Applied Network Science

RESEARCH Open Access

# Sequence-to-sequence modeling for graph representation learning

Aynaz Taheri[1]*, Kevin Gimpel[2] and Tanya Berger-Wolf[1]

*Correspondence: ataher2@uic.edu
[1]University of Illinois at Chicago, Chicago, USA
Full list of author information is available at the end of the article

**Abstract**
We propose sequence-to-sequence architectures for graph representation learning in both supervised and unsupervised regimes. Our methods use recurrent neural networks to encode and decode information from graph-structured data. Recurrent neural networks require sequences, so we choose several methods of traversing graphs using different types of substructures with various levels of granularity to generate sequences of nodes for encoding. Our unsupervised approaches leverage long short-term memory (LSTM) encoder-decoder models to embed the graph sequences into a continuous vector space. We then represent a graph by aggregating its graph sequence representations. Our supervised architecture uses an attention mechanism to collect information from the neighborhood of a sequence. The attention module enriches our model in order to focus on the subgraphs that are crucial for the purpose of a graph classification task. We demonstrate the effectiveness of our approaches by showing improvements over the existing state-of-the-art approaches on several graph classification tasks.

**Keywords:** Graph representation learning, Deep learning, Graph classification, Recurrent models

## Introduction

We address the problem of comparing and classifying graphs by learning their latent representation. This problem arises in many domain areas, including bioinformatics, social network analysis, chemistry, neuroscience, and computer vision. For instance, in neuroscience, comparing brain networks represented by graphs helps to identify brains with neurological disorders (Van Wijk et al. 2010). In social network analysis, we may need to compare egonetworks to detect anomalies (Akoglu et al. 2010) or to identify corresponding egonetworks across multiple social networks. Cutting across domains, we may be interested in understanding how to distinguish the structure of a social network from that of a biological network, an authorship network, a computer network, or a citation network (Newman 2003).

For many years, graph kernels (Vishwanathan et al. 2010; Gärtner et al. 2003; Shervashidze et al. 2009, 2011; Borgwardt and Kriegel 2005), feature extraction methods (Berlingerio et al. 2012; Macindoe and Richards 2010; Yan and Han 2002) and graph matching algorithms (Bunke 2000; Riesen et al. 2010) have been the approaches of choice for graph comparison. Recently, remarkable advancements have been made in developing methods for embedding nodes (Grover and Leskovec 2016; Perozzi et al. 2014; Tang et al. 2015; García-Durán and Niepert 2017), edges (Chen et al. 2018; Trivedi et al. 2017; Rossi et al. 2018) and subgraphs (Narayanan et al. 2016; Yanardag and Vishwanathan 2015; Adhikari et al. 2017) into low-dimensional spaces. Moreover, several other approaches (Niepert et al. 2016, Zhang et al. 2018, Duvenaud et al. 2015, Gilmer et al. 2017, li et al. 2015b, Lee et al. 2018b) have been proposed to learn graph representations for a supervised task such as graph classification or regression. These approaches are aimed at optimizing the performance of the classification task, rather than learning the explicit topology of a graph.

There is still a lack of well-performing approaches for learning the representation for an entire graph. There are several challenges that need to be addressed within this area. First, the choice of the subgraph structures to be incorporated in the graph representation learning has a significant impact on the expressiveness power of the embeddings of an entire graph. Second, choosing the appropriate granularity level of this substructure (*e.g.*, whether to include first or second order neighborhoods of a node when building node sequences), which is necessary to preserve the graph embedding, is an open problem. The choice may depend on many factors, such as the graph domain, scale, density, and its various structural properties. The types of the substructures, from fine-to-coarse, such as nodes, edges, trees, graphlets, random walks, and communities, can capture local and global features of the graph. The question is what types of substructures with what level of granularity are informative enough to capture the general graph structure and recognize similarity between graphs, while reducing the loss of information? The additional challenge is, of course, the efficiency of learning the representation of the substructures and aggregating them into a graph embedding. In this work, we investigate these challenges within the context of our proposed architectures.

Moreover, most of the recent studies (Ying et al. 2018; Niepert et al. 2016; Zhang et al. 2018; Duvenaud et al. 2015; Li et al. 2015b) focus on Graph Neural Networks (GNNs) to investigate the graph representation learning problem. Gilmer et al. (2017) introduced a message passing framework and explored the family of GNNs by massage propagation in the graph topology. In this group of approaches, the hidden representations of the nodes are iteratively updated, using differentiable functions, from the hidden representations of their neighbors. Several concerns regrading the scalability and training time arise since these approaches rely on several iterations of message passing using the entire adjacency matrix. Therefore, in this work we investigate whether it is possible to achieve efficient graph representations by sampling a set of sequences from the graph and solely rely on them in order to find a hidden representation for the whole graph structure. Recently, Lee et al. (2018b) proposed an approach for graph representation learning using embedding sequences. However, their approach is designed in a supervised manner and is not applicable without task-specific supervision. In this work, we study graph representation learning techniques using recurrent models in both supervised and unsupervised regimes.

In the unsupervised representation learning, we only consider a setting in which we have knowledge about the graph structure, such as an adjacency matrix as well as node and edge properties. In the supervised representation learning, we assume knowledge about the supervised task, such as graph classification labels or regression values, in addition to the information about the graph structure. In both settings, we use effective models based on a sequence-to-sequence learning framework and demonstrate the capability of our models in learning graph representation with or without incorporating task-specific supervision information.

In the unsupervised regime, we leverage the LSTM sequence-to-sequence learning framework of (Sutskever et al. 2014), which uses one LSTM to encode the input sequence into a vector and another LSTM to generate the output sequence from that vector. In some variations of our models, we use the same sequence as both input and output, making this a sequence-to-sequence LSTM autoencoder (Li et al. 2015a). We consider several types of substructures including random walks, shortest paths, and breadth-first search, with various levels of node neighborhood granularity to prepare the input to our models. An unsupervised graph representation approach can be used not only in processing labeled data, such as in graph classification in bioinformatics, but can be also used in many practical applications, such as anomaly detection in social networks or streaming data, as well as in exploratory analysis and scientific hypothesis generation. An unsupervised method for learning graph representations provides a fundamental capability to analyze graphs based on their intrinsic properties. Figure 1 shows the result of using our unsupervised approach to embed the graphs in the Proteins dataset (Borgwardt et al. 2005) into a 100-dimensional space, visualized with t-SNE (Maaten and Hinton 2008). Each point is one of the 1113 graphs in the dataset. The two class labels were not used when learning the graph representations, but we still generally find that graphs of the same class are clustered in the learned space.

In the supervised regime, we suggest a supervised version of a sequence-to-sequence learning framework to predict graph classification labels. The supervised model is inspired by the best variation of our unsupervised framework in order to utilize the neighborhood of a sequence for the purpose of the graph classification task. One recurrent neural network is used to gather information from a sequence and another recurrent model is applied to incorporate the information from the neighborhood of the sequence into the graph representation learning. Moreover, our architecture is equipped with a

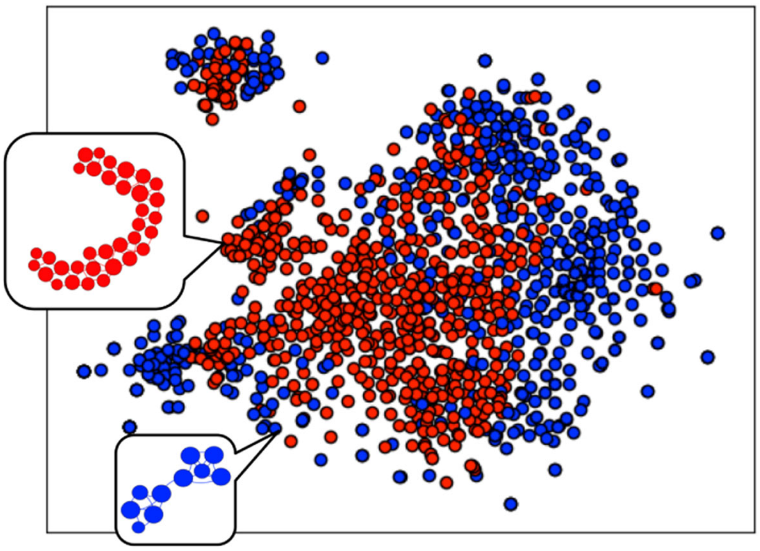
**Fig. 1** Learned graph representations in Proteins (Borgwardt et al. 2005) dataset, which has two classes: enzyme (red) and non-enzyme (blue)

two-level attention mechanism to improve our classification performance by leveraging local and global information from the neighborhoods of nodes in a sequence. Recent graph representation approaches (Lee et al. 2018a, 2018b; Veličković et al. 2018) introduce the notion of attention mechanism to explore the neighbors of a node and detect the most informative neighbors. However, our proposed architecture utilizes two levels of attention to capture more global information from the neighborhood of the entire sequence in addition to the neighbors of a node.

We demonstrate the efficacy of our supervised and unsupervised approaches in classification tasks for both labeled and unlabeled graphs. Our approach outperforms the state-of-the-art methods on nearly all the considered datasets.

## Related work

We discuss prior and relevant work in developing unsupervised and supervised methods for graph comparison.

### Unsupervised

In the unsupervised group, existing graph comparison methods can be categorized into three main (not necessarily disjoint) classes: *feature extraction*, *graph kernels*, and *graph matching*. *Feature extraction* methods compare graphs across a set of features, such as specific subgraphs or numerical properties that capture the topology of the graphs (Berlingerio et al. 2012; Macindoe and Richards 2010; Yan and Han 2002). The efficiency and performance of such methods is highly dependent on the feature selection process. Most *graph kernels* (Vishwanathan et al. 2010) are based on the idea of R-convolutional kernels (Haussler 1999), a way of defining kernels on structured objects by decomposing the objects into substructures and comparing pairs in the decompositions. For graphs, the substructures include graphlets (Shervashidze et al. 2009), shortest paths (Borgwardt and Kriegel 2005), random walks (Gärtner et al. 2003), and subtrees (Shervashidze et al. 2011). Recently, several new graph kernels such as Deep Graph Kernel (DGK) (Yanardag and Vishwanathan 2015), optimal-assignment Weisfeiler-Lehman (WL-OA) (Kriege et al. 2016), Pyramid Match Kernel (PM) (Nikolentzos et al. 2017) and local WL label (LWL) (Morris et al. 2017) have been proposed and evaluated for graph classification tasks. The DGK (Yanardag and Vishwanathan 2015) uses methods from unsupervised learning of word embeddings to augment the kernel with substructure similarity. WL-OA (Kriege et al. 2016) is an assignment kernel that finds an optimal bijection between different parts of the graph. The PM kernel (Nikolentzos et al. 2017) finds an approximate correspondence between the sets of vectors of the two graphs. LWL (Morris et al. 2017) is a kernel that considers both local and global features of the graph. While graph kernel methods are effective, their time complexity is quadratic in the number of graphs, and there is no opportunity to customize their representations for supervised tasks.

*Graph matching* algorithms use the topology of the graphs, their nodes and edges directly, counting matches and mismatches (Bunke 2000; Riesen et al. 2010). These approaches do not consider the global structure of the graphs and are sensitive to noise.

In addition to the three categories above, Graph2vec (Narayanan et al. 2017) is and unsupervised method inspired by document embedding models. This approach finds a representation for a graph by maximizing the likelihood of existing graph subtrees given the graph embedding. Our approach outperforms this method by a large margin.

Graph2vec does not capture global information in the graph structure by only considering subtrees as graph representatives, which ultimately affects its performance. Other methods have been developed to learn representations for individual nodes in graphs, such as DeepWalk (Perozzi et al. 2014), node2vec (Grover and Leskovec 2016), and many others. These methods are not directly related to graph comparison because they require aggregating node representations to represent entire graphs. However, we compared to one such representative method in our experiments to show that the aggregation of node embeddings is not informative enough to represent the structure of a graph.

## Supervised

We now discuss supervised methods that learn graph representations for the graph classification task. The representations obtained by these approaches are tailored for a supervised task and are not based solely on the graph topology. Most of existing approaches (Niepert et al. 2016; Zhang et al. 2018; Ying et al. 2018; Gilmer et al. 2017; Duvenaud et al. 2015; Li et al. 2015b; Bruna et al. 2013; Henaff et al. 2015; Defferrard et al. 2016; Scarselli et al. 2009) are variations of Graph Neural Networks (GNNs) and rely on the idea of message propagation around the neighbors. Niepert et al. (2016) developed a framework (PSCN) to learn graph representations by defining receptive fields of neighborhoods and using canonical node ordering. Deep Graph Convolutional Neural Network (DGCNN) (Zhang et al. 2018) is another model that extracts multi-scale node features and applies a consistent pooling layer on unordered nodes. The main difference between DGCNN and PSCN is the way they deal with the node-ordering problem. We compare our models to them in our experiments, outperforming them on all datasets. Ying et al. (2018) proposed a hierarchical representation learning framework via hierarchical GNNs pooling layers. Gilmer et al. (2017) proposed a message passing neural network framework, and explored the existing supervised approaches (Gilmer et al. 2017; Duvenaud et al. 2015; Li et al. 2015b) that have been recently used for graph-structured data in chemistry applications, such as molecular property prediction. Duvenaud et al. (Duvenaud et al. 2015) introduced a GNN to create “fingerprints" (vectors that encode molecule structure) for graphs derived from molecules. The information about each atom and its neighbors are fed to the neural network, and neural fingerprints are used to predict new features for the graphs. Bruna et al. (2013) proposed spectral networks, generalizations of GNNs on low-dimensional graphs via graph Laplacians. Henaff et al. (2015) and Defferrard et al. (2016) extended spectral networks to high-dimensional graphs. Scarselli et al. (2009) proposed a GNN which extends recursive neural networks and find node representations using random walks. Li et al. (2015b) extended GNNs with gating recurrent neural networks to predict sequences from graphs. In general, neural message passing approaches can suffer from high computational and memory costs, since they perform multiple iterations of updating hidden node states in graph representations. However, our supervised approach obtains strong performance without the requirement of passing messages between nodes for multiple iterations.

Lee et al. (2018b) proposed Graph Attention Model (GAM) using recurrent neural networks for the graph classification task. The goal is to train a model that is able to distinguish between different classes of graphs. This approach is limited to the graph classification application and cannot address the problems of graph comparison and graph representation learning without any task-specific supervision. GAM captures the graph representation by traversing the graph via attention-guided walks. In fact, the approach decides how to guide a walk while traversing the graph by choosing the nodes that are more informative for the purpose of the graph classification. Finally, the nodes selected during the graph traversal and their corresponding attributes are used for graph classification. However, our architecture is flexible enough to work with different choices of extracted sequences such as BFS, shortest paths as well as random walks, and to benefit from various types of substructures in learning to represent the graphs. GAM mainly focuses on local information obtained via attention mechanism on neighbors of nodes and is not capable of considering global structure of the graph. However, by using extracted sequences with different order of neighborhood granularity and applying a two-level attention mechanism, we can retrieve informative information from both local and global structure of the graph, which ultimately improves performance.

## Background

We briefly discuss the required background about graphs and LSTM recurrent neural networks.

**Graphs.** For a graph $G = (V, E)$, $V$ denotes its node set and $E \subseteq V \times V$ denotes its edge set. Edge $e \in E$ is a pair of nodes $(v, v') \in V \times V$ and represents an undirected edge. The set of neighbors for a node $v$ is $Nbrs(v) = \{v' | (v, v') \in E\}$. $G$ is called a *labeled graph* if there is a labeling function $label : V \rightarrow L$ that assigns a label from a set of labels $L$ to each node. The graph $G$ is called an *unlabeled graph* if no labels have been assigned to its nodes.

**Long short-term memory (LSTM).** An LSTM (Hochreiter and Schmidhuber 1997) is a recurrent neural network (RNN) designed to model long-distance dependencies in sequential data. RNNs are a class of neural networks that use their internal hidden states to process sequences of inputs. We denote the input vector at time $t$ by $x_t$ and we denote the hidden vector computed at time $t$ by $h_t$. At each time step, an LSTM computes a memory cell vector $c_t$, an input gate vector $i_t$, a forget gate vector $f_t$, and an output gate vector $o_t$:

$$\begin{aligned} i_t &= \sigma(W_i x_t + U_i h_{t-1} + K_i c_{t-1} + b_i) \\ f_t &= \sigma(W_f x_t + U_f h_{t-1} + K_f c_{t-1} + b_f) \\ o_t &= \sigma(W_o x_t + U_o h_{t-1} + K_o c_{t-1} + b_o) \\ c_t &= f_t \odot c_{t-1} + i_t \odot \tanh(W_c x_t + U_c h_{t-1} + b_c) \\ h_t &= o_t \odot \tanh(c_t) \end{aligned} \tag{1}$$

where $\odot$ denotes elementwise multiplication, $\sigma$ is the logistic sigmoid function, each $W$ is a weight matrix connecting inputs to particular gates (denoted by subscripts), each $U$ is an analogous matrix connecting hidden vectors to gates, each $K$ is a diagonal matrix connecting cell vectors to gates, and each $b$ is a bias. We refer to this as an "LSTM encoder" because it converts an input sequence into a sequence of hidden vectors $h_t$. We will also use a type of LSTM that predicts the next item $\overline{x}$ in the sequence from $h_t$. This architecture, which we refer to as an "LSTM decoder," adds the following:

$$\overline{x} = g(h_t) \tag{2}$$

where $g$ is a function that takes a hidden vector $h_t$ and outputs a predicted observation $\overline{x}$. With symbolic data, $g$ typically computes a softmax distribution over symbols and returns the max-probability symbol. When using continuous inputs, $g$ could be an affine transform of $h_t$ followed by a nonlinearity.

## Approach overview

We propose unsupervised and supervised architectures for learning representations of labeled and unlabeled graphs. In our unsupervised architecture, the goal is to learn graph embeddings such that graphs with similar structure lie close to one another in the embedding space. We seek to learn a mapping function $\Phi : G \rightarrow \mathbb{R}^k$ that embeds a graph $G$ into a $k$-dimensional space. We are interested in methods that scale linearly in the number of graphs in a dataset (as opposed to graph kernel methods that require quadratic time).

Our approach uses encoder-decoder models, particularly autoencoders (Hinton and Zemel 1993), which form one of the principal frameworks of unsupervised learning. Autoencoders are typically trained to reconstruct their input in a way that learns useful properties of the data. There are two parts to an autoencoder: an encoder that maps the input to some intermediate representation, and a decoder that attempts to reconstruct the input from this intermediate representation. We need to decide how to represent graphs in a form that can be encoded and then reconstructed. We do this by extracting ode sequences with various levels of granularity (the increasing order of node neighborhoods) from the graphs. "Generating sequences from graphs" section describes several methods for doing this. Given node sequences from a graph, we then need an encoding-decoding framework that can handle variable-length sequences. We choose LSTMs for both the encoder and decoder, forming an LSTM autoencoder (Li et al. 2015a). LSTM autoencoders use one LSTM to read the input sequence and encode it to a fixed dimensional vector, and then use another LSTM to decode the output sequence from the vector. We consider several variations for the encoder and decoder, described in "Sequence-to-sequence encoder-decoder" section. We experiment with two training objectives, described in "Training" section.

Given the trained encoder $LSTM_{enc}$, we define the graph embedding function $\Phi(G)$ as the mean of the vectors output by the encoder over $Seq(G)$, the set of graph sequences extracted from $G$:

$$\Phi(G) = \frac{1}{|Seq(G)|} \sum_{s \in Seq(G)} LSTM_{enc}(s). \tag{3}$$

Using the mean outperformed max pooling in our experiments so we only report results using the mean in this paper. We use $\Phi$ to represent graphs in our experiments in "Experiments" section, demonstrating state-of-the-art performance for several graph classification tasks.

Our supervised model uses a sequence-to-sequence framework to capture the graph structure with the supervision of a graph classification task. We train our model to predict the label of a graph using randomly selected sets of node sequences over multiple iterations. The model leverages the node embeddings learned by our unsupervised encoder-decoder models in learning to predict graph labels. We describe our supervised approach in "Supervised graph representation learning" section and compare it with other supervised and unsupervised approaches in "Experiments" section.

## Generating sequences from graphs

The input of our model is a set of node sequences generated from different types of graph substructures such as *Random Walks*, *Shortest Paths*, and *Breadth-First Search*. Different substructures capture varying aspects of local vs global topology of the graph. Moreover, we incorporate information about the neighborhood of a node sequence in order to preserve more information about the region of the graph that is traversed by the sequence. The neighborhood of a sequence consists of the neighbors of nodes in that sequence. In order to incorporate the neighborhood of a sequence in graph representation learning, we assign a label to each node in the sequence based on its neighborhood. The order of the neighborhood (which we call granularity) determines whether it is the immediate neighbors (1st order neighborhood) or the neighbors of neighbors as well (2nd order neighborhood), etc. For instance, the $k^{th}$-order granularity for sequence $s : v_1, \ldots, v_{|s|}$, considers all the nodes in the graph that have a distance of $k$ or less from a $v_i \in s$. We examine the impact of various substructures and orders of granularity on our models.

### Types of substructures

**Random Walks (RW)**: Given a source node $u$, we generate a random walk $w_u$ with fixed length $m$. Let $v_i$ denote the $i$th node in $w_u$, starting with $v_0 = u$. $v_{t+1}$ is a node from the $Nbrs(v_t)$ that is selected with probability $1/deg(v_t)$, where $deg(v_t)$ is the degree of $v_t$. (This is a 0-order Random Walk in our context. A higher order Random Walk replaces nodes with higher order neighborhoods–see the following "Substructure granularity" section).

**Shortest Paths (SP)**: We generate all the shortest paths between each pair of nodes in the graph using the Floyd-Warshall algorithm (Floyd 1962).

**Breadth-First Search (BFS)**: We run the BFS algorithm at each node to generate graph sequences for that node. The graph sequences for the graph include the BFS sequences starting at each node in the graph, limited to a maximum number of edges from the starting node. We give details on the maximum used in our experiments below.

### Substructure granularity

Each of the sequences defined in the previous section can be sequences of nodes at different orders of granularity. For example, a Random Walk or a Shortest Path can be a sequence of nodes with 0-order granularity, a sequence of nodes with first-order granularity, or a sequence of nodes with the second-order granularity, etc.

We use *Weisfeiler-Lehman* (WL) algorithm (Weisfeiler and Lehman 1968) to generate labels for nodes, encoding the node neighborhood (of the correpsonding order of granularity) information in the label. This algorithm is typically used as a graph isomorphism test. It is known as an iterative node classification or node refinement procedure (Shervashidze et al. 2011). The WL algorithm uses multiset labels to encode the local structure of the graphs. The idea is to create a multiset label for each node using the sorted list of its neighbors' labels. Then, the sorted list is compressed into a new value. The WL algorithm can iterate this labeling process and add higher order neighbors to the neighborhood list at each iteration. This labeling process continues until the new multiset labels of graphs in the dataset are different or the number of iterations reaches a specified limit. Each iteration increases the order of substructure granularity by expanding the node neighborhood. The WL labels can enrich the information provided by a node, regardless of whether the original graph is labeled or unlabeled.

Each unique WL label is converted to a parameter vector in a $k$-dimensional space where each entry is initialized with a draw from a uniform distribution with range $[-1, 1]$. Overall, depending on the granularity that we consider for nodes, we may include different number of parameter vectors in the model. WL parameter vectors are updated during the training time in addition to the model parameters. For each node $v$ in a sequence, $Emb(v)$ denotes the parameter vector assigned to the corresponding WL label of node $v$.

## Sequence-to-sequence encoder-decoder

We discussed three types of substructures used for extracting sequences from graphs and our approach for embedding nodes considering the order of their neighborhoods. We now describe how we will learn our graph embedding functions in our unsupervised setting.

We formulate graph representation learning as training an encoder-decoder on node sequences generated from graphs. The most common type of encoder-decoder is a feed-forward deep neural network, but those suffer from the limitation of requiring fixed-length inputs and an inability to model sequential data. Therefore, we focus in this paper on *sequence-to-sequence encoder-decoder* architecture, which can support arbitrary-length sequences.

These encoder-decoder models are based on the sequence-to-sequence learning framework of Sutskever et al. (2014), an LSTM-based architecture in which both the inputs and outputs are sequences of variable length. The architecture uses one LSTM as the encoder $LSTM_{enc}$ and another LSTM as the decoder $LSTM_{dec}$. An input sequence $s$ with length $m$ is given to $LSTM_{enc}$ and its elements are processed one per time step. The hidden vector $h_m$ at the last time step $m$ is the fixed-length representation of the input sequence. This vector is provided as the initial vector to $LSTM_{dec}$ to generate the output sequence.

We suggested four different versions of sequence-to-sequence encoder-decoder models to investigate their ability to capture the graph structure. Three out of four encoder-decoder models adapt the sequence-to-sequence learning framework for autoencoding simply by using the same sequence for both the input and output. The autoencoders are trained so that $LSTM_{dec}$ reconstructs the input using the final hidden vector from $LSTM_{enc}$.

In our experiments, we use several graph datasets. We train a single encoder-decoder for each graph dataset. The encoder-decoder is trained on a training set of graph sequences pooled across all graphs in the dataset. After training the encoder-decoder, we obtain the representation $\Phi(G)$ for a single graph $G$ by encoding its sequences $s \in Seq(G)$ using $LSTM_{enc}$, then averaging its encoding vectors, as in Eq. (3).

### Encoder-decoder variations

**S2S-AE**: This is the standard sequence-to-sequence autoencoder inspired by (Li et al. 2015a), which we customize for embedding graphs. Figure 2 shows an overview of this model. We use $h_t^{enc}$ to denote the hidden vector at time step $t$ in $LSTM_{enc}$ and $h_t^{dec}$ to denote the hidden vector at time step $t$ in $LSTM_{dec}$. We define shorthand for Eq. 1 as follows:

$$h_t^{enc} = LSTM_{enc}\left(Emb(v_t), h_{t-1}^{enc}\right), \tag{4}$$

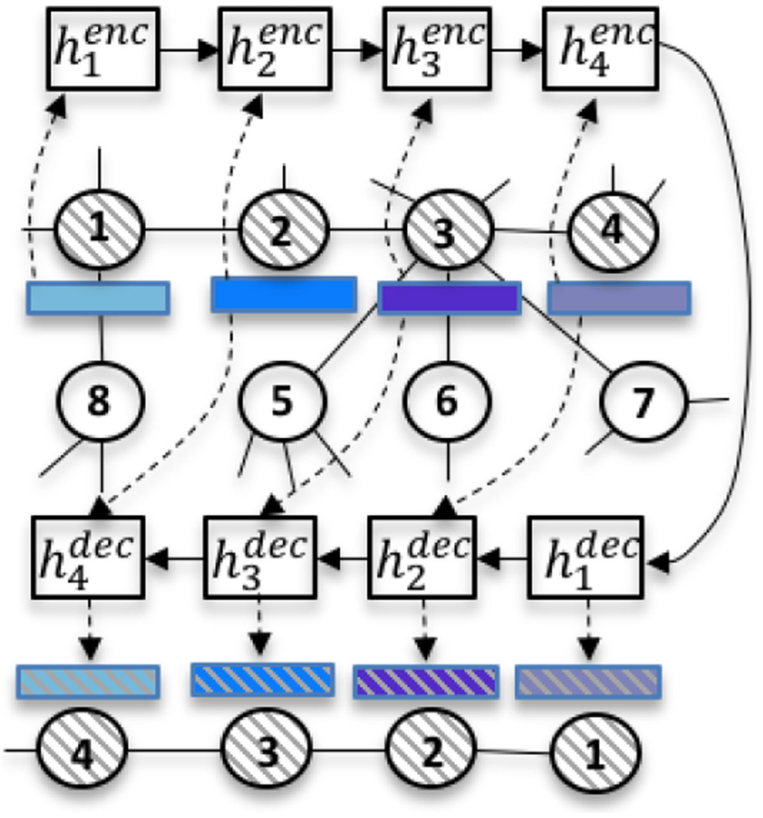

**Fig. 2** S2S-AE

where $Emb(v_t)$ takes the role of $x_t$ in Eq. 1. The hidden vector at the last time step $h_{last}^{enc} \in \mathbb{R}^m$ denotes the representation of the input sequence, and is used as the hidden vector of the decoder at its first time step:

$$h_0^{dec} = h_{last}^{enc} \tag{5}$$

The last cell vector of the encoder is copied over in an analogous way. Then each decoder hidden vector $h_t^{dec}$ is computed based on the hidden vector and the node embedding from the previous time step:

$$h_t^{dec} = LSTM_{dec}\left(Emb(v_{t-1}), h_{t-1}^{dec}\right) \tag{6}$$

The decoder uses $h_t^{dec}$ to predict the next node embedding $\overline{Emb(v_t)}$ as in Eq. 2. We have two different loss functions to test with this model. First, we consider the node embeddings fixed and compute a loss based on the difference between the predicted node embedding $\overline{Emb(v_t)}$ and the true one $Emb(v_t)$. Second, we consider a parameter vector for each embedding and update the node embeddings in addition to the model parameters using a cross entropy function. We discuss training in "Training" section. For $Emb(v_0)$, we use a vector of all zeroes.

**S2S-AE-PP**: In the previous model, $LSTM_{dec}$ predicts the embedding of the node at time step $t$ using $h_{t-1}^{dec}$ as well as $Emb(v_{t-1})$, the true node embedding at time step $t-1$. However, this may enable the decoder to rely too heavily on the previous true node in the sequence, thereby making it easier to reconstruct the input and reducing the need for the encoder to learn an effective representation of the input sequence. We consider a variation ("S2S-AE-PP: sequence-to-sequence autoencoder previous predicted") in which we use the previous predicted node $Emb(\overline{v_{t-1}})$ instead of the previous true one:

$$h_t^{dec} = LSTM_{dec}\left(Emb(\overline{v_{t-1}}), h_{t-1}^{dec}\right) \tag{7}$$

This forces the encoder and decoder to work harder to respectively encode and decode the graph sequences. This variation is related to scheduled sampling (Bengio et al. 2015), in which the training process is changed gradually from using true previous symbols to using predicted previous symbols more and more during training. The difference of S2S-AE-PP with previous model is indicated in Fig. 3.

**S2S-AE-PP-WL1,2**: This model is similar to S2S-AE-PP except that, for each node in the sequence, we use two different levels of neighborhood granularity. We incorporate the first-order neighborhoods and second-order neighborhoods (neighbors of neighbors) of nodes in learning graph representation. We use $x_{1_t}$ to denote the embedding of the label produced by one iteration of WL (for the first-order neighborhood) and $x_{2_t}$ for that produced by two iterations of WL (for second-order neighborhood). Equation 1 is modified to receive both as inputs. For example, the first line of Eq. 1 becomes:

$$i_t = \sigma(W_{1_t}x_{1_t} + W_{2_t}x_{2_t} + U_i h_{t-1} + K_i c_{t-1} + b_i) \tag{8}$$

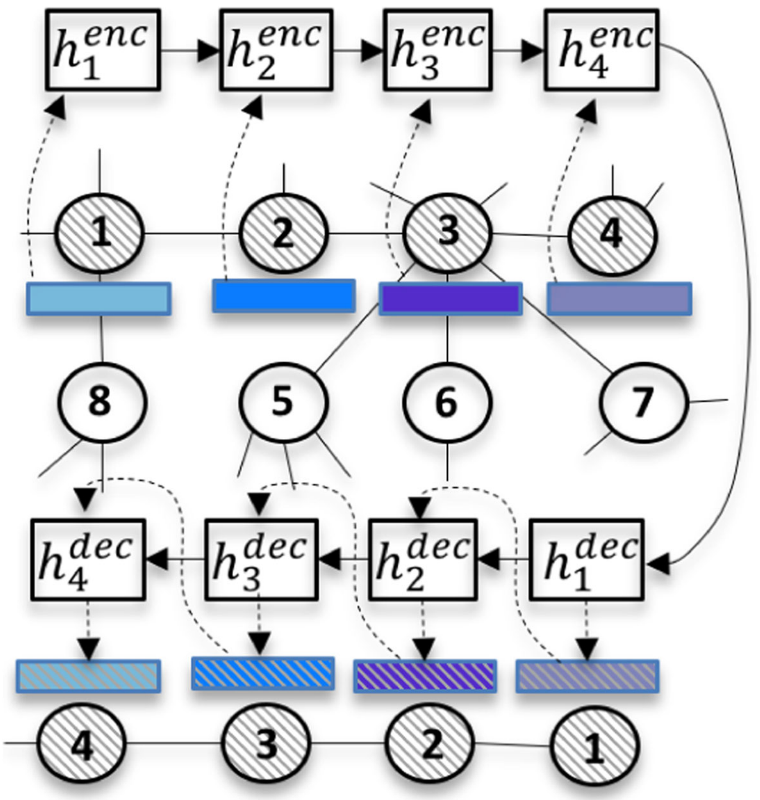

**Fig. 3** S2S-AE-PP

The other equations are changed analogously. The embeddings for both the first-order and second-order neighborhoods are learned. Figure 4 shows how this model integrates the information from neighborhoods in order to learn the graph representation.

**S2S-N2N-PP**: This model is a "neighbors-to-node" (N2N) prediction model and uses random walks as graph sequences (Fig. 5). The idea is to explore the neighborhood of a sequence by an encoder and predict the sequence itself by the decoder using the gathered information via the encoder. The encoder is encouraged to collect the information from the neighborhood which is distinguishable from other graph substructures and decoder can reconstruct the original sequence from that. That is, each item in the input sequence is the set of neighbors (their embeddings are averaged) for the corresponding node in the output sequence:

$$h_t^{enc} = LSTM_{enc}\left(\underset{v_i \in Nbrs(v_t)}{Avg}(Emb(v_i)), h_{t-1}^{enc}\right) \tag{9}$$

where $Nbrs(v)$ returns the set of neighbors of $v$ and we predict the nodes in the random walk via the decoder as in Eq. 7. Unlike the other models, this model is not an autoencoder because the input and output sequences are not the same.

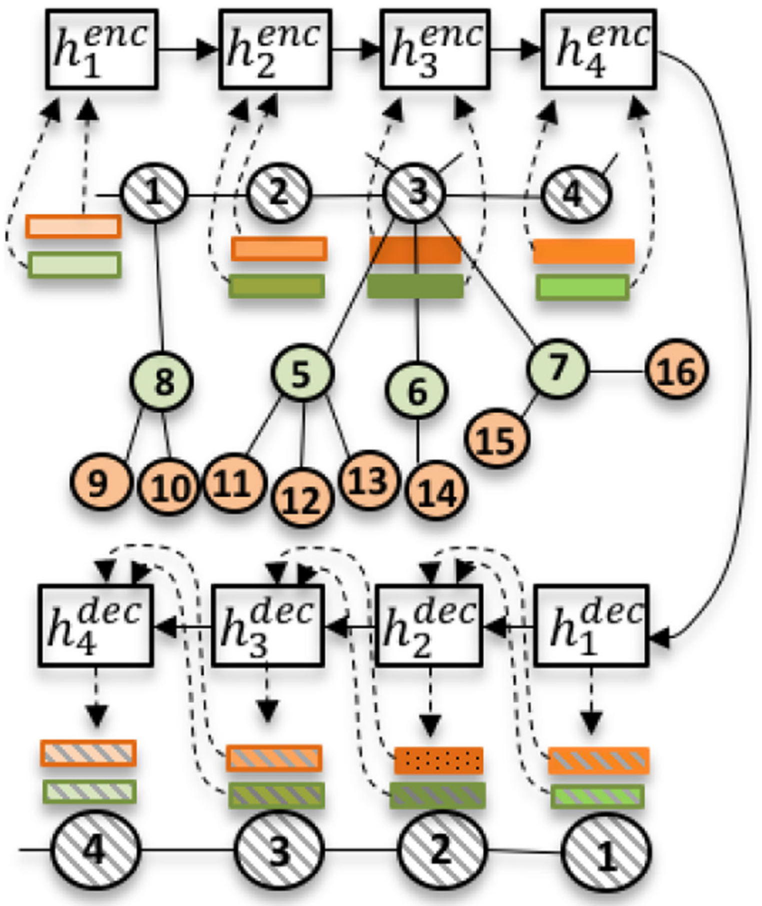

**Fig. 4** S2S-AE-PP-WL1,2

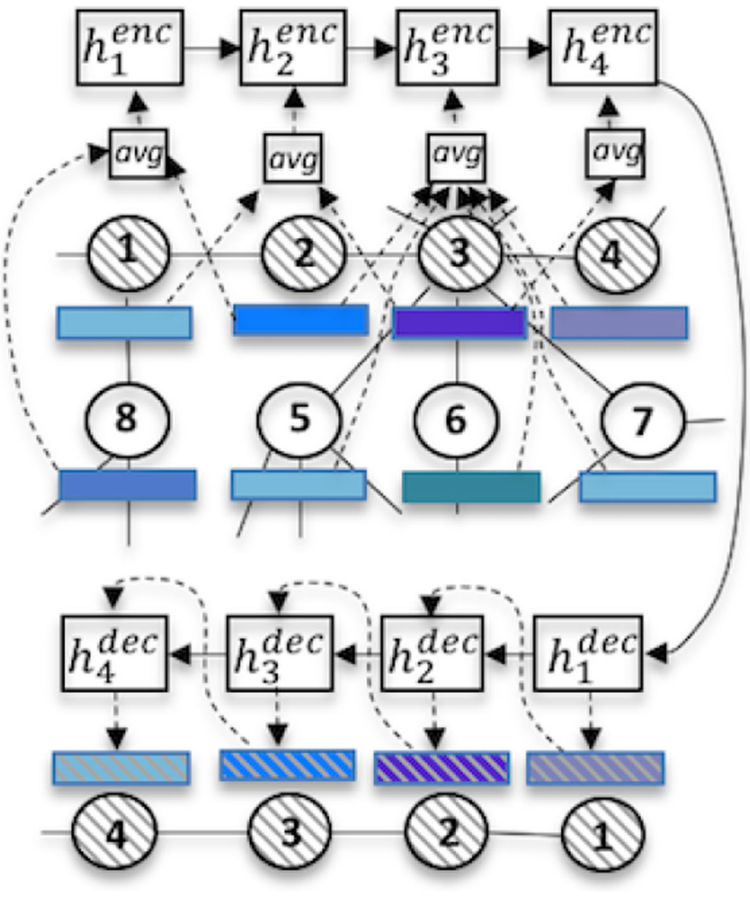

**Fig. 5** S2S-N2N-PP

## Training

Let $S$ be a sequence training set generated from a set of graphs. The representations of the sequences $s \in S$ are computed using the encoders described in the previous section. We use two different loss functions to train our models: *squared error* and *categorical cross-entropy*. The goal is to minimize the following loss functions, summed over all examples $s \in S$, where $s : v_1, \ldots, v_{|s|}$.

### *Squared error*

We used the squared error (SE) loss function for the embeddings that are fixed and are not considered as the trainable parameters of the model. We include a nonlinear transformation to estimate the embedding of the $t$th node in $s$ using the hidden vector of the decoder at time step $t$:

$$\overline{Emb(v_t)} = \text{ReLU}\left(h_t^{dec} W + b\right) \tag{10}$$

where ReLU is the rectified linear unit activation function and $W$ and $b$ are additional parameters.

Given the predicted node embeddings for the sequence $s$, the squared error loss function computes the average of the elementwise squared differences between the input and output sequences:

$$loss_{SE}(s) = \frac{1}{|s|}\sum_{t=1}^{|s|}\left\|\overline{Emb(v_t)} - Emb(v_t)\right\|_2^2 \tag{11}$$

***Categorical cross entropy***

We use the categorical cross entropy (CE) loss function for experiments in which we update the node embeddings during training. We predict the $t$th node as follows:

$$\overline{v_t} = \underset{l \in L}{\operatorname{argmax}}(h_t^{dec}.Emb(l)) \tag{12}$$

where $L$ is the set of labels. The loss computes the categorical cross entropy between the input embeddings and the predicted output embeddings:

$$loss_{CE}(s) = -\sum_{t=1}^{|s|}\log p(\overline{v_t} = l) \tag{13}$$

where $l$ denotes the true label of node $v_t$, and the predicted probability of the true label is computed as follows:

$$p(\overline{v_t} = l) = \frac{e^{h_t^{dec}.Emb(l)}}{\sum\limits_{l' \in L} e^{h_t^{dec}.Emb(l')}} \tag{14}$$

## Supervised graph representation learning

In this setting, the graph representation learning is guided by a task-specific supervision, incorporating the supervision information into the learning process. Given a dataset $D : \{G_1, ..., G_n\}$, the goal is to learn a function $GrLabel : D \rightarrow L_G$ that assigns a label from a set of labels $L_G$ to each graph in the dataset.

We introduce our supervised method inspired by the most effective unsupervised method proposed in "Sequence-to-sequence encoder-decoder" section. As we will show in "Experiments" section, that S2S-N2N-PP outperforms the other methods in almost all experiments. Therefore, we design the foundation of our supervised method based on the S2S-N2N-PP. We utilize one LSTM for processing a sequence ($LSTM_{Seq}$) and one bidrectional LSTM for processing the neighborhoods of the sequence ($BiLSTM_{Nbh}$). The neighborhoods of a sequence consist of the neighbors of nodes in the sequence. The hidden representation obtained by the $BiLSTM_{Nbh}$ from the neighborhoods is given to the $LSTM_{Seq}$ in order to incorporate it in graph representation learning. The $BiLSTM_{Nbh}$ contains an attention mechanism to select the most informative neighborhoods for the purpose of graph classification.

Recently, several approaches focused on the application of attention mechanisms in graph-structured data (Lee et al. 2018a, 2018b; Veličković et al. 2018). The attention mechanism in these methods is mainly designed to explore the neighbors of a node and focus on the most informative neighbors. However, we utilize a two-level attention mechanism to improve our classification performance by leveraging local as well as global information from the neighborhoods of nodes in a sequence. The first-level attention mechanism, $Att_{Nbr}$, attends over the neighbors of a node to capture more relevant information from its neighbors. The second-level attention module, $Att_{Nbh}$, attends over the neighborhoods along a sequence of nodes to focus on the overall more informative neighborhoods. Our approach aims to capture more globally relevant information from the graph by the $Att_{Nbh}$ in comparison with the $Att_{Nbr}$. Figure 6 shows the difference between the two levels of attention mechanism. Overall, given a sequence $s$, our approach includes two main components: (1) Neighborhood embedding to gather information from the neighborhoods of nodes in sequence $s$, and (2) Sequence embedding to represent the sequence $s$ by incorporating information from its neighborhoods. We discuss these two components in the following. Figure 7 shows an overview of the proposed approach.

## Neighborhood embedding

For each sequence $s \in S$, where $s : v_1, \ldots, v_{|s|}$, extracted in "Generating sequences from graphs" section, information from the neighbors of nodes in the sequence is gathered and passed through the $BiLSTM_{Nbh}$. An attention module $Att_{Nbr}$ is utilized in order to gather local information from the neighbors of $v_i \in s$. The $Att_{Nbr}(Nbrs(v_i))$ function decides to which nodes in $Nbrs(v_i)$ to pay attention in order to enhance their impact during training. We rely on the attention mechanism in our model to relieve the classification task from the burden of considering all the nodes in $Nbrs(v_i)$ to be equally informative. For each neighbor $n \in Nbrs(v_i)$:

$$\begin{aligned} e_n &= f(Emb(n), Emb(v_i)) \\ a_n &= \frac{\exp(e_n)}{\sum_{j \in Nbrs(v_i)} \exp(e_j)} \\ r_i &= \sum_{j \in Nbrs(v_i)} a_j Emb(j) \\ h_i^{Nbh} &= BiLSTM_{Nbh}\left(r_i, h_{i-1}^{Nbh}\right), \end{aligned} \tag{15}$$

where $f$ is a neural network that computes the attention coefficient for node $n$. The normalized coefficient, $a_n$, is obtained by a softmax function. The resulting attention readout for node $v_i$ is represented by $r_i$. The information about the neighborhood of $v_i$ is passed through the $BiLSTM_{Nbh}$. Finally, $BiLSTM_{Nbh}$ provides a hidden representation from the neighborhood of a sequence.

## Sequence embedding

After gathering information from the neighborhood of sequence $s$, the last hidden representation of the neighborhood, $h_s^{Nbh}$, is used by $LSTM_{Seq}$ to find a representation for the sequence for the purpose of graph classification. $LSTM_{Seq}$ processes the sequence whose neighborhood has been already processed by $BiLSTM_{Nbh}$:

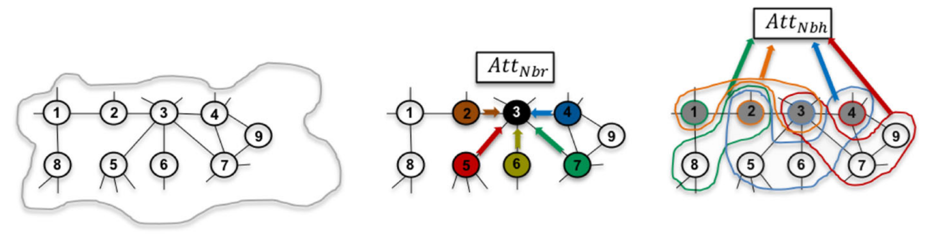

**Fig. 6** Two-level attention mechanism

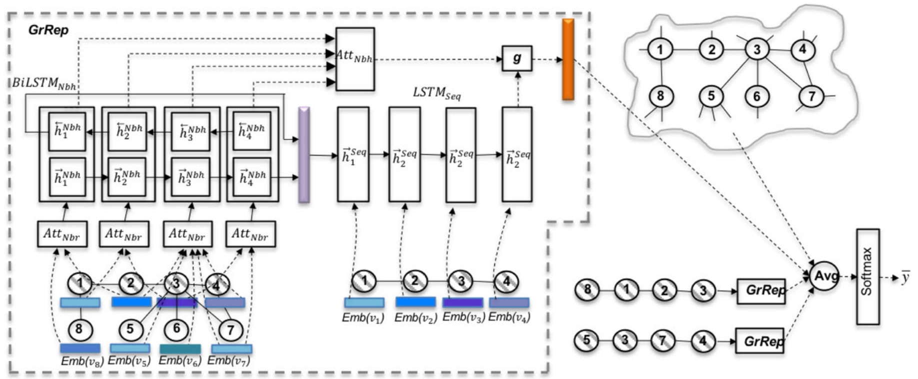
GrRep
$BiLSTM_{Nbh}$
$Att_{Nbh}$
$LSTM_{Seq}$
g
$Att_{Nbr}$
$Emb(v_8)$ $Emb(v_5)$ $Emb(v_6)$ $Emb(v_7)$
$Emb(v_1)$ $Emb(v_2)$ $Emb(v_3)$ $Emb(v_4)$
GrRep
GrRep
Avg
Softmax
$\bar{y}$

**Fig. 7** Supervised graph representation learning

$$h_0^{Seq} = h_s^{Nbh}$$
$$h_i^{Seq} = LSTM_{Seq}\left(Emb(i), h_{i-1}^{Seq}\right)$$

The second-level attention module, $Att_{Nbh}$ is applied over the output of $BiLSTM_{Nbh}$ in order to assign weights to parts of the neighborhood along sequence $s$ and reflect the importance of each part for the task-specific prediction:

$$\begin{aligned} e_i &= f\left(h_i^{Nbh}, h_s^{Seq}\right) \\ a_i &= \frac{\exp(e_i)}{\sum_{j \in s} \exp(e_j)} \\ r_{seq} &= \sum_{i \in s} a_i h_i^{Nbh} \\ rep_s &= g\left(concat\left(r_{Seq}, h_s^{Seq}\right)\right), \end{aligned} \tag{16}$$

where $f$ is a neural network that computes a single scalar as the attention coefficient for each hidden state of $BiLSTM_{Nbh}$. The attention coefficient of $Att_{Nbh}$ is represented by $a_i$. An attention mechanism on the outputs of $BiLSTM_{Nbh}$ provides the capability for the model to focus on the parts of the neighborhoods along the sequence that are influential in the end-to-end training of the model. The concatenation of sequence embedding and attention module is given to a simple feed-forward neural network, $g$, to find the subgraph representation, $rep_s$, by incorporating sequence $s$ and its neighborhood.

## Training

In order to train a scalable end-to-end model, we train the model on a set of extracted sequences from "Generating sequences from graphs" section. The model is trained so that it learns to predict the graph label using the set of chosen sequences. A batch of sequences, $b$, is selected from a graph $G_k \in D$, and the graph representation is obtained as follows:

$$rep_g = Avg_{s \in b}(rep_s) \tag{17}$$

Finally, the probability distribution over labels for $G_k$ is computed by:

$$\bar{y} = \text{softmax}(rep_g W + b), \tag{18}$$

where $W$ is a weight matrix reducing the dimension of the graph representation and $b$ is a bias. The loss function computes the categorical cross entropy between the predicted distribution $\bar{y}$ and the true label of the graph $l_{G_k} \in L_G$:

$$loss_{CE}(G_k) = -\sum_{b \in Batch(G_k)} \log p(\bar{y} = l_{G_k}) \tag{19}$$

## Experiments

In this section, we evaluate our representation learning procedure on both labeled and unlabeled graphs using our supervised and unsupervised models. We use our learned representations for the task of graph classification using several benchmark datasets and compare the accuracy of our models to state-of-the-art approaches.

### Datasets

For our classification experiments with labeled graphs, we use six datasets of bioinformatics graphs. For unlabeled graphs, we use six datasets of social network graphs.

**Labeled graphs**: The bioinformatics benchmarks include several well known datasets of labeled graphs. MUTAG (Debnath et al. 1991) is a dataset of mutagenic aromatic and heteroaromatic nitro compounds. PTC (Toivonen et al. 2003) contains several compounds classified in terms of carcinogenicity for female and male rats. Enzymes (Borgwardt et al. 2005) includes 100 proteins from each of the 6 Enzyme Commission top level enzymes classes. Proteins (Borgwardt et al. 2005) consists of graphs classified into enzymes and non-enzymes groups. Nci1 and Nci109 (Wale et al. 2008) are two balanced subsets of chemical compounds screened for activity against non-small cell lung cancer and ovarian cancer cell lines respectively.

**Unlabeled graphs**: We use several datasets developed by (Yanardag and Vishwanathan 2015) for unlabeled graph classification. COLLAB is a collaboration dataset where each network is generated from ego-networks of researchers in three research fields. Networks are classified based on research field. IMDB-BINARY and IMDB-MULTI include ego-networks for film actors/actresses from various genres on IMDB, and networks are classified by genre. Each graph in the REDDIT dataset corresponds to an online discussion thread. The REDDIT-BINARY dataset includes graphs that are extracted from four different subreddits. These subreddits may belong to a question/answer-based community or a discussion-based community. These community labels are given by the dataset and the task is to classify the graphs based on their community labels. REDDIT-MULTI-5K and REDDIT-MULTI-12K are extracted from five subreddits and eleven subreddits respectively, where the subreddit labels are given by the dataset. The task in these two datasets is to predict which subreddit a graph belongs to.

### Baselines

We compare our approach to several well known graph kernels: shortest path kernel (SPK) (Borgwardt and Kriegel 2005), random walk kernel (RWK) (Gärtner et al. 2003), graphlet kernels (GK) (Shervashidze et al. 2009), Weisfeiler-Lehman subtree kernel (WLSK) (Shervashidze et al. 2011). Also, we compare with recently proposed kernel methods: Deep Graph Kernels (DGK) (Yanardag and Vishwanathan 2015), WL-OA kernel (Kriege et al. 2016), Pyramid Match Kernel (PM) (Nikolentzos et al. 2017) and local WL label (LWL) (Morris et al. 2017). We also compare to four recent supervised methods: (PSCN)

(Niepert et al. 2016), Deep Graph Convolutional Neural Network (DGCNN) (Zhang et al. 2018), Spectral Graph Representation (SGR) (Tsitsulin et al. 2018), GCN (Kipf and Welling 2017) and the unsupervised method: Graph2vec of (Narayanan et al. 2017). We also compare to a graph representation method based on node2vec (Grover and Leskovec 2016); we use it to learn node embeddings and average them for all nodes in a graph. We report the best results from prior work on each dataset, choosing the best from multiple configurations of their methods.

## Experimental setup

For unsupervised setting, we perform 10-fold cross-validation on the graph representations of a dataset using a C-SVM classifier from LIBSVM (Chang and Lin 2011) with a radial basis kernel. Each 10-fold cross-validation experiment is repeated 10 times (with different random splits) and we report average accuracies and standard deviations. We use nested cross-validation for tuning the regularization and kernel hyperparameters of the SVM. For the supervised setting, we again perform 10-fold cross-validation on a dataset, and use the 9 training folds for training our model.

## Hyperparameter selection

We treat three labeled bioinformatics graph datasets (MUTAG, PTC, Enzymes) and two unlabeled social network datasets (IMDB-BINARY and REDDIT-BINARY) as development datasets for tuning certain high-level decisions and hyperparameters of our unsupervised approach, though we generally found results to be robust across most values. Figure 8 shows the effect of dimensionality of the graph representation, showing robustness across values larger than 50; thus, we use 100 in all experiments below. The dashed lines in Fig. 8 show accuracy when the node embeddings are fixed and the solid lines when the node embeddings are considered as vector parameters and are updated during training. We use SE ("Squared error" section) when the node embeddings are

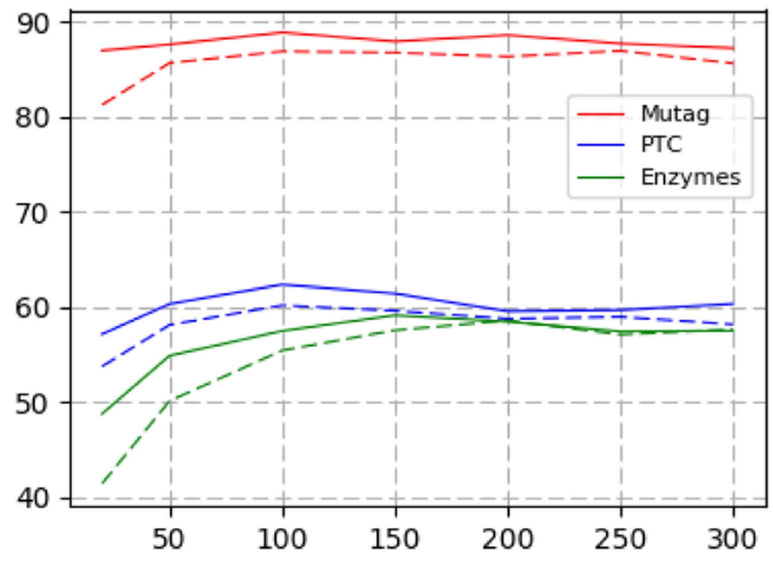

**Fig. 8** Representation dimensionality (*x* axis) vs classification accuracy (*y* axis)

fixed and CE ("Categorical cross entropy" section) when we update the node embeddings during training. CE consistently outperforms SE and we use CE for all remaining experiments. By using CE, we learn representations of neighborhoods at the right order of granularity and, using those, we learn the representation of the entire graph. We use AdaGrad (Duchi et al. 2011) with learning rate 0.01 and mini-batch size 100.

In the experiments that used BFS for sequence generation, we only consider nodes that are at most 1 edge from the starting node. In some cases, this still leads to extremely long sequences for nodes with many neighbors. We convert these to multiple sequences so that each has a maximum length of 10. When doing so, we still prepend the initial starting node to each of the truncated partial sequences.

When using random walks, we generate multiple random walks from each node. We compared random walk lengths of $\{3, 5, 10, 15, 20\}$. Figure 9 shows robust performance with length 5 across datasets and we use this length below.

## Comparing models, type, and granularity of sequences

Figures 10, 11 and 12 show the results of the classification task on the development labeled graphs for our unsupervised models, with varying types of substructures (RW, BFS, SP) and their granularity (WL0, WL1, WL2). WL0 shows the 0-order granularity in which we use the original label of the nodes. WL1 and WL2 show sequences of first order and second order neighborhoods of the nodes, respectively.

Figure 10 shows the accuracy of the graph classification task with different orders of sequence granularity. We show the average of accuracies of the two autoencoders S2S-AE and S2S-AE-PP (rather than all four, since S2S-N2N-PP does not have the full range of substructures and S2S-AE-PP-WL1,2, uses both WL labels together, thus, not comparable). The results indicate that using sequences of neighborhoods improves accuracy substantially compared to using sequences of nodes. There is a large gap between accuracies generated by the first order neighborhoods and original labels (0-order neighborhood) across all sequence types. This provides strong evidence that sequences of

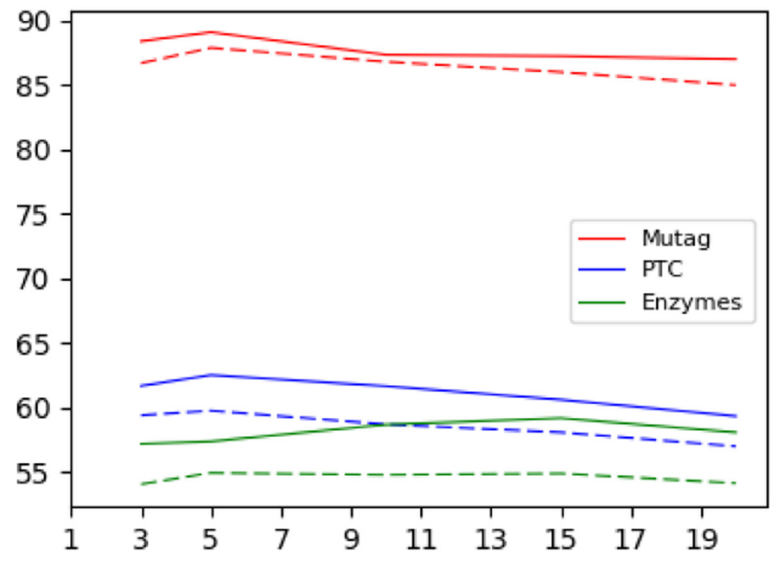

**Fig. 9** Random walk length (*x* axis) vs graph classification accuracy (*y* axis)

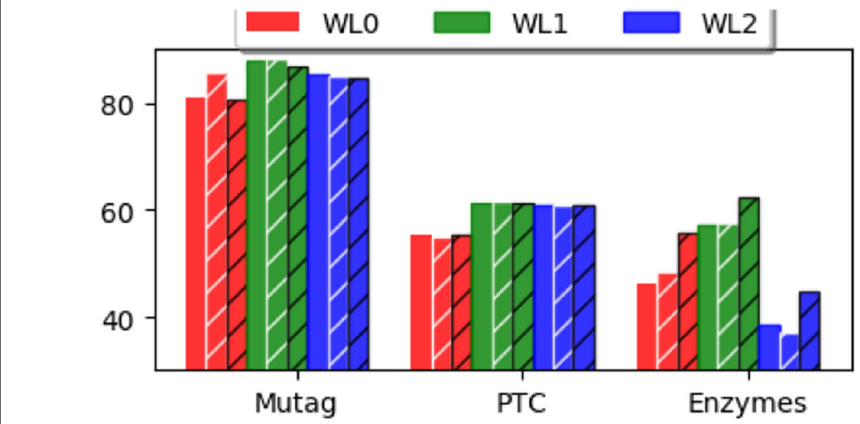

**Fig. 10** Substructure granularity (solid: RW, white hatch: BFS, black hatch:SP)

first order neighborhoods can enrich the original labels in a way that improves graph representations by capturing additional local node structure.

The number of unique substructures increases from first order neighborhoods to second order neighborhoods. Thus, the number of WL labels assigned to neighborhoods increases with each iteration. Although distinctive labels provide more information about the local structure of nodes, using WL labels with higher iterations does not necessarily lead to better graph representations. For example, the accuracy of Enzymes (Fig. 10) shows a significant drop when using the second iteration of the WL algorithm. We believe that the reason of such a sharp drop in this dataset can be explained by the graphs' label entropy (Li et al. 2011). Given a graph $G$ and a set of labels $L$, the label entropy of $G$ is defined as $H(G) = -\sum_{l \in L} p(l) \log p(l)$. The average label entropy in each dataset is depicted in Table 1. The entropy of the Enzymes dataset increases more than MUTAG and PTC from the substructures with first order neighborhoods to the second order neighborhoods. As the entropy increases, the number of unique WL labels in a dataset, and consequently the impurity of the set of labels, increases. When the number of *common* labels shared by different graphs decreases, the model cannot learn the similarity between graphs because each graph is represented by a set of labels that are nearly unique

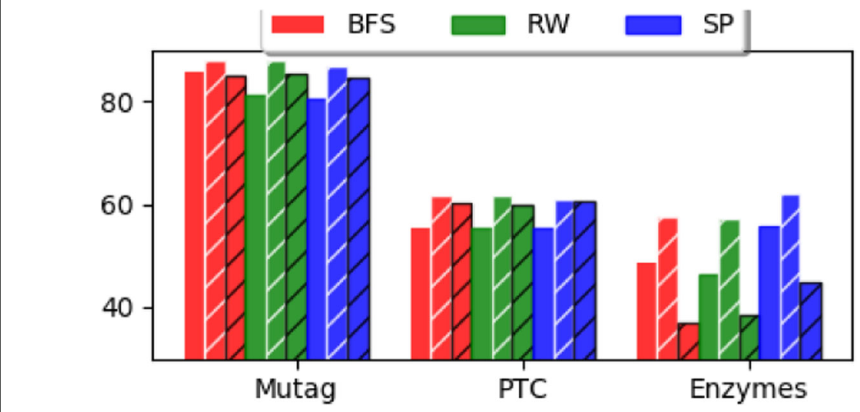

**Fig. 11** Type of substructure (solid: WL0, white hatch: WL1, black hatch:WL2)

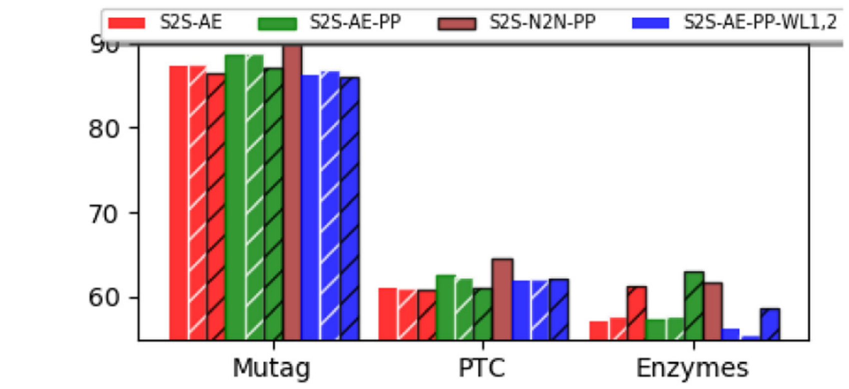

**Fig. 12** Models (solid: RW, white hatch: BFS, black hatch:SP)

to it. However, the model can detect the commonality among the topology of graphs (similar substructures) when they have more shared labels.

Moreover, Fig. 11 shows that the accuracies of classification using different types of sequences are very close to each other when using the first order and second order neighborhoods. The difference between the types of sequences is more evident when we use the original labels (0-order neighborhood sequences of nodes). In Enzymes, using shortest paths clearly results in better accuracies than random walks and BFS, regardless of the sequence granularity. For the same type of graphs, Borgwardt and Kriegel (2005) similarly observed that their shortest path kernel was better than walk-based kernels. We suspect

**Table 1** Classification accuracies for labeled graph datasets

| Datasets | MUTAG | PTC | Enzymes | fig1 | Nci1 | Nci109 | |
|---|---|---|---|---|---|---|---|
| # Graphs | 188 | 344 | 600 | 1113 | 4110 | 4127 | |
| EntropyWL1 | 2.72 | 4.03 | 5.45 | 5.54 | 4.21 | 4.22 | |
| EntropyWL2 | 4.65 | 7.09 | 12.60 | 13.26 | 8.23 | 8.25 | |
| ClusteringCoef | 0 | 0.0025 | 0.453 | 0.51 | 0.003 | 0.003 | |
| Avg. Nodes | 17.9 | 25.5 | 32.6 | 39.1 | 29.8 | 29.6 | |
| # Labels | 7 | 19 | 2 | 3 | 37 | 38 | |
| # Classes | 2 | 2 | 6 | 2 | 2 | 2 | Avg Rank |
| SPK (BK05) | 85.2 ±2.4 | 58.2 ± 2.4 | 40.1 ± 1.5 | 75.1 ± 0.5 | 73.0 ± 0.2 | 73.0 ± 0.2 | 7.5 |
| RWK (G+03) | 83.7 ±1.5 | 57.8 ±1.3 | 24.2 ±1.6 | 74.2 ±0.4 | — | — | — |
| GK (S+09) | 81.7 ± 2.1 | 57.2 ± 1.4 | 26.6 ±0.9 | 71.7 ±0.5 | 62.3 ± 0.2 | 62.6 ± 0.1 | 8.5 |
| WLSK (S+11) | 80.7 ±3.0 | 57.0 ±2.0 | 53.1 ±1.1 | 72.9 ±0.5 | 80.1 ±0.5 | 80.2 ±0.3 | 8.5 |
| node2vec (G+16) | 82.01±1.0 | 55.60±1.4 | 19.42±2.3 | 70.76±1.2 | 61.91±0.3 | 61.53±0.9 | 9.5 |
| DGK (YW15) | 87.4 ±2.7 | 60.1 ±2.5 | 53.4 ±0.9 | 75.7 ±0.5 | 80.3 ±0.4 | 80.3 ±0.3 | 5.6 |
| PSCN (N+16) | **92.6** | 62.3 | — | 75.9 | 78.6 | — | — |
| WL-OA (K+16) | 86.0 ±1.7 | 63.6 ±1.5 | 59.9 ±1.1 | 76.4 ±0.4 | 86.1 ±0.2 | 86.3±0.2 | 3.3 |
| GCN (KI+17) | 86.3±2.1 | 63.28±3.9 | 56.6±3.5 | 75.9±2.8 | 81.1±1.6 | 80.7±1.8 | 5.0 |
| graph2vec (N+17) | 83.15 ±9.2 | 60.17 ±6.9 | — | 73.30 ±2.0 | 73.22 ±1.9 | 74.26 ±1.5 | — |
| WL PM (NI+17) | 87.77 ±0.8 | 61.41±0.8 | 55.55±0.5 | — | 86.40 ±0.2 | 85.34±0.2 | — |
| LWL (M+17) | 85.2 ±1.6 | 64.7±0.2 | 61.8±1.2 | 76.4 ±0.7 | 83.1 ±0.2 | 82.0±0.3 | 3.0 |
| DGCNN (Z+18) | 85.83 ±1.6 | 58.59±2.4 | — | 75.54±0.9 | 74.44±0.4 | — | — |
| SGR (T+18) | 86.97 | — | 33.67 | 73.83 | — | — | — |
| S2S-N2N-PP | 89.86±1.1 | 64.54±**1.1** | **63.96±0.6** | 76.51±0.5 | 83.72±0.4 | 83.64±0.3 | **2.5** |
| supervised | 89.91±1.5 | **65.72±1.2** | 57.48±0.8 | **77.28±0.9** | **86.65±0.6** | **86.61±0.5** | **1.5** |

**Table 2** Classification accuracies for unlabeled graph datasets

| Datasets | COLLAB | IMDB- BINARY | IMDB- MULTI | REDDIT- BINARY | REDDIT- MULTI-5k | REDDIT- MULTI-12k | |
|---|---|---|---|---|---|---|---|
| # Graphs | 5000 | 1000 | 1500 | 2000 | 5000 | 11929 | |
| Avg. Nodes | 74.49 | 19.73 | 13.00 | 429.61 | 508.50 | 391.40 | |
| # Classes | 3 | 2 | 3 | 2 | 5 | 11 | AVG Rank |
| node2vec (G+16) | 56.06$_{\pm0.2}$ | 50.17$_{\pm0.9}$ | 36.02$_{\pm0.7}$ | 71.31$_{\pm2.2}$ | 33.11$_{\pm1.7}$ | 23.62$_{\pm0.3}$ | 6.0 |
| DGK (YW15) | 73.0$_{\pm0.2}$ | 66.9$_{\pm0.5}$ | 44.5$_{\pm0.5}$ | 78.0$_{\pm0.3}$ | 41.2$_{\pm0.1}$ | 32.2$_{\pm0.1}$ | 4.5 |
| PSCN (N+16) | 72.6$_{\pm2.1}$ | 71.0$_{\pm2.2}$ | 45.2$_{\pm2.8}$ | 86.3$_{\pm1.5}$ | 49.1$_{\pm0.7}$ | 41.3$_{\pm0.4}$ | 3.6 |
| WL-OA (K+16) | 80.7$_{\pm0.1}$ | — | — | 89.3$_{\pm0.3}$ | — | — | — |
| GCN (KI+17) | 80.17$_{\pm1.6}$ | 73.6$_{\pm3.4}$ | 51.3$_{\pm3.8}$ | 72.8 $_{\pm2.5}$ | 40.6$_{\pm1.8}$ | 32.4$_{\pm1.9}$ | 3.6 |
| LWL (M+17) | — | 73.5$_{\pm0.5}$ | — | 81.3$_{\pm0.5}$ | — | — | — |
| DGCNN (Z+18) | 73.76$_{\pm0.49}$ | 70.03$_{\pm0.86}$ | 47.83$_{\pm0.85}$ | — | — | — | — |
| SGR (T+18) | 71.98 | 70.38 | 47.97 | 87.45 | 53.22 | — | — |
| S2S-N2N-PP | 81.75$_{\pm0.8}$ | 73.8$_{\pm0.7}$ | 51.19$_{\pm0.5}$ | 86.50$_{\pm0.8}$ | 52.28$_{\pm0.5}$ | 42.47$_{\pm0.1}$ | 2.1 |
| supervised | **82.32$_{\pm0.8}$** | **74.9$_{\pm0.8}$** | **53.26$_{\pm0.7}$** | 89.10$_{\pm0.8}$ | **53.38$_{\pm0.6}$** | **43.57**$_{\pm0.6}$ | 1.0 |

that the reason is related to the clustering coefficient, a popular metric in network analysis. The clustering coefficient is the fraction of triangles connected to node *v* over the total number of connected triples centered on node *v*. Having many triangles in the ego-network of node *v* may cause the tottering problem in a walk traversing node *v* and may generate less discriminative BFS sequences from that ego-network. Shortest paths prevent tottering and capture the global graph structure. BFS sequences mainly consider the local topological features of a graph, and random walks collect a representative sample of nodes rather than of topology (Kurant et al. 2011). Fortunately, using the sequence of substructures augmented with neighborhoods can reduce the effect of sequence type in most settings.

Figure 12 shows the comparison between different unsupervised models, using the sequences of first order neighborhoods. Model S2S-AE-PP is better than Model S2S-AE in nearly all cases. As we conjectured above, Model S2S-AE-PP may force the encoder representation to capture the entire sequence since the decoder has less assistance during reconstruction. Model S2S-N2N-PP obtains higher accuracy in almost all datasets, showing the benefit of capturing local neighborhoods With S2S-AE-PP-WL1,2, we only observe improvements over the other S2S-AE models on the PTC dataset. This suggests that adding substructures from both of the first order neighborhoods and second order neighborhoods could not provide more informative graph representations, regardless of the type of substructures. The reason could be due to the fact that in this model we add too many vector parameters for each neighborhood substructure and our model is not able to learn the meaningful embeddings for these substructures, which leads to poor graph representation.

## Comparison to state-of-the-art

We compare S2S-N2N-PP and our supervised model to the state-of-the-art in Table 1. We exceed all prior results except on MUTAG dataset. Our supervised method outperforms other supervised and unsupervised graph representation learning approaches. However, S2S-N2N-PP in combination with C-SVM performs well in graph classification. Considering that none of the previous work can outperform all others in all datasets, we suggested average ranking measure to compare the performance of all the approaches together. Our supervised approach shows robustness, achieving the first ranking among other methods. S2S-N2N-PP obtains the second ranking and this is a strong evidence

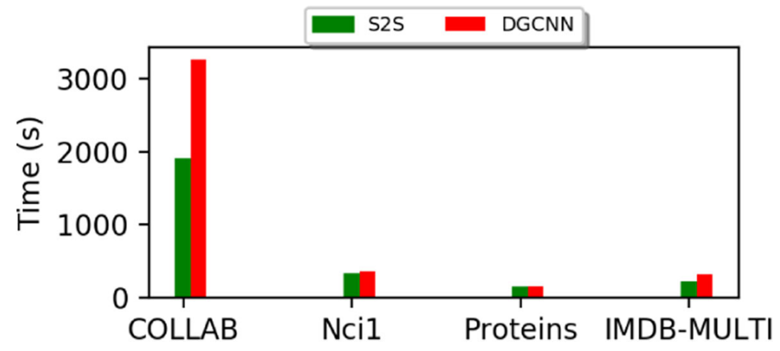

**Fig. 13** Training time of graph embedding methods

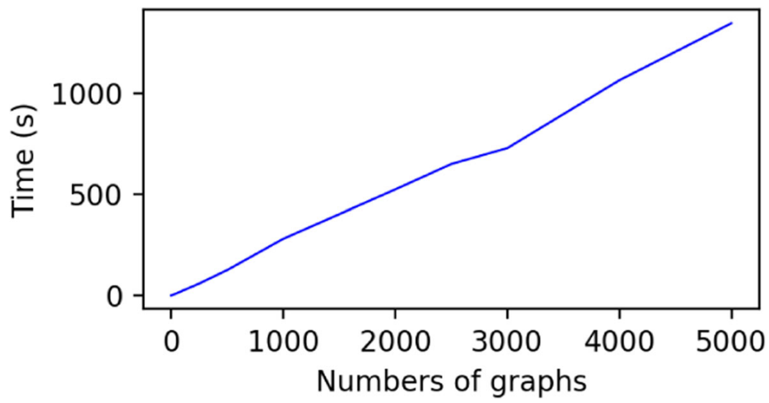

**Fig. 14** Growth of training time versus number of graphs

that our unsupervised method learns the graph structure effectively. The third and fourth rankings are obtained by LWL and WL-OA, which are kernel methods and suffer from the quadratic complexity of the growth of the running time with the number of graphs in the dataset. Table 2 compares our method to prior work on the unlabeled graph datasets. Our approach established new state-of-the-art accuracies on all datasets except REDDIT-BINARY.

Figure 13 shows the duration of training of the graph representation learning in our S2S-N2N-PP approach versus the DGCNN. The parameters of DGCNN are configured according to their paper (Zhang et al. 2018). DGCNN requires more time to learn the graph representation for the purpose of classification in all datasets. However, the difference between the two methods is more obvious in the COLLAB dataset, which includes more graphs and the number of nodes for each graph is more than that of the other datasets. Moreover, Fig. 14 shows the linear growth of training time of our representation learning approach with the increase in the number of graphs in a dataset. We report the training time of our approach by changing the size of a dataset incrementally. The dataset includes a set of randomly selected graphs from the COLLAB dataset.

## Conclusions

We proposed sequence-to-sequence LSTM architectures for learning representations of graphs in both supervised and unsupervised regimes. We trained our models using sequences from different types of substructures (random walks, shortest paths, and breadth-first search) with various levels of granularity (neighborhoods of increasing order). Our experiments demonstrate that our graph representations can increase the accuracy of graph classification tasks on both supervised and unsupervised approaches, achieving to our knowledge, the best results on several datasets considered.

### Abbreviations

BFS: Breadth-first search; CE: Cross entropy DGCNN: Deep graph convolutional neural network; DGK: Deep graph kernel; GAM: Graph attention model; GNNs: Graph neural networks; LSTM: Long short-term memory; LWL: Local WL label; PM: Pyramid match kernel; RNN: Recurrent neural network; RW: Random walk; SP: Shortest path; WL: Weisfeiler-Lehman; WL-OA: Optimal-assignment Weisfeiler Lehman

**Acknowledgements**
Not applicable.

**Authors' contributions**
AT defined, formalized and implemented the approach under the supervision of TBW. KG contributed ideas in the unsupervised setting and the overall preparation of the manuscript. All authors read and approved the final manuscript.

**Funding**
This research was supported in part by the following grants: NSF IIS-1515587 and NSF III-1514126

**Availability of data and materials**
The datasets are selected from the related work mentioned in the experiment section.

**Competing interests**
The authors declare that they have no competing interests.

**Author details**
[1]University of Illinois at Chicago, Chicago, USA. [2]Toyota Technological Institute at Chicago, Chicago, USA.

Received: 15 March 2019 Accepted: 22 July 2019
Published online: 24 August 2019

**References**
Adhikari B, Zhang Y, Ramakrishnan N, Prakash BA (2017) Distributed representations of subgraphs. In: DaMNet
Akoglu L, McGlohon M, Faloutsos C (2010) Oddball: Spotting anomalies in weighted graphs. In: PAKDD
Bengio S, Vinyals O, Jaitly N, Shazeer N (2015) Scheduled sampling for sequence prediction with recurrent neural networks. In: NIPS
Berlingerio M, Koutra D, Eliassi-Rad T, Faloutsos C (2012) NetSimile: a scalable approach to size-independent network similarity. arXiv
Borgwardt KM, Kriegel H-P (2005) Shortest-path kernels on graphs. In: ICDM
Borgwardt K, Ong C, Schönauer S, Vishwanathan S, Smola A, Kriegel H (2005) Protein function prediction via graph kernels. Bioinformatics 21
Bruna J, Zaremba W, Szlam A, LeCun Y (2013) Spectral networks and locally connected networks on graphs. CoRR
Bunke H (2000) Graph matching: Theoretical foundations, algorithms, and applications. In: Vision Interface
Chang C-C, Lin C-J (2011) Libsvm: a library for support vector machines. ACM TIST 2
Chen J, Xu X, Wu Y, Zheng H (2018) Gc-lstm: Graph convolution embedded lstm for dynamic link prediction. arXiv preprint arXiv:1812.04206
Debnath A, Lopez de Compadre R, Debnath G, Shusterman A, Hansch C (1991) Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. J Med Chem
Defferrard M, Bresson X, Vandergheynst P (2016) Convolutional neural networks on graphs with fast localized spectral filtering. arXiv
Duchi J, Hazan E, Singer Y (2011) Adaptive subgradient methods for online learning and stochastic optimization. JMLR
Duvenaud D, Maclaurin D, Iparraguirre J, Bombarell R, Hirzel T, Aspuru-Guzik A, Adams RP (2015) Convolutional networks on graphs for learning molecular fingerprints. In: NIPS
Floyd RW (1962) Algorithm 97: shortest path. Commun ACM
García-Durán A, Niepert M (2017) Learning graph representations with embedding propagation. In: NIPS
Gärtner T, Flach P, Wrobel S (2003) On graph kernels: Hardness results and efficient alternatives. In: COLT
Gilmer J, Schoenholz SS, Riley PF, Vinyals O, Dahl GE (2017) Neural message passing for quantum chemistry. CoRR
Grover A, Leskovec J (2016) node2vec: Scalable feature learning for networks. In: KDD
Haussler D (1999) Convolution kernels on discrete structures. Technical report
Henaff M, Bruna J, LeCun Y (2015) Deep convolutional networks on graph-structured data. arXiv
Hinton GE, Zemel RS (1993) Autoencoders, minimum description length, and helmholtz free energy. In: NIPS
Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Comput
Kipf TN, Welling M (2017) Semi-supervised classification with graph convolutional networks. In: ICLR
Kriege NM, Giscard P-L, Wilson R (2016) On valid optimal assignment kernels and applications to graph classification. In: NIPS
Kurant M, Markopoulou A, Thiran P (2011) Towards unbiased bfs sampling. IEEE J Sel Areas Commun
Lee JB, Rossi RA, Kim S, Ahmed NK, Koh E (2018a) Attention models in graphs: A survey. arXiv preprint arXiv:1807.07984
Lee JB, Rossi R, Kong X (2018b) Graph classification using structural attention. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. ACM. pp 1666–1674
Li J, Luong M, Jurafsky D (2015a) A hierarchical neural autoencoder for paragraphs and documents. In: ACL
Li G, Semerci M, Yener B, Zaki MJ (2011) Graph classification via topological and label attributes. In: MLG
Li Y, Tarlow D, Brockschmidt M, Zemel R (2015b) Gated graph sequence neural networks. arXiv
Maaten Lvd, Hinton G (2008) Visualizing data using t-SNE. JMLR
Macindoe O, Richards W (2010) Graph comparison using fine structure analysis. In: SocialCom
Morris C, Kersting K, Mutzel P (2017) Glocalized weisfeiler-lehman graph kernels: Global-local feature maps of graphs. In: ICDM
Narayanan A, Chandramohan M, Chen L, Liu Y, Saminathan S (2016) subgraph2vec: Learning distributed representations of rooted sub-graphs from large graphs. MLG
Narayanan A, Chandramohan M, Venkatesan R, Chen L, Liu Y, Jaiswal S (2017) graph2vec: Learning distributed representations of graphs. In: MLG
Newman ME (2003) The structure and function of complex networks. SIAM Rev
Niepert M, Ahmed M, Kutzkov K (2016) Learning convolutional neural networks for graphs. In: ICML
Nikolentzos G, Meladianos P, Vazirgiannis M (2017) Matching node embeddings for graph similarity. In: AAAI

Perozzi B, Al-Rfou R, Skiena S (2014) DeepWalk: Online learning of social representations. In: KDD
Riesen K, Jiang X, Bunke H (2010) Exact and inexact graph matching: Methodology and applications. In: Managing and Mining Graph Data
Rossi RA, Zhou R, Ahmed N (2018) Deep inductive graph representation learning. IEEE Trans Knowl Data Eng
Scarselli F, Gori M, Tsoi C, Hagenbuchner M, Monfardini G (2009) The graph neural network model. IEEE Trans Neural Netw 20
Shervashidze N, Schweitzer P, Leeuwen EJv, Mehlhorn K, Borgwardt KM (2011) Weisfeiler-Lehman graph kernels. JMLR
Shervashidze N, Vishwanathan S, Petri T, Mehlhorn K, Borgwardt KM (2009) Efficient graphlet kernels for large graph comparison. In: AISTATS
Sutskever I, Vinyals O, Le QV (2014) Sequence to sequence learning with neural networks. In: NIPS
Tang J, Qu M, Wang M, Zhang M, Yan J, Mei Q (2015) Line: Large-scale information network embedding. In: WWW
Toivonen H, Srinivasan A, King R, Kramer S, Helma C (2003) Statistical evaluation of the predictive toxicology challenge. Bioinformatics 19
Tsitsulin A, Mottin D, Karras P, Bronstein A, Müller E (2018) Sgr: Self-supervised spectral graph representation learning. arXiv preprint arXiv:1811.06237
Trivedi R, Dai H, Wang Y, Song L (2017) Know-evolve: Deep temporal reasoning for dynamic knowledge graphs. In: ICML
Van Wijk BC, Stam CJ, Daffertshofer A (2010) Comparing brain networks of different size and connectivity density using graph theory. PLoS ONE 5
Veličković P, Cucurull G, Casanova A, Romero A, Lio P, Bengio Y (2018) Graph attention networks. In: ICLR
Vishwanathan S, Schraudolph N, Kondor R, Borgwardt K (2010) Graph kernels. JMLR
Wale N, Watson IA, Karypis G (2008) Comparison of descriptor spaces for chemical compound retrieval and classification. KAIS 14
Weisfeiler B, Lehman A (1968) A reduction of a graph to a canonical form and an algebra arising during this reduction. Nauchno-Technicheskaya Informatsia
Yan X, Han J (2002) gspan: Graph-based substructure pattern mining. In: ICDM
Yanardag P, Vishwanathan S (2015) Deep graph kernels. In: KDD
Ying Z, You J, Morris C, Ren X, Hamilton W, Leskovec J (2018) Hierarchical graph representation learning with differentiable pooling. In: Advances in 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: AAAI

## Publisher's Note