UNSUPERVISED UNIVERSAL SELF-ATTENTION NETWORK FOR GRAPH CLASSIFICATION

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 U2GAN, a novel unsupervised 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, U2GAN 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, U2GAN 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 show that our unsupervised U2GAN produces new state-of-the-art performances on a range of well-known benchmark datasets for the graph classification task. It even outperforms supervised methods in most of benchmark cases.

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., graphlet, 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 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 promis-

ing performances, nonetheless the dependency aspect among nodes, which often exhibit strongly in many kinds of real-world networks, has not been exploited effectively.

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 U2GAN – a novel **u**nsupervised **u**niversal **g**raph **a**ttention **n**etwork embedding model for the graph classification task. 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 U2GAN 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. After the training process, we 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 U2GAN, the novelty of memorizing the dependencies among nodes implies that U2GAN can explore the graph structure similarities locally and globally – an important feature that most of existing approaches are unable to do.
- The experimental results on 5 social network datasets and 6 bioinformatics datasets show that U2GAN produces new state-of-the-art (SOTA) accuracies on 8 datasets by a large margin and comparable accuracies on 3 remaining datasets. Noticeably, despite being unsupervised, it even outperforms most of up-to-date supervised approaches.
- To qualitatively demonstrate an advantage of U2GAN in capturing local and global graph properties, we utilize t-SNE (Maaten & Hinton, 2008) to visualize the learned node and graph embeddings to show well-separated clusters of embeddings according to their 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 U2GAN

In this section, we detail how to construct our U2GAN and then present how U2GAN 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 U2GAN aims to learn a plausible embedding $\mathbf{o}_{\mathcal{G}}$ of each entire graph $\mathcal{G}$ in order to predict its label y.

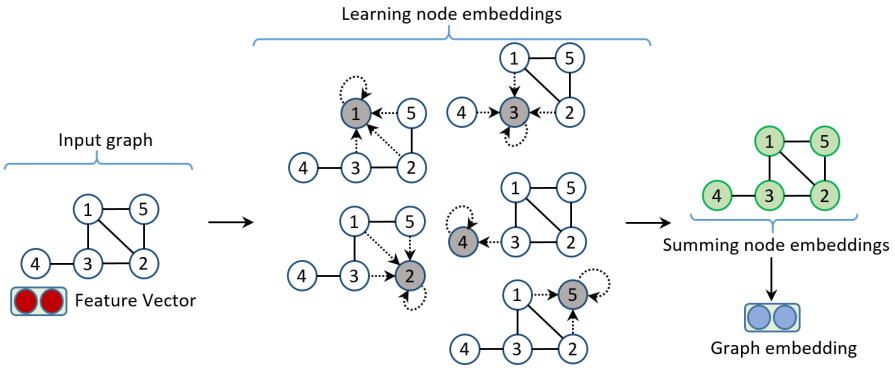

Figure 1: Illustration of our U2GAN learning process.

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 U2GAN, 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 U2GAN. 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 U2GAN 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 U2GAN 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).

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 $\{\mathbf{h}^0_{\mathsf{v}_i}\}_{i=1}^{N+1}$ for which $\mathbf{h}^0_{\mathsf{v}_i} = \mathbf{X}_{\mathsf{v}_i} \in \mathbb{R}^d$. In U2GAN, at

[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.

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:

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

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

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

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

where $\mathbf{W}_1$ and $\mathbf{W}_2$ 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.

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 $\mathbf{o}_3$.

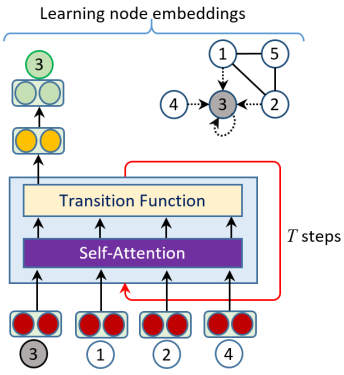

Figure 2: Illustration of learning an embedding for node 3 with $d = 2$ and $N = 3$.

Algorithm 1: The U2GAN learning process.

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

Learning parameters of U2GAN: 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:

$$\mathcal{L}_{\mathsf{U2GAN}}(\mathsf{v}) = -\log\frac{\exp(\mathbf{o}_\mathsf{v}\cdot\mathbf{h}_\mathsf{v}^T)}{\sum_{\mathsf{v}'\in\mathcal{V}'}\exp(\mathbf{o}_{\mathsf{v}'}\cdot\mathbf{h}_\mathsf{v}^T)} \tag{7}$$

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

We briefly describe the general learning process of our proposed U2GAN 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.

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 U2GAN

can memorize the potential dependencies among nodes within substructures. On the graph level, U2GAN 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, U2GAN 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 unsupervised U2GAN on the graph classification task using a range of well-known benchmark datasets as follows: (i) We unsupervisedly train U2GAN to obtain graph embeddings. (ii) We use the obtained graph embeddings as feature vectors to train a classifier to predict graph labels. (iii) We evaluate the classification performance, and then analyze effects of main hyper-parameters.

4.1 DATASETS

We use 11 well-known datasets consisting of 5 social network datasets (COLLAB, IMDB-B, IMDB-M, REDDIT-B and REDDIT-M5K) (Yanardag & Vishwanathan, 2015) and 6 bioinformatics datasets (DD, MUTAG, NCI1, NCI109, 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 experimental 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	–
REDDIT-B	2,000	2	429.6	2.3	–
REDDIT-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
NCI1	4,110	2	29.8	2.2	37
NCI109	4,127	2	29.6	2.2	38
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. REDDIT-B and REDDIT-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). NCI1 and NCI109 are two balanced datasets of 4,110 and 4,127 chemical compound graphs with 37 and 38 discrete node labels, respectively. 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:

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

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

Hyper-parameter setting. To learn our model parameters for all experimental datasets, we fix the hidden size of the feed-forward network in Equation 4 to 1024, and the number of samples in the sampled loss function $\mathcal{L}_{\mathsf{U2GAN}}$ to 512 ($|\mathcal{V}'| = 512$) in Equation 7. We set the batch size to 512 for COLLAB, DD, REDDIT-B and REDDIT-M5K, and 128 for remaining datasets. 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 U2GAN model and apply a grid search to select the Adam initial learning rate $lr \in \{5e^{-5}, 1e^{-4}, 5e^{-4}\}$. 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. We use LIBLINEAR (Fan et al., 2008) and report the mean and standard deviation of the accuracies over 10 folds within the cross-validation procedure.[3]

Baseline models: We compare our U2GAN 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),[4] 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).

5 EXPERIMENTAL RESULTS

Table 2 presents the experimental results on the 11 benchmark datasets. Regarding the social network datasets, our unsupervised U2GAN produces new state-of-the-art performances on COLLAB, IMDB-B, IMDB-M and REDDIT-M5K, especially U2GAN significantly achieves 14+% absolute higher accuracies than all baselines on these 4 datasets. In addition, U2GAN obtains comparable scores in comparison with other unsupervised models on REDDIT-B. These demonstrate a high impact of U2GAN in inferring plausible node and graph embeddings on the social networks.

[3] We use the logistic regression classifier from LIBLINEAR with setting termination criterion to 0.001.

[4] As applied in (Ying et al., 2018), GraphSAGE (Hamilton et al., 2017a) obtained low accuracies for the graph classification task, thus we do not include GraphSAGE as a strong supervised baseline.

	Model	COLLAB	IMDB-B	IMDB-M	REDDIT-B	REDDIT-M5K
Un-sup.	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
	U2GAN	**95.62 ± 0.92**	**93.50 ± 2.27**	**74.80 ± 4.11**	84.80 ± 1.53	**77.25 ± 1.46**
Sup.	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
	PSCN (2016)	72.60 ± 2.15	71.00 ± 2.29	45.23 ± 2.84	86.30 ± 1.58	49.10 ± 0.70

	Model	DD	PROTEINS	NCI1	NCI109	MUTAG	PTC
Un-sup.	GK (2009)	78.45 ± 0.26	71.67 ± 0.55	62.49 ± 0.27	80.32 ± 0.33	81.58 ± 2.11	57.26 ± 1.41
	WL (2011)	79.78 ± 0.36	74.68 ± 0.49	82.19 ± 0.18	82.46 ± 0.24	82.05 ± 0.36	57.97 ± 0.49
	DGK (2015)	73.50 ± 1.01	75.68 ± 0.54	80.31 ± 0.46	62.69 ± 0.23	87.44 ± 2.72	60.08 ± 2.55
	AWE (2018)	71.51 ± 4.02	–	–	–	87.87 ± 9.76	–
	U2GAN	**95.67 ± 1.89**	**78.07 ± 3.36**	82.55 ± 2.11	**83.33 ± 1.78**	81.34 ± 6.56	**84.59 ± 5.12**
Sup.	DSGC (2019)	77.40 ± 6.40	74.20 ± 3.80	79.80 ± 1.20	–	86.70 ± 7.60	–
	GFN (2019)	78.78 ± 3.49	76.46 ± 4.06	82.77 ± 1.49	–	90.84 ± 7.22	–
	PPGN (2019a)	–	77.20 ± 4.73	83.19 ± 1.11	81.84 ± 1.85	90.55 ± 8.70	66.17 ± 6.54
	GIN (2019)	–	76.20 ± 2.80	82.70 ± 1.70	–	89.40 ± 5.60	64.60 ± 7.00
	IEGN (2019b)	–	75.19 ± 4.30	73.71 ± 2.60	72.48 ± 2.50	84.61 ± 10.0	59.47 ± 7.30
	CapsGNN (2019)	75.38 ± 4.17	76.28 ± 3.63	78.35 ± 1.55	–	86.67 ± 6.88	–
	GCAPS (2018)	77.62 ± 4.99	76.40 ± 4.17	82.72 ± 2.38	81.12 ± 1.28	–	66.01 ± 5.91
	DGCNN (2018b)	79.37 ± 0.94	75.54 ± 0.94	74.44 ± 0.47	75.03 ± 1.72	85.83 ± 1.66	58.59 ± 2.47
	GCN (2017)	79.12 ± 3.07	75.65 ± 3.24	**83.65 ± 1.69**	–	87.20 ± 5.11	–
	PSCN (2016)	77.12 ± 2.41	75.89 ± 2.76	78.59 ± 1.89	–	**92.63 ± 4.21**	62.29 ± 5.68

Table 2: Graph classification results (% accuracy) on the experimental benchmarks. "**Un-sup.**" denotes unsupervised graph embedding models. "**Sup.**" denotes supervised graph embedding models that use graph class labels when training the models. The best scores are in **bold**.

Regarding the bioinformatics datasets, U2GAN obtains new highest scores on DD, PROTEINS, NCI109 and PTC, and competitive scores on NCI1 and MUTAG. In particular, U2GAN notably outperforms all baseline models on DD and PTC by large margins of 15+%. It is also to note that there are no significant differences between our unsupervised U2GAN and some supervised baselines (e.g., GFN, GIN and GCAPS) on NCI1. Besides, U2GAN obtains promising accuracies with those of baseline models on MUTAG. Note that there are only 188 graphs in this dataset, which explains the high variance in the results. Overall, our proposed U2GAN achieves state-of-the-art performances on a range of benchmarks against up-to-date supervised and unsupervised baseline models for the graph classification task.

Next we investigate the effects of hyper-parameters on the experimental datasets in Figure 3.[5] In general, our U2GAN 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. It is to note that 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.

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

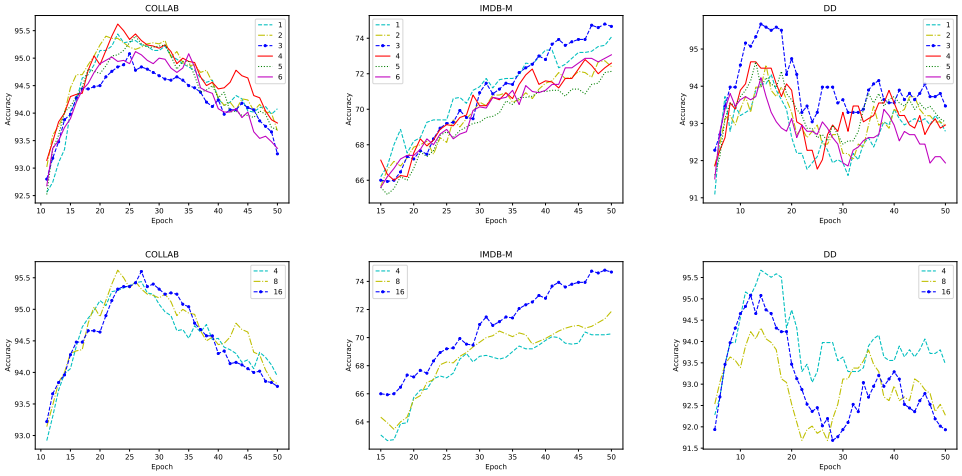

Figure 3: Effects of the number of steps (T) (in the 3 top figures), and the number of neighbors (N) sampled for each node (in the 3 bottom figures). For each dataset, for all 10 folds, we vary the value of either T or N while using the same fixed values of other hyper-parameters.

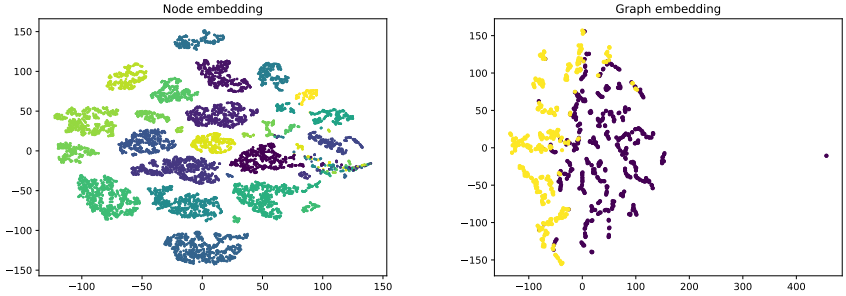

Figure 4: A visualization of the node and graph embeddings learned by U2GAN on the DD dataset.

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 on the DD dataset where the node labels are available. It can be seen from Figure 4 that our U2GAN 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 unsupervised embedding model U2GAN for the graph classification task. Inspired by the universal self-attention network, given an input graph, U2GAN applies a self-attention mechanism followed by a recurrent transition to learn node embeddings and then sums all learned node embeddings to obtain an embedding of the entire graph. We evaluate the performance of U2GAN on 11 well-known benchmark datasets, using the same 10-fold cross-validation scheme to compute the classification accuracies for a fair comparison, against up-to-date unsupervised and supervised baseline models. The experiments show that our U2GAN achieves new highest results on 8 out of 11 datasets and comparable results on the rest. In addition, U2GAN can be seen as a general framework where we can derive effective node and graph embeddings, as we plan to investigate the effectiveness of U2GAN on other important tasks such as node classification and link prediction in the future work. 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.

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 *Proceedings of the Fifth IEEE International Conference on Data Mining*, ICDM '05, 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. *International Conference on Learning Representations*, 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 *Proceedings of the 34th International Conference on Machine Learning*, pp. 1263–1272, 2017.

William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, 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 *Proceedings of the IEEE conference on computer vision and pattern recognition*, 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 *International Conference on Machine Learning*, pp. 2191–2200, 2018.

Sébastien Jean, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio. On using very large target vocabulary for neural machine translation. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 1–10, 2015.

Hisashi Kashima, Koji Tsuda, and Akihiro Inokuchi. Marginalized kernels between labeled graphs. In *Proceedings of the 20th International Conference on Machine Learning*, 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 *International Conference on Learning Representations*, 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 *Proceedings of the 31th International Conference on Machine Learning*, pp. 1188–1196, 2014.

Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated Graph Sequence Neural Networks. *International Conference on Learning Representations*, 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. *International Conference on Learning Representations*, 2019b.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, 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 *International conference on machine learning*, pp. 2014–2023, 2016.

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

Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. Dynamic routing between capsules. In *Advances in Neural Information Processing Systems*, 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 *Artificial Intelligence and Statistics*, 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 *Advances in Neural Information Processing Systems*, pp. 5998–6008, 2017.

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. *International Conference on Learning Representations*, 2019.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How Powerful Are Graph Neural Networks? *International Conference on Learning Representations*, 2019.

Pinar Yanardag and SVN Vishwanathan. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 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 *Proceedings of the 32Nd International Conference on Neural Information Processing Systems*, 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 *Thirty-Second AAAI Conference on Artificial Intelligence*, 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 APPENDIX

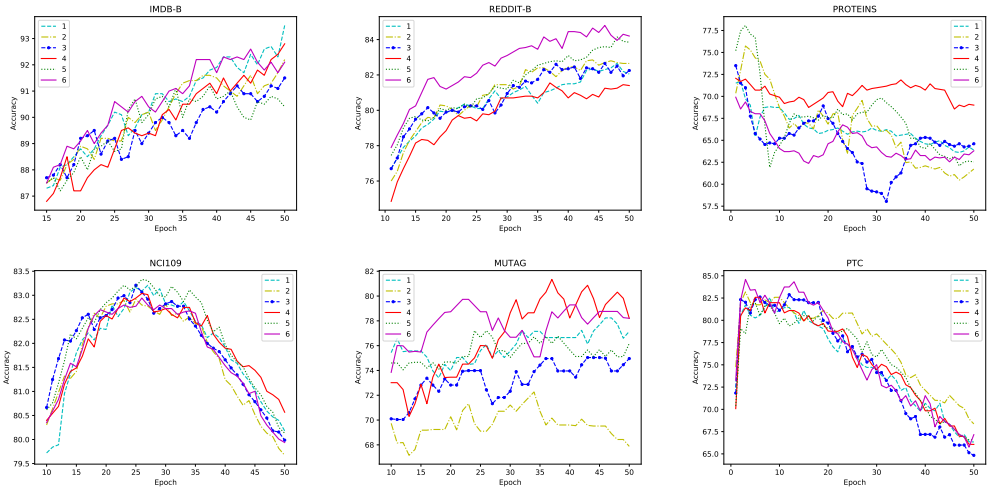

Figure 5: Effects of the number of steps (T). For each dataset, for all 10 folds, we vary the value of either T or N while using the same fixed values of other hyper-parameters.

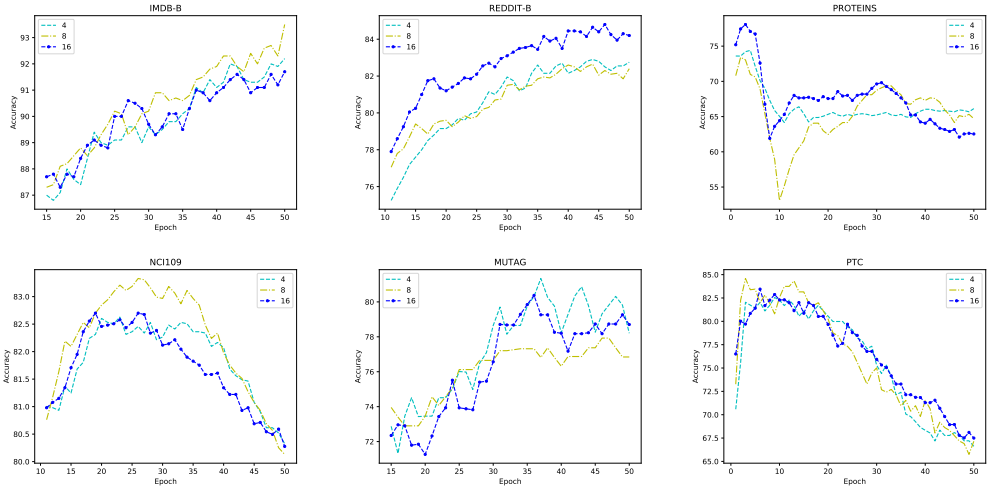

Figure 6: Effects of the number of neighbors (N) sampled for each node. For each dataset, for all 10 folds, we vary the value of either T or N while using the same fixed values of other hyper-parameters.