2015).

The input to the model is a triplet of sequences (s_{i-1}, s_i, s_{i+1}), with $\mathbf{x}_i^t$ being the t-th value in the sequence s_i. In the case where the sequences are sentences, each input $\mathbf{x}$ simply represents an embedding of a word in a sentence. The vectors $\mathbf{x}_i^t$ that comprise the middle sequence, s_i, are then fed sequentially as input into the encoder. The encoder generates a hidden vector $\mathbf{h}_i^t$ at each time step t, this is the information the model retained after processing the sub-sequence $\mathbf{x}_i^1, \cdots, \mathbf{x}_i^t$ and can be thought of as the representation of the particular sub-sequence. The hidden state $\mathbf{h}_i^{N_i}$ can thus be considered the representation of the entire sequence, where N_i is the length of sequence s_i. Given a sequence to encode, the encoder iterates through the following equations. Here the subscripts i are dropped for simplicity.

$$\mathbf{r}^t = \sigma\left(\mathbf{W}_r\mathbf{x}^t + \mathbf{U}_r\mathbf{h}^{t-1}\right) \tag{1}$$

$$\mathbf{z}^t = \sigma\left(\mathbf{W}_z\mathbf{x}^t + \mathbf{U}_z\mathbf{h}^{t-1}\right) \tag{2}$$

$$\bar{\mathbf{h}}^t = \tanh\left(\mathbf{W}\mathbf{x}^t + \mathbf{U}\left(\mathbf{r}^t \odot \mathbf{h}^{t-1}\right)\right) \tag{3}$$

$$\mathbf{h}^t = \left(1 - \mathbf{z}^t\right) \odot \mathbf{h}^{t-1} + \mathbf{z}^t \odot \bar{\mathbf{h}}^t \tag{4}$$

where $\mathbf{r}^t$ is the forget gate, $\mathbf{z}^t$ is the update gate, $\bar{\mathbf{h}}^t$ is the proposed hidden state, and $\odot$ is the component-wise product. Here $\mathbf{r}^t$ decides what information to discard from the previous state, $\mathbf{z}^t$ decides what new information to encode, and the new hidden vector $\mathbf{h}^t$ is calculated accordingly. Values in $\mathbf{r}^t$ and $\mathbf{z}^t$ are within the range $[0, 1]$.

Two decoders with separate parameters are then used to reconstruct the previous sequence s_{i-1} and the next sequence s_{i+1}. The computation for the decoder is similar to that of the encoder, except this time the models are also conditioned on the final encoder output or representation $\mathbf{h}_i$ (which is $\mathbf{h}_i^{N_i}$). Decoding involves iterating through the following statements. Again the subscript $i+1$ (similarly, $i-1$) is dropped.

$$\mathbf{r}^t = \sigma\left(\mathbf{W}_r^d\mathbf{x}^{t-1} + \mathbf{U}_r^d\mathbf{h}^{t-1} + \mathbf{C}_r\mathbf{h}_i\right) \tag{5}$$

$$\mathbf{z}^t = \sigma\left(\mathbf{W}_z^d\mathbf{x}^{t-1} + \mathbf{U}_z^d\mathbf{h}^{t-1} + \mathbf{C}_z\mathbf{h}_i\right) \tag{6}$$

$$\bar{\mathbf{h}}^t = \tanh\left(\mathbf{W}^d\mathbf{x}^{t-1} + \mathbf{U}^d\left(\mathbf{r}^t \odot \mathbf{h}^{t-1}\right) + \mathbf{C}\mathbf{h}_i\right) \tag{7}$$

$$\mathbf{h}_{i+1}^t = \left(1 - \mathbf{z}^t\right) \odot \mathbf{h}^{t-1} + \mathbf{z}^t \odot \bar{\mathbf{h}}^t \tag{8}$$

here the $\mathbf{C}$ matrices are used to bias the decoder computation using the representation produced by the encoder. Also, note that the input values $\mathbf{x}$ are from the previous time step since the decoder's job is to reconstruct the sequence s_{i+1} (similarly, s_{i-1}) one step at

a time. The probability of value $\mathbf{x}_{i+1}^t$ can then be calculated by

$$P\left(\mathbf{x}_{i+1}^t|\mathbf{x}_{i+1}^{<t},\mathbf{h}_i\right) \propto \exp\left(\mathbf{v}_{\mathbf{x}_{i+1}^t}\mathbf{h}_{i+1}^t\right) \tag{9}$$

where $\mathbf{v}_{\mathbf{x}_{i+1}^t}$ is the row vector in the "vocabulary" vector $\mathbf{V}$ corresponding to the input $\mathbf{x}_{i+1}^t$. The vocabulary matrix, $\mathbf{V}$, is a weight matrix shared by both decoders connecting the decoder's hidden states for computing a distribution over the inputs.

Finally, given a triplet of sequences, the training objective is then given by

$$\sum_t \log P\left(\mathbf{x}_{i+1}^t|\mathbf{x}_{i+1}^{<t},\mathbf{h}_i\right) + \sum_t \log P\left(\mathbf{x}_{i-1}^t|\mathbf{x}_{i-1}^{<t},\mathbf{h}_i\right) \tag{10}$$

which is the sum of log-probabilities for the values in the previous and next statements, s_{i-1} and s_{i+1}, conditioned on the final representation for s_i. The total objective would then be the above summed for all triplets used in the training data.

Skip-graph

We now discuss how an encoder-decoder can be used to learn useful representations for sub-sequences derived from walks over a labeled graph. Figure 3 shows an example of how a random walk over a graph can be fed into the encoder-decoder introduced above.

Training set generation

Given a set of graphs $\mathcal{D}$, a sample size K, a minimum random walk length l_{min}, and a maximum random walk length l_{max}, we take each graph $\mathcal{G} \in \mathcal{D}$ and generate K random walk sequences. Specifically, for a graph $\mathcal{G}$, K sequences of the form

$$\ell_v(v_1), \cdots, \ell_v(v_k), \ell_v\left(v_{k+1}\right), \cdots, \ell_v\left(v_{k+k'}\right), \ell_v\left(v_{k+k'+1}\right), \cdots, \ell_v\left(v_{k+k'+k''}\right) \tag{11}$$

are generated. Here, $v_1 \in \mathcal{V}$ is a randomly selected start node, $(v_i, v_{i+1}) \in \mathcal{E}$ for i from $1 \cdots k + k^{'} + k^{''} - 1$, and $l_{min} \geq k, k^{'}, k^{''} \geq l_{max}$. Each sequence can then be split into a triplet of three sub-sequences with $s_1 = \ell_v(v_1), \cdots, \ell_v(v_k)$, $s_2 = \ell_v\left(v_{k+1}\right), \cdots, \ell_v\left(v_{k+k'}\right)$, and $s_3 = \ell_v\left(v_{k+k'+1}\right), \cdots, \ell_v\left(v_{k+k'+k''}\right)$.

When generating sequences, $k, k^{'}$, and $k^{''}$ are randomly drawn to be between the constraints l_{min} and l_{max} each time. This is to ensure that the length of the sub-sequences do not need to have fixed lengths and can instead vary. Because of this, graph sub-structures or regions of varying sizes can easily be processed by the model.

In the above formulation, we assume that only the vertices in the graph are labeled and node and edge features are not given. When nodes, or edges, are labeled and feature vectors are provided we can use a one-hot embedding to represent each unique combination of labels and features. This treats each distinct combination as a unique "word" and does not capture the relationship between nodes or edges that share labels or certain features. A better approach is to simply use a one-of-$|\mathcal{L}|$ vector to encode the label and concatenate this with the feature vector, this allows the node or edge embedding to capture shared features and labels.

Once all the triplets of random walk sequences have been generated, they can be used to train the encoder-decoder[1] in an unsupervised fashion. The intuition behind this is quite simple. If an encoder-decoder model is trained with a large number of random walks, the sub-sequence corresponding to sub-structures in the graph that co-appear frequently

will have learned embeddings that are more similar. This allows us to learn representations for sub-structures that are more compact since the different sub-structures are not considered independently of one another. Figure 5 illustrates this idea.

Obtaining final graph representation

After the encoder-decoder has been trained, we can freeze the model and use the encoder to generate representations, $\mathbf{h}_i$, for any arbitrary random walk sequence. Ultimately, however, we are interested in obtaining representations for entire graphs so we try several strategies for aggregating the encoder representations obtained from a set of independent random walks sampled from a given graph. Sampling multiple short walks from a graph allows us to obtain a relatively accurate profile of a graph. The representations can then be aggregated to get a representation for the graph as a whole. While one can certainly try more sophisticated approaches for aggregation like a neural network that learns a final graph representation from the sampled representations, we choose to use relatively simple aggregation techniques to highlight the usefulness of the model.

1. **Single walk:** In this approach we do not use several encoder representations. Instead, we train the model on relatively long (relative to the size of the graphs in the dataset) random walk sequences and use a single long walk over the graph to obtain its representation.
2. **Average:** We compute the component-wise average of the encoder representations of the sampled random walk sequences. This is then used as the graph representation.
3. **Max:** As in (Kiela and Bottou 2014), we take the component-wise absolute maximum of all encoder representations.
4. **Cluster:** The encoder representations are first fed into a clustering technique like K-means (Hamerly and Elkan 2003) and we use the cluster information to create a bag-of-cluster vector that serves as the graph's representation.

The procedure for obtaining the graph embeddings is summarized in Algorithm 1. The calculated graph embeddings can now be used with any off-the-shelf classifier.

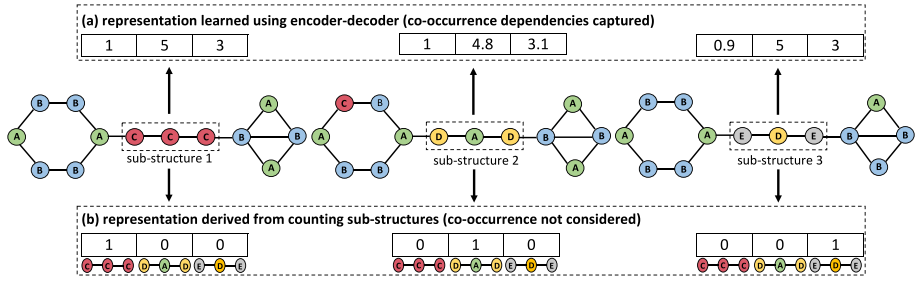


Fig. 5 (**a**) An encoder-decoder can be used to learn similar representations for sub-structures that share the same function. Here, the sub-structures "C-C-C", "D-A-D", and "E-D-E" are structurally dissimilar. However, they seem to be serving the same function of connecting similar regions together. If these patterns appear frequently, the encoder-decoder will learn to capture the functional similarity by learning similar representations for all three sub-structures. The learned representations are more compact since the co-occurrence dependencies of sub-structures are considered. (**b**) On the other hand, simply counting the occurrence of sub-structures to create a graph representation can result in representations that are less compact and less useful as the different sub-structures are treated independently

Algorithm 1: Calculate graph embedding

Input : Training set $\mathcal{D}$, sample size K, walk lengths l_{min} and l_{max}, aggregate sample size $K^{'}$, and aggregate method *agg*

Output: Graph embeddings

Generate set of $K \times |\mathcal{D}|$ random walk triplets, $\mathcal{S}$;
Train encoder-decoder model using $\mathcal{S}$;
for *each* $\mathcal{G}$ *in* $\mathcal{D}$ **do**
    Randomly select $K^{'}$ random walks;
    Obtain encoder representations $\mathbf{h}_1, ..., \mathbf{h}_{K'}$ from the random walks;
    Compute graph embedding with *agg* $\left(\mathbf{h}_1, ..., \mathbf{h}_{K'}\right)$;
end
Return final graph embeddings;

Time complexity

The overall time it takes to train an encoder-decoder model depends on two things: the size of the training set $\mathcal{D}$, and the average length of the walks in each triplet. In previous work, an encoder-decoder was trained on a very large dataset with 74,004,228 sequences with average length of 13, demonstrating that the model can be trained in a relatively reasonable amount of time on large datasets (Kiros et al. 2015).

Once the unsupervised training of the model is complete, we can proceed to compute a graph's embedding (even for an unseen sample) in time $\mathcal{O}(K' \cdot T \cdot d^2)$. As mentioned previously, K' is the number of random walks we use to calculate the final graph embedding, T is the average length of the random walks, and d is the embedding size (for simplicity we assume that the input size is equal to the embedding size).

Experiments

Dataset

We evaluate our proposed method on the binary classification task using three chemical compound datasets (Kong et al. 2011). The datasets contain chemical compounds encoded in the simplified molecular-input line-entry system (SMILES) format (Weininger 1988); class labels indicate the anti-cancer properties (active or inactive) of each compound. We use the RDKit[2] package to obtain the molecular graphs from the SMILES data. We also use RDKit to obtain the labels for the nodes (atom type) and edges (bond type). Additionally, like previous work (Duvenaud et al. 2015), we used the number of attached hydrogens as a node feature and bond conjugation as an edge feature. Since the edges in the datasets we evaluate on are also labeled, the generated random walk sequences include edges. The datasets are all highly skewed with far more negative samples than positive ones, we tested the methods on balanced datasets by selecting a random set of negative samples equal to the positive ones.

Table 1 shows a summary of the datasets used. The average size of the molecular graphs in each of the four datasets is around 30. Although the compared datasets all come from the chemoinformatic domain, the chemical compounds in each dataset are screened for different purposes. In NCI81, we screen for anti-cancer properties against Colon Cancer while NCI83 deals with Breast Cancer. In the final dataset, we screen for anti-viral properties against the human immunodeficiency virus (HIV).

Table 1 Summary of experimental datasets

Dataset	# graphs	# pos	Details
NCI81	40700	1396	Colon Cancer
NCI83	27992	2276	Breast Cancer
HIV	7781	266	HIV Anti-virus

"# pos" stands for the number of positive samples.

Compared methods

We compared our proposed approach against several state-of-the-art techniques. Our primary objective is to see whether the method has the potential to learn useful and compact representations for graph classification so we compare against a variety of approaches for graph classification in the literature. Since we are testing the method using molecular graph datasets, we first compare against techniques that have achieved state-of-the-art performance on these type of graphs including one that uses a deep learning framework that is end-to-end differentiable (Duvenaud et al. 2015). We also compared against several general graph kernel-based approaches. Finally, we tested against a modified version of a method designed to learn node embeddings and used the same aggregation techniques we discussed to obtain a final graph representation. We provide more information on the compared methods below.

- **ECFP** (Rogers and Hahn 2010): Extended-connectivity circular fingerprints, which are a refinement of the Morgan algorithm (Morgan 1965), use an iterative approach to encode information about substructures in a molecular graph in a fingerprint vector. In this method a hash function is used to map the concatenated features from a neighborhood to an index in the fingerprint vector. This method uses the same iterative process that the WL graph-kernel (Weisfeiler and Lehman 1968; Shervashidze et al. 2011) employs to generate an initial graph representation.
- **NeuralFPS** (Duvenaud et al. 2015): Neural fingerprints replace the function that is used to compute a fingerprint vector with a differentiable neural network. This allows the method to learn from the data, prioritizing useful or discriminative features. One can think of this method as end-to-end differentiable version of the WL algorithm.
- **DeepWalk** (Perozzi et al. 2014): The DeepWalk model was originally designed to learn representations for nodes in a single graph. We modify it slightly and train the model using random walks from multiple graphs. Since the various graphs in our dataset share the same types of node, the model will then learn to generate similar representations for nodes that co-occur frequently across all the graphs. To generate the final embedding for a graph, we simply apply average pooling to the vectors of all the nodes in the graph – which is a reasonable strategy to capture the overall profile of the graph.
- **3-GK & SP** (Borgwardt and Kriegel 2005; Shervashidze et al. 2009; Yanardag and Vishwanathan 2015): We compare against the graphlet kernel and the shortest-path kernel – both of which support labeled graphs. The former calculates similarity between a pair of graphs by counting subgraphs while the latter uses shortest-paths to measure similarity. In the experiments of (Yanardag and Vishwanathan 2015), there is not a huge difference in performance between graph kernels and their

deep-learning variant on tests performed on chemoinformatic datasets. Because of this, we only compare against traditional graph kernels.

- **Skip-graph**: Our proposed method. We train an encoder-decoder model using random walks generated from the graphs and use the encoder's random walk representations to calculate the final graph embedding.

To test ECFP and NeuralFPS, we used the library[3] provided by (Duvenaud et al. 2015). The size of the graph embedding was restricted to 164 for all applicable methods and a grid-search was done to optimize the parameters of the various methods. For ECFP and NeuralFPS, we tested different values for the following parameters: fingerprint radius, ℓ_2 regularization penalty, step size for the optimization, hidden layer dimension, and convolution layer dimension (only for NeuralFPS). All results reported are the average over 5-fold cross validation. Since a neural network, with a single hidden layer, was used as the classifier in (Duvenaud et al. 2015), we chose to use the same classifier for all baselines. Furthermore, for a fair comparison, a grid-search was performed over the same set of values for classifier-related parameters. In particular, for the neural network, we tested various settings with hidden layer size selected from $\{70, 100, 140\}$, and ℓ_2 regularization chosen from $\{0.0001, 0.001, 0.01, 0.1\}$.

All tests were conducted on a machine running Ubuntu with 160GB of memory and 48 CPU cores (Intel Xeon E5-2680 v3 @ 2.5 GHz).

Classification results

We show the classification accuracy of the different methods in Table 2. The proposed method achieves top performance in all of the tested datasets. It is a little surprising, however, to find that NeuralFPS performs slightly worse than ECFP. This seems to suggest that it is overfitting the data as NeuralFPS is a generalization of ECFP and should, in theory, be at least as good as ECFP. We find that the methods based on graph kernels perform quite poorly. This may be due to the fact that we are using a combination of the atom type and the additional node feature as the node "label" leading to an increase in the number of possible features – which, in turn, causes sparsity where only a few sub-structures are shared across graphs. This phenomenon has been shown in previous experiments, see (Yanardag and Vishwanathan 2015) for instance.

Finally, we find that averaging the DeepWalk embeddings trained from random walks generated from the entire training set can be a simple yet effective way to generate a graph representation. When DeepWalk is run with window size set to 3, one can consider it to be a special case of the proposed method where the sub-sequences are constrained to

Table 2 Summary of experimental results

Method	Dataset		
	HIV	NCI81	NCI83
ECFP	68.30%	68.90%	62.06%
NeuralFPS	67.48%	65.24%	59.91%
DeepWalk	69.90%	68.00%	**63.89%**
3-GK	61.78%	53.34%	53.01%
SP	59.10%	53.02%	56.26%
Skip-graph (ours)	**72.77%**	**69.98%**	**63.80%**

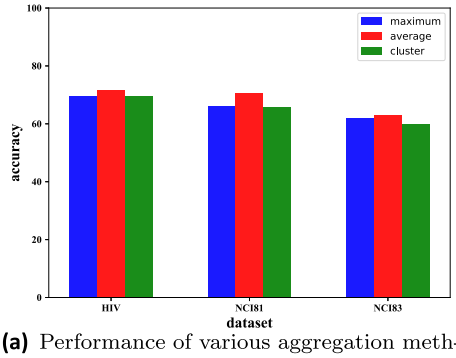
The best result(s) per dataset are highlighted.

be of length 1. These results seem to show that in some cases aggregation at the node level is sufficient. This may be the case, in particular, for graphs that are relatively small with not too many different complex sub-structures. However, we see from the results that focusing on the subgraph level can certainly be beneficial and can lead to better performance.

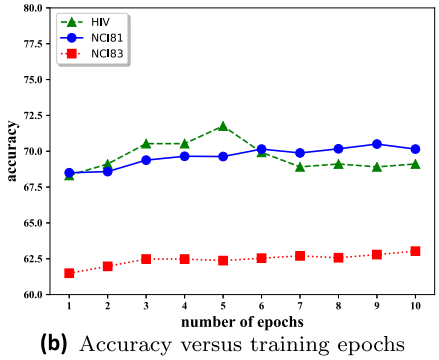
Since it is possible to generate many different sets of random walks to train the encoder-decoder model, we tried training five distinct encoders on five independent sets of random walks. An ensemble (Opitz and Maclin 1999) of five classifiers is then created with each classifier trained on the graph representations obtained from one of the five encoders. We compare the predictive accuracy of the ensemble versus the single classifier when all other settings are fixed. We observed a slight improvement (around $1-3\%$) in the accuracy of the model. To maintain a fair comparison and to keep the method simple, however, all the results reported above are for the single classifier case.

Parameter study

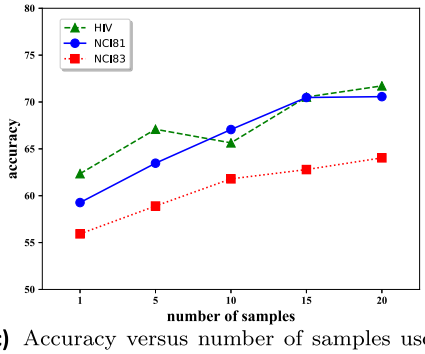
We tested the performance of the method using the various aggregation methods. The performance was extremely poor when we trained the encoder-decoder model on long random walks and used a single long walk to generate the graph representation. The other three aggregation strategies yielded better results. Figure 6a shows the performance of these methods. Averaging the hidden vector representations seems to yield the best performance, calculating the component-wise maximum yielded the second best results



(a) Performance of various aggregation methods



(b) Accuracy versus training epochs



(c) Accuracy versus number of samples used for aggregation

Fig. 6 The performance of our proposed method under various settings. **a** Performance of various aggregation methods, **b** Accuracy versus training epochs, **c** Accuracy versus number of samples used for aggregation

while the method that had the additional cluster pre-processing step performed slightly worse.

We plot the accuracy of the method over the number of training epochs in Fig. 6b. With the exception of the HIV dataset, which has a relatively few number of samples, the results show a gradual increase in the classification accuracy as the number of training epochs is increased. This is consistent with results in other work that show that given a large number of training data, recurrent neural models generally achieve better results when trained longer. Since we used a relatively large number of samples to train the encoder-decoder model ($K = 100$ random walks were generated from each graph), the learned embeddings already yielded good results even after a single epoch of training.

Figure 6c shows the accuracy in the classification task over different sample sizes K', or the number of samples aggregated to obtain the final graph representation. It is clear from the results that a better graph representation is obtained if we use more samples to calculate the final graph representation. This is quite intuitive as a limited sample may not be representative and may fail to capture the properties of the graph well enough.

We tested several different values for l_{min} and l_{max} and the one that seemed to perform best in our case was $l_{min} = 7$ and $l_{max} = 12$. This is a reasonable constraint on the random walk length given that the average size of the molecular graphs was around 30. We used $K = 100$ when generating a set of random walks to train the encoder-decoder. Finally, the encoder-decoder model was trained using a batch-size of 64.

Visualization of graph embeddings

We show a scatterplot of the HIV graph embeddings learned by our model in Fig. 7. In particular, we highlight two pairs of graphs that had very similar embeddings. We note that the first pair of graphs (the one on the right) are structurally similar, that is they have a large sub-structure in common. The graphs in the second pair each contain two similar substructures that are joined by segments that appear to be “functionally" similar.

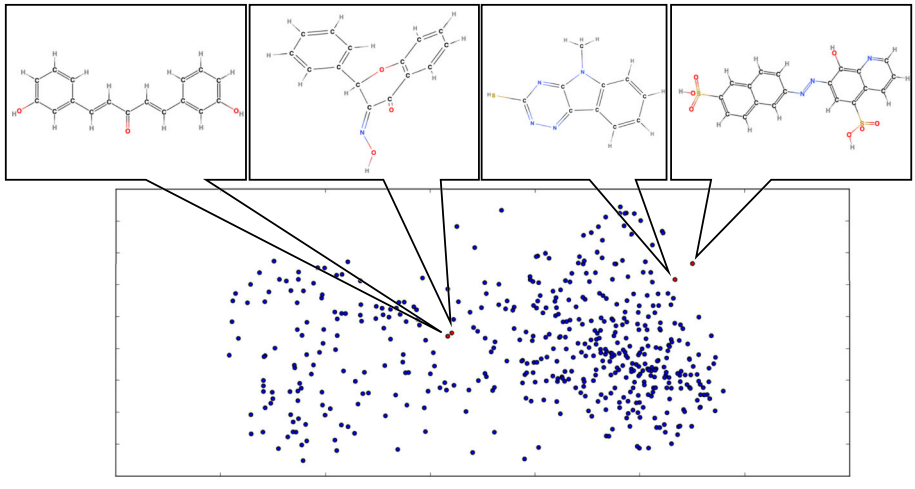


Fig. 7 The embeddings for graphs in the HIV dataset visualized in 2D space (we used Kernel PCA (Mika et al. 1998) for dimension reduction). We highlight two different pairs of embeddings that are close to each other – to show structurally and “functionally” similar graphs that have learned embeddings that are also similar. We include the other embeddings to give the reader a sense of the proximity of the graphs in our example to other graphs and to each other

Related Work

One popular approach to measure similarity between various graph objects is to use a graph kernel method. Kernel methods utilize a kernel function which corresponds to an inner product in reproducing kernel Hilbert space (RKHS) (Vishwanathan et al. 2010) to calculate similarity between pairs of graphs. The dot product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ in RKHS $\mathcal{H}$ is performed between graph representations (*i.e.*, vectors) which capture structural properties of the graphs. Various structural properties have been considered, and these include the shortest path between nodes (Borgwardt and Kriegel 2005), the count of various graphlets or subgraphs (Shervashidze et al. 2009), decomposed directed acyclic graph patterns (Martino et al. 2012), and even the structure of the graph at varying scales (Kondor and Pan 2016).

More recently, various methods have been proposed that generalize over previous approaches. These methods learn data-driven graph representations using deep learning approaches (Duvenaud et al. 2015; Yanardag and Vishwanathan 2015; Ying et al. 2018; Zhang et al. 2018). For instance, one can think of (Duvenaud et al. 2015) as an end-to-end differentiable version of the Weisfeiler-Lehman graph kernel (Shervashidze et al. 2011). In contrast to NeuralFPS (Duvenaud et al. 2015) which relies on flat "message-passing" steps, DiffPool (Ying et al. 2018) and deep graph convolutional neural networks (Zhang et al. 2018) are recently-introduced models with differentiable graph pooling operations to learn a hierarchical set of representations for graphs. This is similar to how a conventional convolutional neural network learns hierarchical features for images. For DiffPool, each pooling step coarsens the input graph by clustering the nodes into a small set of clusters. Since these work (Duvenaud et al. 2015; Ying et al. 2018; Zhang et al. 2018) are end-to-end differentiable they can learn task-relevant graph embeddings. However, Skip-graph has the advantage of being able to take a large number of graphs without task labels for training since the encoder-decoder is trained in an unsupervised fashion. This is particularly useful since it is usually quite costly to label large datasets.

Perhaps the work that resembles this work the most is that of (Yanardag and Vishwanathan 2015). However, there are several key differences between their work and ours. In our work, the use of RNNs as encoders and decoders allow us to more naturally learn embeddings for sub-sequences that differ slightly. This is in contrast to (Yanardag and Vishwanathan 2015) where each sub-structure is treated as a unique "word." Furthermore, we utilize the skip-thought model (Kiros et al. 2015) which is comprised of an encoder and two decoders to capture structural and also functional similarity in sub-structures.

The network embedding problem has also received much attention recently where the goal is to learn embeddings for nodes in graphs using node co-occurrence patterns (Grover and Leskovec 2016; Perozzi et al. 2014; Tang et al. 2015; Huang et al. 2017). In their seminal paper, (Perozzi et al. 2014) trained a skip-gram model (Mikolov et al. 2013) using random walks on a graph to generate node embeddings. Various extensions to the original approach have been proposed, including node2vec (Grover and Leskovec 2016) which introduced the concept of a biased random walk. Accelerated Attributed Network Embedding (Huang et al. 2017) is another approach that calculates attribute-sensitive embeddings for attributed networks. These methods learn node embeddings that are suitable for node-level tasks such as link prediction (Miller et al. 2009), node clustering (Vinayak et al. 2014), and item recommendation (Wang et al. 2016). In all of the above-mentioned approaches, the node embeddings are learned for nodes on a single graph only.

In this work, we consider multiple graphs. Furthermore, we study the problem of learning representations for entire graphs (as opposed to individual nodes) which is more suitable for graph-level tasks like graph classification.

Encoder-decoder models have been applied successfully to various tasks in the NLP domain including machine translation (Cho et al. 2014), semantic relatedness prediction (Kiros et al. 2015), paraphrase detection (Kiros et al. 2015), and image captioning (Vinyals et al. 2015). To the best of our knowledge, this is the first work that uses an encoder-decoder model for the task of graph representation learning.

Conclusion

We introduced an unsupervised method, based on the encoder-decoder model, for generating compact feature representations for graph-structured data. The learned representations were evaluated on the binary classification task on several real-world datasets. The method outperformed several state-of-the-art algorithms on the tested datasets.

There are several interesting directions for future work. For instance, we can try training multiple encoders on random walks generated using very different neighborhood selection strategies. This may allow the different encoders to capture different properties in the graphs yielding better performance. We would also like to test the approach using other deep learning architectures.

Endnotes

[1] We use the implementation in https://github.com/ryankiros/skip-thoughts.

[2] http://www.rdkit.org/

[3] https://github.com/HIPS/neural-fingerprint

Abbreviations

ECFP: Extended-connectivity circular fingerprints; GRU: Gated recurrent unit; HIV: Human immunodeficiency virus; NLP: Natural language processing; RKHS: Reproducing kernel hilbert space; RNN: Recurrent neural network; SVM: Support vector machine; WL: Weisfeiler-Lehman algorithm

Acknowledgements

Not applicable.

Authors' contributions

JBL and XK conceived of the idea of the study. JBL implemented the methods and ran the experiments. JBL and XK analyzed the results and wrote the paper. All authors read and approved the final manuscript.

Funding

This work is supported in part by National Science Foundation through grant IIS-1718310.

Availability of data and materials

The HIV dataset analyzed during the current study is available from the AIDS Anti-Viral Screen Program, https://dtp.cancer.gov/. The NCI datasets analyzed during the current study are available from PubChem, https://pubchem.ncbi.nlm.nih.gov/.

Competing interests

The authors declare that they have no competing interests.

Received: 2 February 2019 Accepted: 18 June 2019
Published online: 18 July 2019

References

Ahmed N, Neville J, Rossi R, Duffield N, Willke T (2017) Graphlet decomposition: framework, algorithms, and applications. Knowl Inf Syst 50:689–722

Ahmed NK, Rossi R, Lee JB, Willke TL, Zhou R, Kong X, Eldardiry H (2018) Learning role-based graph embeddings. In: International Workshop on Statistical Relational AI @ IJCAI '18. pp 1–8

Borgwardt K, Kriegel HP (2005) Shortest-path kernels on graphs. In: Proceedings of International Conference on Data Mining. pp 74–81

Cho K, van Merrienboer B, Gulcehre C, Bahdanau D, Bougares F, Schwenk H, Bengio Y (2014) Learning phrase representations using rnn encoder-decoder for statistical machine translation. In: Proceedings of Conference on Empirical Methods in Natural Language Processing. pp 1724–1734

Chung J, Gulcehre C, Cho K, Bengio Y (2014) Empirical evaluation of gated recurrent neural networks on sequence modeling. In: Neural Information Processing Systems Deep Learning Workshop
Duvenaud DK, Maclaurin D, Aguilera-Iparraguirre J, Gomez-Bombarelli R, Hirzel T, Aspuru-Guzik A, Adams RP (2015) Convolutional networks on graphs for learning molecular fingerprints. In: Proceedings of Conference on Neural Information Processing Systems,. pp 2224–2232
Grover A, Leskovec J (2016) node2vec: Scalable feature learning for networks. In: Proceedings of SIGKDD Conference on Knowledge Discovery and Data Mining. pp 855–864
Hamerly G, Elkan C (2003) Learning the k in k-means. In: Proceedings of Conference on Neural Information Processing Systems. pp 281–288
Huang X, Li J, Hu X (2017) Accelerated attributed network embedding. In: Proceedings of SIAM International Conference on Data Mining. pp 633–641
Hwang T, Kuang R (2010) A heterogeneous label propagation algorithm for disease gene discovery. In: Proceedings of SIAM International Conference on Data Mining. pp 583–594
Jin N, Wang W (2011) LTS: Discriminative subgraph mining by learning from search history. In: Proceedings of International Conference on Data Engineering. pp 207–218
Kalchbrenner N, Blunsom P (2013) Recurrent continuous translation models. In: Proceedings of Conference on Empirical Methods in Natural Language Processing. pp 1700–1709
Kiela D, Bottou L (2014) Learning image embeddings using convolutional neural networks for improved multi-modal semantics. In: Proceedings of Conference on Empirical Methods in Natural Language Processing. pp 36–45
Kingma D, Ba J (2015) Adam: A method for stochastic optimization. In: Proceedings of International Conference on Learning Representations
Kiros R, Zhu Y, Salakhutdinov R, Zemel RS, Urtasun R, Torralba A, Fidler S (2015) Skip-thought vectors. In: Proceedings of Conference on Neural Information Processing Systems. pp 3294–3302
Kondor R, Pan H (2016) The multiscale lapla- cian graph kernel. In: Proceedings of Conference on Neural Information Processing Systems. pp 2982–2990
Kong X, Fan W, Yu PS (2011) Dual active feature and sample selection for graph classification. In: Proceedings of SIGKDD Conference on Knowledge Discovery and Data Mining. pp 654–662
Martino GDS, Navarin N, Sperduti A (2012) A tree-based kernel for graphs. In: Proceedings of SIAM International Conference on Data Mining. pp 975–986
Mika S, Scholkopf B, Smola A, Muller K, Scholz M, Ratsch G (1998) Kernel PCA and de-noising in feature spaces. In: Proceedings of Conference on Neural Information Processing Systems
Mikolov T, Chen K, Corrado G, Dean J (2013) Efficient estimation of word representations in vector space. In: Proceedings of International Conference on Learning Representations
Miller KT, Griffiths TL, Jordan MI (2009) Nonparametric latent feature models for link prediction. In: Proceedings of Conference on Neural Information Processing Systems. pp 1276–1284
Morgan H (1965) The generation of a unique machine description for chemical structure. J Chem Doc 5:107–113
Natarajan D, Ranu S (2018) RESLING: a scalable and generic framework to mine top-k representative subgraph patterns Knowledge and Information Systems. Knowl Inf Syst 54(1):123–149
Opitz D, Maclin R (1999) Popular ensemble methods: An empirical study. J Artif Intell Res 11:169–198
Perozzi B, Al-Rfou' R, Skiena S (2014) DeepWalk: online learning of social representations. In: Proceedings of SIGKDD Conference on Knowledge Discovery and Data Mining. pp 701–710
Rogers D, Hahn M (2010) Extended-connectivity fingerprints. J Chem Inf Model 50:742—754
Scarselli F, Gori M, Tsoi A, Hagenbuchner M, Monfardini G (2009) Computational capabilities of graph neural networks. IEEE Trans Neural Netw 20:1938–1949
Shervashidze N, Vishwanathan S, Petri T, Mehlhorn K, Borgwardt K (2009) Efficient graphlet kernels for large graph comparison. In: Proceedings of International Conference on Artificial Intelligence and Statistics. pp 488–495
Shervashidze N, Schweitzer P, van Leeuwen E, Mehlhorn K, Borgwardt K (2011) Weisfeiler-lehman graph kernels. J Mach Learn Res 12:2539–2561
Tang J, Qu M, Wang M, Zhang M, Yan J, Mei Q (2015) LINE: Large-scale information network embedding. In: Proceedings of International World Wide Web Conference. pp 1067–1077
Vinayak RK, Oymak S, Hassibi B (2014) Graph clustering with missing data: Convex algorithms and analysis. In: Proceedings of Conference on Neural Information Processing Systems. pp 2996–3004
Vinyals O, Toshev A, Bengio S, Erhan D (2015) Show and tell: A neural image caption generator. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition. pp 3156–3164
Vishwanathan SVN, Schraudolph NN, Kondor R, Borgwardt KM (2010) Graph kernels. J Mach Learn Res 11:1201–1242
Wang H, Zhang P, Zhu X, Tsang I, Chen L, Zhang C, Wu X (2017) Incremental subgraph feature selection for graph classification. IEEE Trans Knowl Data Eng 29:128–142
Wang X, Xu C, Guo Y, Qian H (2016) Constrained preference embedding for item recommendation. In: Proceedings of International Joint Conference on Artificial Intelligence. pp 2139–2145
Weininger D (1988) SMILES, a chemical language and information system. J Chem Inf Model 28:31–36
Weisfeiler B, Lehman A (1968) A reduction of a graph to a canonical form and an algebra arising during this reduction. Nauchno-Technicheskaya Informatsiya 2:12–16
Yanardag P, Vishwanathan S (2015) Deep graph kernels. In: Proceedings of SIGKDD Conference on Knowledge Discovery and Data Mining. pp 1365–1374
Ying R, You J, Morris C, Ren X, Hamilton WL, Leskovec J (2018) Hierarchical graph representation learning with differentiable pooling. In: Proceedings of Conference on Neural Information Processing Systems. pp 4805–4815
Zhang M, Cui Z, Neumann M, Chen Y (2018) An end-to-end deep learning architecture for graph classification. In: Proceedings of AAAI Conference on Artificial Intelligence. pp 4438–4445

Publisher's Note