A UNIVERSAL SELF-ATTENTION GRAPH NEURAL NETWORK

Anonymous authors
Paper under double-blind review

ABSTRACT

Existing graph embedding models often have weaknesses in exploiting graph structure similarities, potential dependencies among nodes and global network properties. To this end, we present U2GNN, a novel embedding model leveraging on the strength of the recently introduced universal self-attention network (Dehghani et al., 2019), to learn low-dimensional embeddings of graphs which can be used for graph classification. In particular, given an input graph, U2GNN first applies a self-attention computation, which is then followed by a recurrent transition to iteratively memorize its attention on vector representations of each node and its neighbors across each iteration. Thus, U2GNN can address the weaknesses in the existing models in order to produce plausible node embeddings whose sum is the final embedding of the whole graph. Experimental results in both supervised and unsupervised fashions show that our U2GNN produces new state-of-the-art performances on a range of well-known benchmark datasets for the graph classification task.

1 INTRODUCTION

Many real-world and scientific data are represented in forms of graphs, e.g. data from knowledge graphs, recommender systems, social and citation networks as well as telecommunication and biological networks (Battaglia et al., 2018; Zhang et al., 2018c). In general, a graph can be viewed as a network of nodes and edges, where nodes correspond to individual objects and edges encode relationships among those objects. For example, in online forum, each discussion thread can be constructed as a graph where nodes represent users and edges represent commenting activities between users (Yanardag & Vishwanathan, 2015).

Early approaches focus on computing the similarities among graphs to build a graph kernel for graph classification (Gärtner et al., 2003; Kashima et al., 2003; Borgwardt & Kriegel, 2005; Shervashidze et al., 2009; Vishwanathan et al., 2010; Shervashidze et al., 2011; Yanardag & Vishwanathan, 2015; Narayanan et al., 2017; Ivanov & Burnaev, 2018). These graph kernel-based approaches treat each atomic substructure (e.g., subtree structure, random walk or shortest path) as an individual feature, and count their frequencies to construct a numerical vector to represent the entire graph, hence they ignore *node attributes*, substructure similarities and global network properties.

One recent notable strand is to learn low-dimensional continuous embeddings of the whole graphs (Hamilton et al., 2017b; Zhang et al., 2018a; Zhou et al., 2018), and then use these learned embeddings to train a classifier to predict graph labels (Wu et al., 2019). Advanced approaches in this direction have attempted to exploit graph neural network (Scarselli et al., 2009), capsule network (Sabour et al., 2017) or graph convolutional neural network (Kipf & Welling, 2017; Hamilton et al., 2017a) for supervised learning objectives (Li et al., 2016; Niepert et al., 2016; Zhang et al., 2018b; Ying et al., 2018; Verma & Zhang, 2018; Xu et al., 2019; Xinyi & Chen, 2019; Maron et al., 2019b; Chen et al., 2019). These graph neural network (GNN)-based approaches usually consist of two common phases: the propagating phase and the readout phase. The former phase aims to iteratively update vector representation of each node by recursively aggregating representations of its neighbors, and then the latter phase applies a pooling function (e.g., mean, max or sum pooling) on output node representations to produce an embedding of each entire graph; and this graph embedding is used to predict the graph label. We find that these approaches are currently showing very promising performances, nonetheless the dependency aspect among nodes, which often exhibit strongly

in many kinds of real-world networks, *has not been exploited effectively due to the straightforward computations within the propagating phase.*

Very recently, the universal self-attention network (Dehghani et al., 2019) has been shown to be very powerful in NLP tasks such as question answering, machine translation and language modeling. Inspired by this new attention network, we propose U2GNN – a novel universal self-attention graph neural network to learn plausible node and graph embeddings. Our intuition comes from an observation that the recurrent attention process in the universal self-attention network can memorize implicit dependencies between each node and its neighbors from previous iterations, which can be then aggregated to further capture the dependencies among substructures into latent representations in subsequent iterations; this process, hence, can capture both local and global graph structures. Algorithmically, at each timestep, our proposed U2GNN iteratively exchanges a node representation with its neighborhood representations using a self-attention mechanism (Vaswani et al., 2017) followed by a recurrent transition to infer node embeddings. We finally take the sum of all learned node embeddings to obtain the embedding of the whole graph. Our main contributions are as follows:

- In our proposed U2GNN, the novelty of memorizing the dependencies among nodes implies that U2GNN can explore the graph structure similarities locally and globally – an important feature that most of existing approaches are unable to do.
- U2GNN can be seen as a general framework where we prove the powerfulness of our model in both supervised or unsupervised fashions. The experimental results on 9 well-known benchmark datasets for the graph classification task show that both our supervised and unsupervised U2GNN models produce new state-of-the-art (SOTA) accuracies in most of benchmark cases.
- To our best of knowledge, our work is the first to show that a unsupervised model can noticeably outperform up-to-date supervised approaches by a large margin. Therefore, we suggest that future GNN works should pay more attention to the unsupervised fashion as well as not comparing supervised models with unsupervised models together. This is important in both industry and academic applications in reality where expanding unsupervised GNN models is more suitable due to the limited availability of class labels.

2 RELATED WORK

Early popular approaches are based on "graph kernel" which aims to recursively decompose each graph into "atomic substructures" (e.g., graphlets, subtree structures, random walks or shortest paths) in order to measure the similarity between two graphs (Gärtner et al., 2003). For this reason, we can view each atomic substructure as a word token and each graph as a text document, hence we represent a collection of graphs as a document-term matrix which describes the normalized frequency of terms in documents. Then, we can use a dot product to compute the similarities among graphs to derive a kernel matrix used to measure the classification performance using a kernel-based learning algorithm such as Support Vector Machines (SVM) (Hofmann et al., 2008). We refer to an overview of the graph kernel-based approaches in (Nikolentzos et al., 2019; Kriege et al., 2019).

Since the introduction of word embedding models i.e., Word2Vec (Mikolov et al., 2013) and Doc2Vec (Le & Mikolov, 2014), there have been several efforts attempted to apply them for the graph classification task. Deep Graph Kernel (DGK) (Yanardag & Vishwanathan, 2015) applies Word2Vec to learn embeddings of atomic substructures to create the kernel matrix. Graph2Vec (Narayanan et al., 2017) employs Doc2Vec to obtain embeddings of entire graphs in order to train a SVM classifier to perform classification. Anonymous Walk Embedding (AWE) (Ivanov & Burnaev, 2018) maps random walks into "anonymous walks" which are considered as word tokens, and then utilizes Doc2Vec to achieve the graph embeddings to produce the kernel matrix.

In parallel, another recent line of work has focused on using deep neural networks to perform the graph classification in a supervised manner. PATCHY-SAN (Niepert et al., 2016) adapts a graph labeling procedure to generate a fixed-length sequence of nodes from an input graph, and orders k-hop neighbors for each node in the generated sequence according to their graph labelings; PATCHY-SAN then selects a fixed number of ordered neighbors for each node and applies a convolutional neural network to classify the input graph. MPNN (Gilmer et al., 2017), DGCNN (Zhang et al., 2018b) and DIFFPOOL (Ying et al., 2018) are end-to-end supervised models which share similar two-phase process by (i) using stacked multiple graph convolutional layers (e.g., GCN layer (Kipf & Welling,

2017) or GraphSAGE layer (Hamilton et al., 2017a)) to aggregate node feature vectors, and (ii) applying a graph-level pooling layer (e.g., mean, max or sum pooling, sort pooling or differentiable pooling) to obtain the graph embeddings which are then fed to a fully-connected layer followed by a softmax layer to predict the graph labels.

Graph neural networks (GNNs) (Scarselli et al., 2009) aim to iteratively update the vector representation of each node by recursively propagating the representations of its neighbors using a recurrent function until convergence. The recurrent function can be a neural network e.g., gated recurrent unit (GRU) (Li et al., 2016), or multi-layer perceptron (MLP) (Xu et al., 2019). Note that both stacked GCN and GraphSAGE multiple layers can be seen as variants of the recurrent function in GNNs. Other graph embedding models are briefly summarized in (Zhou et al., 2018; Zhang et al., 2018c; Wu et al., 2019).

3 THE PROPOSED U2GNN

In this section, we detail how to construct our U2GNN and then present how U2GNN learns model parameters to produce node and graph embeddings.

Graph classification. Given a set of graphs $\{\mathcal{G}_1, \mathcal{G}_2, ..., \mathcal{G}_N\} \subseteq \mathsf{G}$ and their corresponding class labels $\{\mathsf{y}_1, \mathsf{y}_2, ..., \mathsf{y}_N\} \subseteq \mathcal{Y}$, our U2GNN aims to learn a plausible embedding $\mathbf{o}_{\mathcal{G}}$ of each entire graph $\mathcal{G}$ in order to predict its label y.

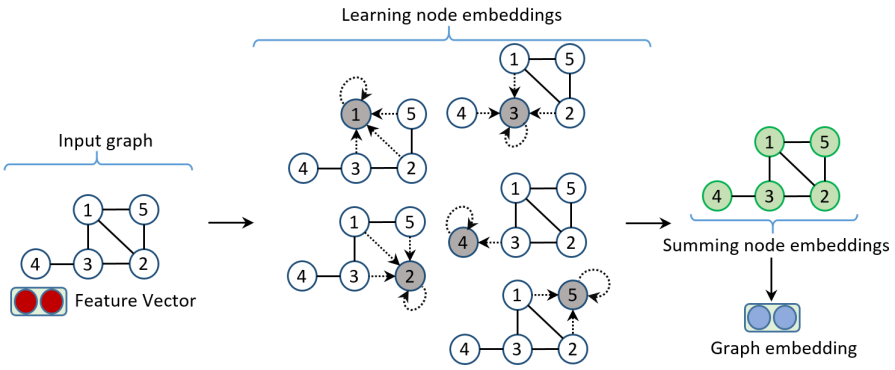

Figure 1: Illustration of our U2GNN learning process.

Each graph $\mathcal{G}$ is defined as $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$, where $\mathcal{V}$ is a set of nodes, $\mathcal{E}$ is a set of edges, and $\mathbf{X} \in \mathbb{R}^{|\mathcal{V}| \times d}$ represents feature vectors of nodes. In U2GNN, as illustrated in Figures 1 and 2, we use a universal self-attention network (Dehghani et al., 2019) to learn a node embedding $\mathbf{o}_{\mathsf{v}}$ of each node $\mathsf{v} \in \mathcal{V}$, and then $\mathbf{o}_{\mathcal{G}}$ is simply returned by summing all learned node embeddings as follows:[1]

$$\mathbf{o}_{\mathcal{G}} = \textsc{Sum}\left(\{\mathbf{o}_{\mathsf{v}}, \forall \mathsf{v} \in \mathcal{V}\}\right) \tag{1}$$

Constructing U2GNN. Formally, given an input graph $\mathcal{G}$, we uniformly sample a set of N neighbors for each $\mathsf{v} \in \mathcal{V}$, and then use node v and its neighbors for the U2GNN learning process.[2] For example, as illustrated in Figure 2, we generate a set of $N = 3$ neighbors $\{1, 2, 4\}$ for node 3, and then consider $\{3, 1, 2, 4\}$ as an input to U2GNN where we leverage on the universal self-attention network (Dehghani et al., 2019) to learn an effective embedding of node 3.

Intuitively, the universal self-attention network can help to better aggregate feature vectors from neighbors of a given node to produce its plausible embedding. In particular, each node and its neighbors is transformed into a sequence of feature vectors which are then iteratively refined at each timestep – using a self-attention mechanism (Vaswani et al., 2017) followed by a recurrent transition along with adding residual connection (He et al., 2016) and layer normalization (LAYERNORM) (Ba et al., 2016).

[1]The experimental results in (Xu et al., 2019) show that the sum pooling performs better than the mean and max poolings.

[2]We sample a different set of neighbors at each training step.

Given a sampled sequence of $(N+1)$ nodes $\{\mathsf{v}_i\}_{i=1}^{N+1}$ where $\{\mathsf{v}_2, \mathsf{v}_3, ..., \mathsf{v}_{N+1}\}$ are neighbors of v_1, we obtain an input sequence of feature vectors $\{\mathbf{h}_{\mathsf{v}_i}^0\}_{i=1}^{N+1}$ for which $\mathbf{h}_{\mathsf{v}_i}^0 = \mathbf{X}_{\mathsf{v}_i} \in \mathbb{R}^d$. In U2GNN, at each step t, we consider $\{\mathbf{h}_{\mathsf{v}_i}^{t-1}\}_{i=1}^{N+1}$ as an input sequence and produce an output sequence $\{\mathbf{h}_{\mathsf{v}_i}^t\}_{i=1}^{N+1}$ for which $\mathbf{h}_{\mathsf{v}_i}^t \in \mathbb{R}^d$ as follows:

$$\mathbf{h}_{\mathsf{v}_i}^t = \text{LayerNorm}\left(\mathbf{x}_{\mathsf{v}_i}^t + \text{Transition}\left(\mathbf{x}_{\mathsf{v}_i}^t\right)\right) \tag{2}$$

$$\text{with } \mathbf{x}_{\mathsf{v}_i}^t = \text{LayerNorm}\left(\mathbf{h}_{\mathsf{v}_i}^{t-1} + \text{ATT}\left(\mathbf{h}_{\mathsf{v}_i}^{t-1}\right)\right) \tag{3}$$

where $\text{Transition}(.)$ and $\text{ATT}(.)$ denote a feed-forward network and a self-attention network respectively as follows:

$$\text{Transition}\left(\mathbf{x}_{\mathsf{v}_i}^t\right) = \mathbf{W}_2 \mathsf{ReLU}\left(\mathbf{W}_1 \mathbf{x}_{\mathsf{v}_i}^t + \mathbf{b}_1\right) + \mathbf{b}_2 \tag{4}$$

where $\mathbf{W}_1 \in \mathbb{R}^{k \times d}$ and $\mathbf{W}_2 \in \mathbb{R}^{d \times k}$ are weight matrices, and $\mathbf{b}_1$ and $\mathbf{b}_2$ are bias parameters, and:

$$\text{ATT}\left(\mathbf{h}_{\mathsf{v}_i}^{t-1}\right) = \sum_{j=1}^{N+1} \alpha_{i,j} \left(\boldsymbol{V} \mathbf{h}_{\mathsf{v}_j}^{t-1}\right) \tag{5}$$

where $\boldsymbol{V} \in \mathbb{R}^{d \times d}$ is a value-projection weight matrix; $\alpha_{i,j}$ is an attention weight which is computed using the $\mathsf{softmax}$ function over scaled dot products between the i-th and j-th input nodes:

$$\alpha_{i,j} = \mathsf{softmax}\left(\frac{\left(\boldsymbol{Q}\mathbf{h}_{\mathsf{v}_i}^{t-1}\right) \cdot \left(\boldsymbol{K}\mathbf{h}_{\mathsf{v}_j}^{t-1}\right)}{\sqrt{d}}\right) \tag{6}$$

where $\boldsymbol{Q} \in \mathbb{R}^{d \times d}$ and $\boldsymbol{K} \in \mathbb{R}^{d \times d}$ are query-projection and key-projection matrices, respectively.[3]

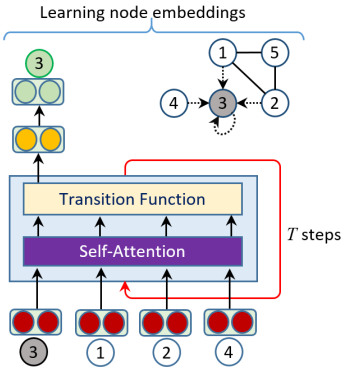

Figure 2: Illustration of learning an embedding for node 3 with $d = 2$ and $N = 3$.

Algorithm 1: The unsupervised learning process.

Input: A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$.
for $\mathsf{v} \in \mathcal{V}$ **do**
  SAMPLE a set C_v of N neighbors of v
  $\forall \mathsf{u} \in \{\mathsf{v}\} \cup \mathsf{C}_\mathsf{v}$
  for t = *1, 2, ..., T* **do**
    $\mathbf{x}_\mathsf{u}^t \leftarrow \text{LayerNorm}\left(\mathbf{h}_\mathsf{u}^{t-1} + \text{Att}\left(\mathbf{h}_\mathsf{u}^{t-1}\right)\right)$
    $\mathbf{h}_\mathsf{u}^t \leftarrow \text{LayerNorm}\left(\mathbf{x}_\mathsf{u}^t + \text{Transition}\left(\mathbf{x}_\mathsf{u}^t\right)\right)$
$\mathbf{o}_\mathsf{v} \leftarrow \mathbf{h}_\mathsf{v}^T, \forall \mathsf{v} \in \mathcal{V}$
$\mathbf{o}_\mathcal{G} = \text{Sum}\left(\{\mathbf{o}_\mathsf{v}, \forall \mathsf{v} \in \mathcal{V}\}\right)$

Learning parameters of U2GNN in "unsupervised" fashion: After T steps, we use the vector representation $\mathbf{h}_{\mathsf{v}_1}^T$ to infer node embeddings $\mathbf{o}_{\mathsf{v}_1}$. For example, as shown in Figure 2, we have $\mathsf{v}_1 = 3$, $\mathsf{v}_2 = 1$, $\mathsf{v}_3 = 2$ and $\mathsf{v}_4 = 4$, and then consider $\mathbf{h}_3^T$ to infer $\mathbf{o}_3$. We learn our model parameters (including the weight matrices and biases as well as node embeddings $\mathbf{o}_\mathsf{v}$) by minimizing the sampled softmax loss function (Jean et al., 2015) applied to node $\mathsf{v} \in \mathcal{V}$ as follows:

$$\mathcal{L}_{\text{U2GNN}}\left(\mathsf{v}\right) = -\log \frac{\exp(\mathbf{o}_\mathsf{v} \cdot \mathbf{h}_\mathsf{v}^T)}{\sum_{\mathsf{v}' \in \mathcal{V}'} \exp(\mathbf{o}_{\mathsf{v}'} \cdot \mathbf{h}_\mathsf{v}^T)} \tag{7}$$

[3]It is important to note that the self-attention mechanism is much different with GAT (Veličković et al., 2018). GAT borrows the standard attention technique from (Bahdanau et al., 2015) in using a single-layer feedforward neural network parametrized by a weight vector and then applying the LeakyReLU non-linearity function followed by the $\mathsf{softmax}$ function to compute importance weights of neighbors of a given node. More information about GCN, GraphSAGE and GAT can be found in Appendix A.

where $\mathcal{V}'$ is a subset sampled from $\mathcal{V}$.

We briefly describe the general learning process of our proposed U2GNN model in Algorithm 1. Here, the learned node embeddings $\mathbf{o}_{\mathsf{v}}$ are used as the final representations of nodes $\mathsf{v} \in \mathcal{V}$. After that, we obtain the plausible embedding $\mathbf{o}_{\mathcal{G}}$ of the graph $\mathcal{G}$ by summing all learned node embeddings as mentioned in Equation 1. Finally, we employ a logistic regression classifier from LIBLINEAR (Fan et al., 2008) to predict the graph labels.

Learning parameters of U2GNN in "supervised" fashion: After T steps, we use $\mathbf{h}_{\mathsf{v}_1}^T$ as the final embedding of node $\mathsf{v}_1 \in \mathcal{V}$ (i.e., $\mathbf{o}_{\mathsf{v}_1} = \mathbf{h}_{\mathsf{v}_1}^T$). We then feed $\mathbf{o}_{\mathcal{G}}$ to a fully connected layer to predict the graph labels. We learn the model parameters by minimizing the cross entropy loss function.

Intuition: In general, on the node level, each node and its neighbors are iteratively attended in the recurrent process with weight matrices shared across timesteps and iterations, thus U2GNN can memorize the potential dependencies among nodes within substructures. On the graph level, U2GNN views the shared weight matrices as memories to access the updated node-level information from previous iterations to further aggregate broader dependencies among substructures into implicit graph representations in subsequent iterations. Therefore, U2GNN is advantageous to capture both global and local graph structures to learn effective node and graph embeddings, leading to state-of-the-art performances for the graph classification task.

4 EXPERIMENTAL SETUP

We prove the effectiveness of our U2GNN on the graph classification task using a range of well-known benchmark datasets for both supervised and unsupervised fashions.

4.1 DATASETS

We use 9 well-known datasets consisting of 5 social network datasets (COLLAB, IMDB-B, IMDB-M, RDT-B and RDT-M5K) (Yanardag & Vishwanathan, 2015) and 4 bioinformatics datasets (DD, MUTAG, PROTEINS and PTC). We follow (Niepert et al., 2016; Zhang et al., 2018b) to use node degrees as features on all social network datasets as these datasets do not have available node features. Table 1 reports the statistics of these datasets.

Dataset	**#G**	**#Cls**	**A.NG**	**A.NN**	d
COLLAB	5,000	3	74.5	65.9	–
IMDB-B	1,000	2	19.8	9.8	–
IMDB-M	1,500	3	13.0	10.1	–
RDT-B	2,000	2	429.6	2.3	–
RDT-M5K	5,000	5	508.5	2.3	–
DD	1,178	2	284.3	5.0	82
MUTAG	188	2	17.9	2.2	7
PROTEINS	1,113	2	39.1	3.7	3
PTC	344	2	25.6	2.0	19

Table 1: Statistics of the experimental benchmark datasets. **#G** denotes the numbers of graphs. **#Cls** denotes the number of graph classes. **A.NG** denotes the average number of nodes per graph. **A.NN** denotes the average number of neighbors per node. d is the dimension of node feature vectors (i.e. the number of node labels).

Social networks datasets. COLLAB is a scientific dataset where each graph represents a collaboration network of a corresponding researcher with other researchers from each of 3 physics fields; and each graph is labeled to a physics field the researcher belong to. IMDB-B and IMDB-M are movie collaboration datasets where each graph is derived from actor/actress and genre information of different movies on IMDB, in which nodes correspond to actors/actresses, and each edge represents a co-appearance of two actors/actresses in the same movie; and each graph is assigned to a genre. RDT-B and RDT-M5K are datasets derived from Reddit community, in which each online discussion thread is viewed as a graph where nodes correspond to users, two users are linked if at least one of them replied to another's comment; and each graph is labeled to a sub-community the corresponding thread belongs to.

Bioinformatics datasets. DD (Dobson & Doig, 2003) is a collection of 1,178 protein network structures with 82 discrete node labels, where each graph is classified into enzyme or non-enzyme class.

PROTEINS comprises 1,113 graphs obtained from (Borgwardt et al., 2005) to present secondary structure elements (SSEs). MUTAG (Debnath et al., 1991) is a collection of 188 nitro compound networks with 7 discrete node labels, where classes indicate a mutagenic effect on a bacterium. PTC (Toivonen et al., 2003) consists of 344 chemical compound networks with 19 discrete node labels where classes show carcinogenicity for male and female rats.

4.2 TRAINING PROTOCOL TO LEARN GRAPH EMBEDDINGS

Coordinate embedding: The relative coordination among nodes might provide meaningful information about graph structure. We follow Dehghani et al. (2019) to associate each position i at step t a pre-defined coordinate embedding $\mathbf{p}_i^t$ using the sinusoidal functions (Vaswani et al., 2017), thus we can change Equation 3 in Section 3 to:

$$\begin{aligned} \mathbf{x}_{\mathsf{v}_i}^t &= \textsc{LayerNorm}\left(\mathbf{h}_{\mathsf{v}_i}^{t-1} + \text{ATT}\left(\mathbf{h}_{\mathsf{v}_i}^{t-1} + \mathbf{p}_i^t\right)\right) \\ \text{with } \mathsf{p}_{i,2j}^t &= \sin(i/10000^{2j/d}) + \sin(t/10000^{2j/d}) \\ \mathsf{p}_{i,2j+1}^t &= \cos(i/10000^{2j/d}) + \cos(t/10000^{2j/d}) \end{aligned} \tag{8}$$

From the preliminary experiments, adding coordinate embeddings enhances classification results on MUTAG and PROTEINS in the unsupervised fashion, hence we use the coordinate embeddings only for these two datasets.

Hyper-parameter setting to learn our model parameters for all experimental datasets: (i) Regarding the unsupervised fashion, we fix the hidden size of the feed-forward network in Equation 4 to 1024 ($k = 1024$), and the number of samples in the sampled loss function $\mathcal{L}_{\mathsf{U2GNN}}$ to 512 ($|\mathcal{V}'| = 512$) in Equation 7. We set the batch size to 512 for COLLAB, DD and RDT-B; 1024 for RDT-M5K; and 128 for remaining datasets. (ii) Regarding the supervised fashion, we vary the the hidden size k in $\{32, 128, 512, 1024\}$ and the batch size in $\{1, 2, 4\}$. (iii) For both the unsupervised and supervised fashions, we use the number N of neighbors sampled for each node from $\{4, 8, 16\}$ and the number T of steps from $\{1, 2, 3, 4, 5, 6\}$. We apply the Adam optimizer (Kingma & Ba, 2014) to train our U2GNN model and apply a grid search to select the Adam initial learning rate $lr \in \{5e^{-5}, 1e^{-4}, 5e^{-4}, 1e^{-3}\}$. We run up to 50 epochs and evaluate the model as in what follows.

4.3 EVALUATION PROTOCOL

For each dataset, after obtaining the graph embeddings, we perform the same evaluation process from (Yanardag & Vishwanathan, 2015; Niepert et al., 2016; Zhang et al., 2018b; Xu et al., 2019; Xinyi & Chen, 2019), which is using 10-fold cross-validation scheme to calculate the classification performance for a fair comparison.[4]

Baseline models: We compare our U2GNN with up-to-date strong baselines as follows:

- **Unsupervised approaches:** Graphlet Kernel (GK) (Shervashidze et al., 2009), Weisfeiler-Lehman kernel (WL) (Shervashidze et al., 2011), Deep Graph Kernel (DGK) (Yanardag & Vishwanathan, 2015) and Anonymous Walk Embedding (AWE) (Ivanov & Burnaev, 2018).
- **Supervised approaches:** PATCHY-SAN (PSCN) (Niepert et al., 2016), Graph Convolutional Network (GCN) (Kipf & Welling, 2017), GraphSAGE (Hamilton et al., 2017a), Deep Graph CNN (DGCNN) (Zhang et al., 2018b), Graph Capsule Convolution Neural Network (GCAPS) (Verma & Zhang, 2018), Capsule Graph Neural Network (CapsGNN) (Xinyi & Chen, 2019), Graph Isomorphism Network (GIN) (Xu et al., 2019), Graph Feature Network (GFN) (Chen et al., 2019), Invariant-Equivariant Graph Network (IEGN) (Maron et al., 2019b), Provably Powerful Graph Network (PPGN) (Maron et al., 2019a) and Discriminative Structural Graph Classification (DSGC) (Seo et al., 2019).

We report the baseline results taken from the original papers or published in (Ivanov & Burnaev, 2018; Verma & Zhang, 2018; Xinyi & Chen, 2019; Chen et al., 2019; Seo et al., 2019).

[4]Regarding the unsupervised fashion, we use the logistic regression classifier from LIBLINEAR (Fan et al., 2008) with setting the termination criterion to 0.001.

5 EXPERIMENTAL RESULTS

Table 2 presents the experimental results on the 9 benchmark datasets. We use "**Unsupervised**" to denote the unsupervised graph embedding models that can access all nodes from the whole dataset, but not the graph labels; and we use "**Supervised**" to denote the supervised models that can use the graph labels of training graphs during training.

In general, our unsupervised U2GNN achieves state-of-the-art performances on a range of benchmarks for the graph classification task, hence this demonstrates a high impact of the unsupervised U2GNN in inferring the plausible node and graph embeddings. It is to note that several existing works (Xinyi & Chen, 2019; Xu et al., 2019; Chen et al., 2019; Maron et al., 2019b; Seo et al., 2019) compared the supervised models with the unsupervised models together due to the use of the same 10-fold cross-validation scheme. *However, our unsupervised U2GNN significantly outperforms the supervised methods in most of benchmark cases by a large margin of 18+%, thus we suggest that future GNN works should separate two training fashions.*

	Model	COLLAB	IMDB-B	IMDB-M	RDT-B	RDT-M5K
Unsupervised	GK (2009)	72.84 ± 0.28	65.87 ± 0.98	43.89 ± 0.38	77.34 ± 0.18	41.01 ± 0.17
	WL (2011)	79.02 ± 1.77	73.40 ± 4.63	49.33 ± 4.75	81.10 ± 1.90	49.44 ± 2.36
	DGK (2015)	73.09 ± 0.25	66.96 ± 0.56	44.55 ± 0.52	78.04 ± 0.39	41.27 ± 0.18
	AWE (2018)	73.93 ± 1.94	74.45 ± 5.83	51.54 ± 3.61	**87.89 ± 2.53**	50.46 ± 1.91
	U2GNN	**95.62 ± 0.92**	**93.50 ± 2.27**	**74.80 ± 4.11**	84.80 ± 1.53	**77.25 ± 1.46**
Supervised	DSGC (2019)	79.20 ± 1.60	73.20 ± 4.90	48.50 ± 4.80	92.20 ± 2.40	–
	GFN (2019)	81.50 ± 2.42	73.00 ± 4.35	51.80 ± 5.16	–	**57.59 ± 2.40**
	PPGN (2019a)	81.38 ± 1.42	73.00 ± 5.77	50.46 ± 3.59	–	–
	GIN (2019)	80.20 ± 1.90	75.10 ± 5.10	52.30 ± 2.80	**92.40 ± 2.50**	57.50 ± 1.50
	IEGN (2019b)	77.92 ± 1.70	71.27 ± 4.50	48.55 ± 3.90	–	–
	CapsGNN (2019)	79.62 ± 0.91	73.10 ± 4.83	50.27 ± 2.65	–	50.46 ± 1.91
	GCAPS (2018)	77.71 ± 2.51	71.69 ± 3.40	48.50 ± 4.10	87.61 ± 2.51	50.10 ± 1.72
	DGCNN (2018b)	73.76 ± 0.49	70.03 ± 0.86	47.83 ± 0.85	76.02 ± 1.73	48.70 ± 4.54
	GCN (2017)	**81.72 ± 1.64**	73.30 ± 5.29	51.20 ± 5.13	–	56.81 ± 2.37
	GraphSAGE (2017a)	79.70 ± 1.70	72.40 ± 3.60	49.90 ± 5.00	89.10 ± 1.90	–
	PSCN (2016)	72.60 ± 2.15	71.00 ± 2.29	45.23 ± 2.84	86.30 ± 1.58	49.10 ± 0.70
	U2GNN	77.84 ± 1.48	**79.40 ± 4.35**	**56.20 ± 3.35**	80.25 ± 2.38	50.90 ± 2.31

	Model	DD	PROTEINS	MUTAG	PTC
Unsupervised	GK (2009)	78.45 ± 0.26	71.67 ± 0.55	81.58 ± 2.11	57.26 ± 1.41
	WL (2011)	79.78 ± 0.36	74.68 ± 0.49	82.05 ± 0.36	57.97 ± 0.49
	DGK (2015)	73.50 ± 1.01	75.68 ± 0.54	87.44 ± 2.72	60.08 ± 2.55
	AWE (2018)	71.51 ± 4.02	–	**87.87 ± 9.76**	–
	U2GNN	**95.67 ± 1.89**	**78.07 ± 3.36**	81.34 ± 6.56	**84.59 ± 5.12**
Supervised	DSGC (2019)	77.40 ± 6.40	74.20 ± 3.80	86.70 ± 7.60	–
	GFN (2019)	78.78 ± 3.49	76.46 ± 4.06	90.84 ± 7.22	–
	PPGN (2019a)	–	77.20 ± 4.73	90.55 ± 8.70	66.17 ± 6.54
	GIN (2019)	–	76.20 ± 2.80	89.40 ± 5.60	64.60 ± 7.00
	IEGN (2019b)	–	75.19 ± 4.30	84.61 ± 10.0	59.47 ± 7.30
	CapsGNN (2019)	75.38 ± 4.17	76.28 ± 3.63	86.67 ± 6.88	–
	GCAPS (2018)	77.62 ± 4.99	76.40 ± 4.17	–	66.01 ± 5.91
	DGCNN (2018b)	79.37 ± 0.94	75.54 ± 0.94	85.83 ± 1.66	58.59 ± 2.47
	GCN (2017)	79.12 ± 3.07	75.65 ± 3.24	87.20 ± 5.11	–
	GraphSAGE (2017a)	65.80 ± 4.90	65.90 ± 2.70	79.80 ± 13.90	–
	PSCN (2016)	77.12 ± 2.41	75.89 ± 2.76	**92.63 ± 4.21**	62.29 ± 5.68
	U2GNN	**81.24 ± 1.84**	**78.53 ± 4.07**	89.97 ± 3.65	**79.36 ± 4.06**

Table 2: Graph classification results (% accuracy). The best scores are in **bold**.

Regarding our supervised U2GNN, on the social network datasets, our model produces new state-of-the-art performances on IMDB-B and IMDB-M, especially U2GNN significantly achieves 4+%

absolute higher accuracies than all supervised baselines. In addition, U2GNN obtains promising scores on COLLAB, RDT-B and RDT-M5K. On the bioinformatics datasets, our U2GNN obtains new highest accuracies on DD, PROTEINS and PTC. Moreover, U2GNN produces a competitive accuracy compared with those of baseline models on MUTAG, for which there are no significant differences between our supervised U2GNN and some supervised baselines (e.g., GFN, GIN and PPGN). Note that there are only 188 graphs in the MUTAG dataset, which explains the high variance in the results.

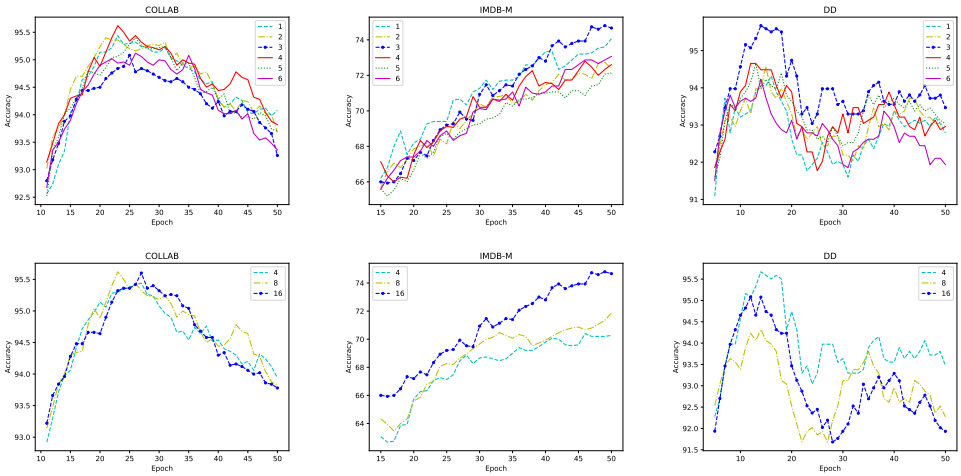

Figure 3: Effects of the number of steps (T) (in the 3 top figures), and the number of neighbors (N) sampled for each node (in the 3 bottom figures) in the unsupervised fashion. Regarding each dataset, for all 10 folds, we vary the value of either T or N while using the same fixed values of other hyper-parameters.

Next we investigate the effects of hyper-parameters on the experimental datasets in the unsupervised fashion in Figure 3.[5] In general, our unsupervised U2GNN could consistently obtain better results than those of baselines with any value of T and N, as long as the training process is stopped precisely for all datasets. In particular, we find that higher T helps on most of the datasets, and especially boosts the performance on bioinformatics data. A possible reason is that the bioinformatics datasets comprise sparse networks where the average number of neighbors per node is below 5 as shown in Table 1, hence we need to use more steps to learn graph properties. In addition, using small N generally produces higher performances on the bioinformatics datasets, while using higher values of N is more suitable for the social network datasets. Besides, the social network datasets are much denser than the bioinformatics datasets, thus this is reason why we should use more sampled neighbors on the social networks rather than the bioinformatics ones. Note that the similar findings also occur in the supervised fashion.

To qualitatively demonstrate the effectiveness of capturing the local and global graph properties, we use t-SNE (Maaten & Hinton, 2008) to visualize the learned node and graph embeddings in the unsupervised fashion on the DD dataset where the node labels are available. It can be seen from Figure 4 that our unsupervised U2GNN can effectively capture the local structure wherein the nodes are clustered according to the node labels, and the global structure wherein the graph embeddings are well-separated from each other; verifying the plausibility of the learned node and graph embeddings.

6 Conclusion

In this paper, we introduce a novel graph neural network model U2GNN for the graph classification task. Given an input graph, U2GNN applies a self-attention mechanism followed by a recurrent transition to learn the node embeddings and then sums all the learned node embeddings to obtain the embedding of the entire graph. Our U2GNN achieves new highest accuracies on most of 9

[5] More figures can be found in Appendix B.

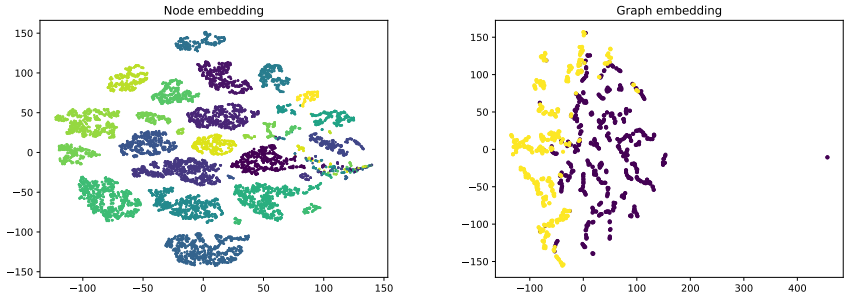

Figure 4: A visualization of the node and graph embeddings learned by our unsupervised U2GNN on the DD dataset. We do not include the embedding visualization of other baselines due to a significant improvement of the unsupervised U2GNN.

well-known benchmark datasets in both the supervised and unsupervised fashions, using the same 10-fold cross-validation scheme, against the up-to-date unsupervised and supervised baselines. In future works, we plan to investigate the effectiveness of U2GNN on other important tasks such as node classification and link prediction. Our code is available at: https://anonymous-url/.

REFERENCES

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *ICLR*, 2015.

Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

Karsten M. Borgwardt and Hans-Peter Kriegel. Shortest-Path Kernels on Graphs. In *ICDM*, pp. 74–81, 2005.

Karsten M Borgwardt, Cheng Soon Ong, Stefan Schönauer, SVN Vishwanathan, Alex J Smola, and Hans-Peter Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 21(suppl_1): i47–i56, 2005.

Ting Chen, Song Bian, and Yizhou Sun. Are powerful graph neural nets necessary? a dissection on graph classification. *arXiv preprint arXiv:1905.04579*, 2019.

Asim Kumar Debnath, Rosa L Lopez de Compadre, Gargi Debnath, Alan J Shusterman, and Corwin Hansch. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. Correlation with molecular orbital energies and hydrophobicity. *Journal of medicinal chemistry*, 34(2):786–797, 1991.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal Transformers. *ICLR*, 2019.

Paul D Dobson and Andrew J Doig. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of molecular biology*, 330(4):771–783, 2003.

Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. LIBLINEAR: A Library for Large Linear Classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008.

Thomas Gärtner, Peter Flach, and Stefan Wrobel. On graph kernels: Hardness results and efficient alternatives. In *Learning Theory and Kernel Machines*, pp. 129–143. 2003.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural Message Passing for Quantum Chemistry. In *ICML*, pp. 1263–1272, 2017.

William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NIPS*, pp. 1024–1034, 2017a.

William L. Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*, 2017b.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pp. 770–778, 2016.

Thomas Hofmann, Bernhard Schölkopf, and Alexander J Smola. Kernel methods in machine learning. *The annals of statistics*, pp. 1171–1220, 2008.

Sergey Ivanov and Evgeny Burnaev. Anonymous walk embeddings. In *ICML*, pp. 2191–2200, 2018.

Sébastien Jean, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio. On using very large target vocabulary for neural machine translation. In *ACL*, pp. 1–10, 2015.

Hisashi Kashima, Koji Tsuda, and Akihiro Inokuchi. Marginalized kernels between labeled graphs. In *ICML*, pp. 321–328, 2003.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

Nils M Kriege, Fredrik D Johansson, and Christopher Morris. A survey on graph kernels. *arXiv preprint arXiv:1903.11835*, 2019.

Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *ICML*, pp. 1188–1196, 2014.

Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated Graph Sequence Neural Networks. *ICLR*, 2016.

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9:2579–2605, 2008.

Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. Provably powerful graph networks. *arXiv preprint arXiv:1905.11136*, 2019a.

Haggai Maron, Heli Ben-Hamu, Nadav Shamir, and Yaron Lipman. Invariant and equivariant graph networks. *ICLR*, 2019b.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, pp. 3111–3119, 2013.

Annamalai Narayanan, Mahinthan Chandramohan, Rajasekar Venkatesan, Lihui Chen, Yang Liu, and Shantanu Jaiswal. graph2vec: Learning distributed representations of graphs. *arXiv preprint arXiv:1707.05005*, 2017.

Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning Convolutional Neural Networks for Graphs. In *ICML*, pp. 2014–2023, 2016.

Giannis Nikolentzos, Giannis Siglidis, and Michalis Vazirgiannis. Graph kernels: A survey. *arXiv preprint arXiv:1904.12218*, 2019.

Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. DeepWalk: Online Learning of Social Representations. In *SIGKDD*, pp. 701–710, 2014.

Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. Dynamic routing between capsules. In *NIPS*, pp. 3859–3869, 2017.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

Younjoo Seo, Andreas Loukas, and Nathanael Peraudin. Discriminative structural graph classification. *arXiv preprint arXiv:1905.13422*, 2019.

Nino Shervashidze, SVN Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten Borgwardt. Efficient Graphlet Kernels for Large Graph Comparison. In *AISTATS*, pp. 488–495, 2009.

Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(Sep):2539–2561, 2011.

Hannu Toivonen, Ashwin Srinivasan, Ross D King, Stefan Kramer, and Christoph Helma. Statistical evaluation of the predictive toxicology challenge 2000–2001. *Bioinformatics*, 19(10):1183–1193, 2003.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NIPS*, pp. 5998–6008, 2017.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. *ICLR*, 2018.

Saurabh Verma and Zhi-Li Zhang. Graph capsule convolutional neural networks. *arXiv preprint arXiv:1805.08090*, 2018.

S Vichy N Vishwanathan, Nicol N Schraudolph, Risi Kondor, and Karsten M Borgwardt. Graph kernels. *Journal of Machine Learning Research*, 11(Apr):1201–1242, 2010.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596*, 2019.

Zhang Xinyi and Lihui Chen. Capsule Graph Neural Network. *ICLR*, 2019.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How Powerful Are Graph Neural Networks? *ICLR*, 2019.

Pinar Yanardag and SVN Vishwanathan. Deep graph kernels. In *SIGKDD*, pp. 1365–1374, 2015.

Rex Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L. Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *NeurIPS*, pp. 4805–4815, 2018.

Daokun Zhang, Jie Yin, Xingquan Zhu, and Chengqi Zhang. Network representation learning: A survey. *IEEE Transactions on Big Data*, to appear, 2018a.

Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An End-to-End Deep Learning Architecture for Graph Classification. In *AAAI*, 2018b.

Ziwei Zhang, Peng Cui, and Wenwu Zhu. Deep learning on graphs: A survey. *arXiv preprint arXiv:1812.04202*, 2018c.

Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434*, 2018.

A GNN VARIANTS

Regarding GCN (Kipf & Welling, 2017), each node $\mathsf{v} \in \mathcal{V}$ has a feature vector $\boldsymbol{x}_{\mathsf{v}} \in \mathbb{R}^{d_1}$, and GCN infers a new vector representation $\boldsymbol{h}_{\mathsf{v}} \in \mathbb{R}^{d_2}$ for node v from its neighbors as follows:

$$\boldsymbol{h}_{\mathsf{v}} = \mathrm{g}\left(\sum_{\mathsf{u} \in \mathcal{N}_{\mathsf{v}}} (\boldsymbol{W}\boldsymbol{x}_{\mathsf{u}} + \boldsymbol{b})\right), \forall \mathsf{v} \in \mathcal{V} \tag{9}$$

where $\boldsymbol{W} \in \mathbb{R}^{d_2 \times d_1}$ and $\boldsymbol{b} \in \mathbb{R}^{d_2}$ are a weight matrix and a bias respectively, g is a non-linear activation function, and $\mathcal{N}_{\mathsf{v}}$ is the set of neighbors of node v. GCN can indirectly capture nodes at many hops away by using multiple GCN layers stacked on top of each other as follows:

$$\boldsymbol{h}_{\mathsf{v}}^{(k+1)} = \mathsf{g}\left(\sum_{\mathsf{u} \in \mathcal{N}_{\mathsf{v}}} \left(\boldsymbol{W}^{(k)} \boldsymbol{h}_{\mathsf{u}}^{(k)} + \boldsymbol{b}^{(k)}\right)\right), \forall \mathsf{v} \in \mathcal{V} \tag{10}$$

where k is the layer index, and $\boldsymbol{h}_{\mathsf{u}}^{(0)} = \boldsymbol{x}_{\mathsf{u}}$. The vector outputs of the last GCN layer are used as the final vector representations of nodes. GCN can be jointly trained with a classification classifier by minimizing the cross-entropy loss function.

GraphSAGE (Hamilton et al., 2017a) is an extension of GCN, in which the representation $\boldsymbol{h}_{\mathsf{v}}^{(k+1)}$ of node v after k-th GraphSAGE layer is produced as follows:

$$\boldsymbol{h}_{\mathsf{v}}^{(k+1)} = \mathsf{g}\left(\boldsymbol{W}_{SAGE}^{(k)} \left[\boldsymbol{h}_{\mathsf{v}}^{(k)} \oplus \boldsymbol{h}_{\mathcal{N}_{\mathsf{v}}}^{(k+1)}\right]\right), \forall \mathsf{v} \in \mathcal{V} \tag{11}$$

where $[\oplus]$ denotes a vector concatenation operation, and $\boldsymbol{h}_{\mathcal{N}_{\mathsf{v}}}^{(k+1)}$ can be computed in several ways such as using the GCN operation as follows:

$$\boldsymbol{h}_{\mathcal{N}_{\mathsf{v}}}^{(k+1)} = \mathsf{g}\left(\sum_{\mathsf{u} \in \mathcal{N}_{\mathsf{v}}} \left(\boldsymbol{W}_{GCN}^{(k)} \boldsymbol{h}_{\mathsf{u}}^{(k)} + \boldsymbol{b}^{(k)}\right)\right), \forall \mathsf{v} \in \mathcal{V} \tag{12}$$

or applying an element-wise max-pooling operation as follows:

$$\boldsymbol{h}_{\mathcal{N}_{\mathsf{v}}}^{(k+1)} = \mathsf{max}\left(\left\{\mathsf{g}\left(\boldsymbol{W}_{pool} \boldsymbol{h}_{\mathsf{u}}^{(k)} + \boldsymbol{b}\right), \forall \mathsf{u} \in \mathcal{N}_{\mathsf{v}}\right\}\right) \tag{13}$$

In addition, $\boldsymbol{h}_{\mathcal{N}_{\mathsf{v}}}^{(k+1)}$ can be computed by taking the mean (or sum) of all vectors in $\{\boldsymbol{h}_{\mathsf{u}}^{(k)}, \forall \mathsf{u} \in \mathcal{N}_{\mathsf{v}}\}$. $\mathcal{N}_{\mathsf{v}}$ is defined as a fixed-size, uniformly drawn from the set of all neighbor nodes of v, and uniformly sampled differently through each layer. The vector outputs of the last GraphSAGE layer are used to infer node embeddings. GraphSAGE employs the random walk algorithm (Perozzi et al., 2014) to generate context nodes for each target node and then adapts the negative sampling technique (Mikolov et al., 2013) to learn the model parameters.

GAT (Veličković et al., 2018) extends GCN in assigning importance weights to neighbors of a given node by exploring the standard attention technique (Bahdanau et al., 2015). GAT induces a new vector representation $\boldsymbol{h}_{\mathsf{v}}$ of node v from its neighbors as follows:

$$\boldsymbol{h}_{\mathsf{v}} = \mathsf{g}\left(\sum_{\mathsf{u} \in \mathcal{N}_{\mathsf{v}}} \tau_{\mathsf{v},\mathsf{u}} \boldsymbol{W} \boldsymbol{x}_{\mathsf{u}}\right), \forall \mathsf{v} \in \mathcal{V} \tag{14}$$

where $\tau_{\mathsf{v},\mathsf{u}}$ is an importance weight which is computed as follows:

$$\tau_{\mathsf{v},\mathsf{u}} = \frac{\exp\left(\mathsf{LeakyReLU}\left(\boldsymbol{a}^{\mathsf{T}}\left[\boldsymbol{W}\boldsymbol{x}_{\mathsf{v}} \oplus \boldsymbol{W}\boldsymbol{x}_{\mathsf{u}}\right]\right)\right)}{\sum_{\mathsf{u}' \in \mathcal{N}_{\mathsf{v}}} \exp\left(\mathsf{LeakyReLU}\left(\boldsymbol{a}^{\mathsf{T}}\left[\boldsymbol{W}\boldsymbol{x}_{\mathsf{v}} \oplus \boldsymbol{W}\boldsymbol{x}_{\mathsf{u}'}\right]\right)\right)} \tag{15}$$

B EFFECTS OF HYPER-PARAMETERS

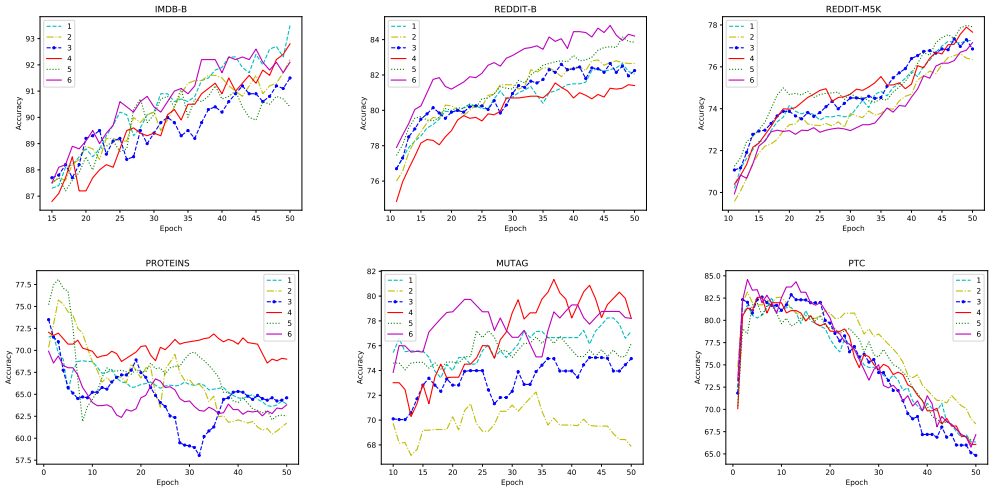

Figure 5: Effects of the number of steps (T) in the unsupervised fashion. Regarding each dataset, for all 10 folds, we vary the value of T while using the same fixed values of other hyper-parameters.

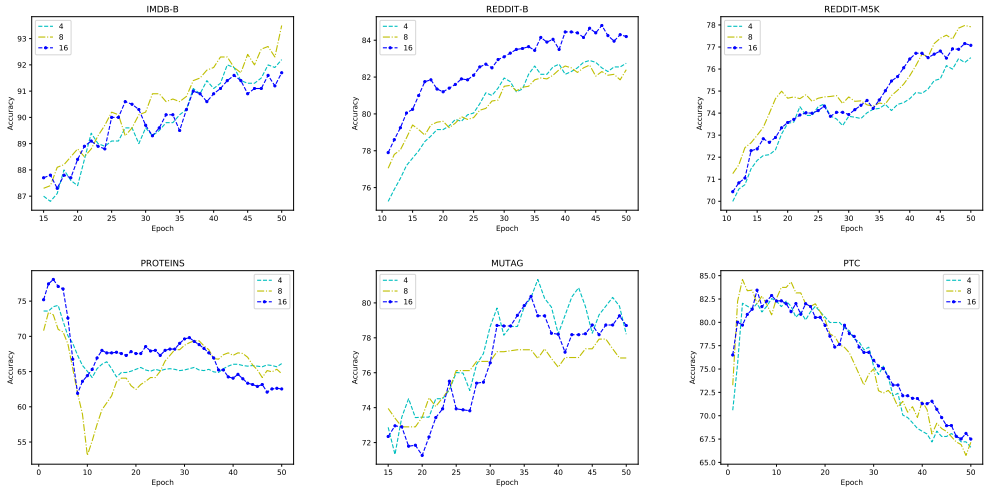

Figure 6: Effects of the number of neighbors (N) sampled for each node in the unsupervised fashion. Regarding each dataset, for all 10 folds, we vary the value of N while using the same fixed values of other hyper-parameters.