Universal Graph Transformer Self-Attention Networks

Dai Quoc Nguyen*   Tu Dinh Nguyen†   Dinh Phung‡

Abstract

The transformer self-attention network has been extensively used in research domains such as computer vision, image processing, and natural language processing. But it has not been actively used in graph neural networks (GNNs) where constructing an advanced aggregation function is essential. To this end, we present U2GNN, an effective GNN model leveraging a transformer self-attention mechanism followed by a recurrent transition, to induce a powerful aggregation function to learn graph representations. Experimental results show that the proposed U2GNN achieves state-of-the-art accuracies on well-known benchmark datasets for graph classification.[1]

Keywords: Graph neural networks · Graph classification · Transformer · Self-attention

1 Introduction

Graph-structured data appear in many real-world and scientific fields, e.g., knowledge graphs, recommender systems, social and citation networks, as well as telecommunication and biological networks [3, 56]. In general, we can view a graph as a network of nodes and edges, where nodes correspond to individual objects, and edges encode relationships among those objects. For example, in an online forum, each discussion thread can be constructed as a graph where nodes represent users, and edges represent commenting activities between users [49].

Learning graph representations is one of the most important topics for the graph-structured data [17, 57, 46, 52], where a goal is to construct vector representations for graphs to predict graph labels. Early approaches measure similarities among graphs to build a graph kernel for graph classification [14], wherein graph kernel-based approaches decompose graphs into "atomic subgraphs" (such as random walks [14, 22, 44], shortest paths [4], graphlets [39], and Weisfeiler-Lehman subtree patterns [38]) to have a vector of frequencies of atomic subgraphs for each given graph. Besides, word embedding-based approaches utilize Word2Vec [32] and Doc2Vec [26] to learn embeddings for the atomic subgraphs [49] and the entire graphs [33, 20].

Recently, graph neural networks (GNNs) become an essential strand to learn low-dimensional continuous embeddings of the entire graphs to predict the graph labels [36, 17, 57, 46, 52]. GNNs use an AGGREGATION function to update the vector representation of each node by transforming and aggregating the vector representations of its neighbors [24, 16, 42]. Then GNNs apply a graph-level pooling function (i.e., a READOUT operation such as simple sum pooling [48]) to obtain graph embeddings. Compared to the graph kernel and word embedding-based approaches, GNNs have been achieving state-of-the-art accuracies for the graph classification task.

To further improve the classification performance, constructing a powerful aggregation function is essential for GNNs as also discussed in [48]. Nowadays, there are novel applications of the transformer self-attention network [41, 9] recognized, published, and used successfully in research domains such as computer vision, image processing, and natural language processing. Hence we consider the use of the transformer to a new domain, i.e., graph neural networks (GNNs), as a novelty. Inspired by this advanced self-attention network, we present U2GNN, an effective GNN model, which induces a powerful AGGREGATION function leveraging a self-attention mechanism [41] followed by a recurrent transition, to update the vector representations of nodes from their neighbors. In particular, our U2GNN is different from related work as follows:

- A "concurrent" work – Hyper-SAGNN [55] – utilizes the transformer self-attention network for hypergraphs that have diverse and different structures, hence requiring a different solution. Besides, the later and closely related work, Graph-BERT [53], is an extension of our U2GNN for semi-supervised node classification.

- Graph Attention Network (GAT) [42] borrows

---
*Monash University, Australia, email: dai.nguyen@monash.edu

†nguyendinhtu@gmail.com

‡Monash University, Australia, email: dinh.phung@monash.edu

[1]Our code is available at: https://github.com/daiquocnguyen/Graph-Transformer.

the standard attention technique from [2] in using a single-layer feedforward neural network parametrized by a weight vector and then applying the non-linearity function followed by the softmax to compute the importance weights of neighbors of a given node. Note that our U2GNN adopts a scaled dot-product attention mechanism which is more robust and efficient than the attention technique used in GAT.

- Regarding the model architecture, Graph Transformer Network (GTN) [51] identifies useful meta-paths [45] to transform graph structures and applies GCN [24] to learn the node embeddings for the node classification task on heterogeneous graphs. Self-Attention Graph Pooling (SAGPool) [27] uses GCN as an attention mechanism to weight the nodes, employs a node selection method [13] to retain a portion of the nodes, and applies the existing graph-level READOUT pooling layers (consisting of global pooling [54] and hierarchical pooling [6]) to obtain the graph embeddings.

- To this end, we note that U2GNN is entirely different from GAT [42], Graph Transformer Network [51], and Self-Attention Graph Pooling [27], except similar titles.

Contributions. Our main contributions in this paper are as follows:

- We propose U2GNN, an effective GNN model, leveraging the transformer self-attention network to construct an advanced aggregation function to learn the graph representations.

- Experimental results show that U2GNN obtains state-of-the-art accuracies on well-known benchmark datasets for the graph classification task.

2 Related work

2.1 Graph kernel-based approaches These early approaches decompose graphs into "atomic subgraphs" (such as random walks [14, 22, 44], shortest paths [4], graphlets [39], and Weisfeiler-Lehman subtree patterns [38]) to measure the similarities among graphs [14]. Therefore, we can view each atomic substructure as a word term and each graph as a text document. We then represent a collection of graphs as a document-term matrix that describes the normalized frequency of terms in documents. We use an inner product to compute the graph similarities to derive a "kernel matrix" used for the kernel-based learning algorithms such as Support Vector Machines (SVM) [19] to measure the graph classification performance. We refer to an overview of the graph kernel-based approaches in [35, 25].

2.2 Word embedding-based approaches Since the introduction of word embedding models such as Word2Vec [32] and Doc2Vec [26], several works have used them for the graph classification task. Deep graph kernel (DGK) [49] applies Word2Vec to learn the embeddings of the atomic substructures (such as the graphlets, the WL subtree patterns, and the shortest paths) to derive the kernel matrix. Anonymous walk embedding (AWE) [20] maps random walks into "anonymous walks", views each anonymous walk as a word token, and utilizes Doc2Vec to achieve the graph embeddings to compute the graph similarities to construct the kernel matrix. Graph2Vec [33] employs Doc2Vec to learn the embeddings for the WL subtree patterns and to obtain the graph embeddings to train a SVM classifier.

2.3 Graph neural networks Recent works have focused on using graph neural networks (GNNs) to perform the graph classification task [36, 28, 34, 15, 54, 50, 43, 48]. In general, GNNs use an AGGREGATION function which aims to update the vector representation of each node by recursively propagating the vector representations of its neighbors [36, 24, 16, 42]. The AGGREGATION function can be a neural network such as gated recurrent units (GRU) [28] or multi-layer perceptrons (MLPs) [48]. Besides, GCN [24], GraphSAGE [16], and GAT [42] are also used as the AGGREGATION functions. GNNs then utilize a READOUT pooling function to obtain the graph embeddings, which are fed to multiple fully-connected layers followed by a softmax layer to predict the graph labels.

Regarding the aggregation of node representations, GCN [24] updates vector representation for a given node $\mathsf{v} \in \mathcal{V}$ from its neighbors, using multiple GCN layers stacked on top of each other to capture k-hops neighbors, as:

$$(2.1) \quad \mathbf{h}_\mathsf{v}^{(k+1)} = \mathsf{g}\left(\sum_{\mathsf{u} \in \mathcal{N}_\mathsf{v} \cup \{\mathsf{v}\}} a_{\mathsf{v},\mathsf{u}} \mathbf{W}^{(k)} \mathbf{h}_\mathsf{u}^{(k)}\right), \forall \mathsf{v} \in \mathcal{V}$$

where k is the layer index; $a_{\mathsf{v},\mathsf{u}}$ is an edge constant between nodes v and u in the re-normalized adjacency matrix; $\mathbf{W}^{(k)}$ is a weight matrix; $\mathbf{h}_\mathsf{u}^{(0)}$ is a feature vector of node u; g is a non-linear activation function; and $\mathcal{N}_\mathsf{v}$ is the set of neighbors of node v.

GraphSAGE [16] extends GCN to use a node-wise procedure of uniformly sampling a fixed number of neighbors for each node at each layer as:

$$(2.2) \quad \mathbf{h}_\mathsf{v}^{(k+1)} = \mathsf{g}\left(\mathbf{W}^{(k)}\left[\mathbf{h}_\mathsf{v}^{(k)}; \mathbf{h}_{\mathcal{N}_\mathsf{v}'}^{(k)}\right]\right), \forall \mathsf{v} \in \mathcal{V}$$

where [;] denotes a vector concatenation, and $\mathbf{h}_{\mathcal{N}_\mathsf{v}'}^{(k)}$ can be obtained using an element-wise max-pooling operation

as:

$$(2.3) \quad \mathbf{h}_{\mathcal{N}_v'}^{(k)} = \mathsf{max}\left(\left\{\mathsf{g}\left(\mathbf{W}_{pool}\mathbf{h}_u^{(k)} + \mathbf{b}\right), \forall u \in \mathcal{N}_v'\right\}\right)$$

where $\mathcal{N}_v'$ is defined as a fixed-size, uniformly sampled from $\mathcal{N}_v$ of v. Besides, $\mathcal{N}_v'$ is sampled differently through each layer.

Graph Attention Network (GAT) [42] extends GCN to compute edge weights following the standard attention technique [2] as:

$$(2.4) \quad \mathbf{h}_v^{(k+1)} = \mathsf{g}\left(\sum_{u \in \mathcal{N}_v \cup \{v\}} \tau_{v,u}^{(k)} \mathbf{W}^{(k)}\mathbf{h}_u^{(k)}\right), \forall v \in \mathcal{V}$$

where $\tau_{v,u}^{(k)}$ is an edge weight between nodes v and u, which is computed as: $\tau_{v,u}^{(k)} =$

$$(2.5)$$
$$\mathsf{softmax}\left(\mathsf{LeakyReLU}\left(\boldsymbol{a}^{(k)\mathsf{T}}\left[\mathbf{W}^{(k)}\mathbf{h}_v^{(k)}; \mathbf{W}^{(k)}\mathbf{h}_u^{(k)}\right]\right)\right)$$

3 U2GNN: Universal Graph Transformer Self-Attention Networks

In this section, we present the additional background of graph neural networks and detail our proposed U2GNN.

3.1 Graph classification

We represent each graph $\mathcal{G} = \left(\mathcal{V}, \mathcal{E}, \{\mathbf{h}_v^{(0)}\}_{v \in \mathcal{V}}\right)$, where $\mathcal{V}$ is a set of nodes, $\mathcal{E}$ is a set of edges, and $\mathbf{h}_v^{(0)} \in \mathbb{R}^d$ represents the feature vector of node $v \in \mathcal{V}$. Given a set of M graphs $\{\mathcal{G}_m\}_{m=1}^M$ and their corresponding class labels $\{y_m\}_{m=1}^M \subseteq \mathcal{Y}$, the graph classification task is to learn an embedding $\mathbf{e}_{\mathcal{G}_m}$ for each graph $\mathcal{G}_m$ to predict its label y_m.

3.2 Graph neural networks

In general, GNNs aim to update the vector representation of each node by recursively aggregating and transforming the vector representations of its neighbors [24, 16, 42]. After that, GNNs use a READOUT pooling function to obtain the vector representations of the entire graphs [15, 54, 50, 43, 48]. Mathematically, given a graph $\mathcal{G}$, we can formalize GNNs as follows:

$$(3.6) \quad \mathbf{h}_v^{(k+1)} = \textsc{Aggregation}\left(\left\{\mathbf{h}_u^{(k)}\right\}_{u \in \mathcal{N}_v \cup \{v\}}\right)$$

$$(3.7) \quad \mathbf{e}_{\mathcal{G}} = \textsc{ReadOut}\left(\left\{\left\{\mathbf{h}_v^{(k)}\right\}_{k=0}^K\right\}_{v \in \mathcal{V}}\right)$$

where $\mathbf{h}_v^{(k)}$ is the vector representation of node v at the k-th iteration/layer, $\mathcal{N}_v$ is the set of neighbors of v.

Many methods have been proposed to construct the AGGREGATION functions, e.g., GCN [24], GraphSAGE

[16], and GAT [42]. Recently, Graph Isomorphism Network (GIN-0) [48] uses a more powerful AGGREGATION function based on a multi-layer perceptron (MLP) network of two fully-connected layers as:

$$(3.8) \quad \mathbf{h}_v^{(k+1)} = \mathsf{MLP}^{(k)}\left(\sum_{u \in \mathcal{N}_v \cup \{v\}} \mathbf{h}_u^{(k)}\right), \forall v \in \mathcal{V}$$

Following [48], we employ a concatenation over the vector representations of node v at the different layers to construct a vector representation $\mathbf{e}_v$ for each node v as:

$$(3.9) \quad \mathbf{e}_v = \left[\mathbf{h}_v^{(1)}; \mathbf{h}_v^{(2)}; ...; \mathbf{h}_v^{(K)}\right], \forall v \in \mathcal{V}$$

where K is the index of the last layer. The graph-level READOUT function can be a simple sum pooling [48] or a complicated pooling such as hierarchical pooling [6], and differentiable pooling [50]. As the sum pooling produces competitive results [48], we use the sum pooling to obtain the embedding $\mathbf{e}_{\mathcal{G}}$ of the entire graph $\mathcal{G}$ as:

$$(3.10) \quad \mathbf{e}_{\mathcal{G}} = \sum_{v \in \mathcal{V}} \mathbf{e}_v = \sum_{v \in \mathcal{V}} \left[\mathbf{h}_v^{(1)}; \mathbf{h}_v^{(2)}; ...; \mathbf{h}_v^{(K)}\right]$$

After that, we can follow [48] to feed the graph embeddings $\mathbf{e}_{\mathcal{G}}$ to a single fully-connected layer followed by a softmax layer to predict the graph labels.

3.3 The proposed U2GNN

The transformer self-attention network [41, 9] has widely applied as a novelty in research domains such as computer vision and NLP. Similarity, we also consider the successful use of this recent advanced technique to a new domain, i.e., graph neural networks (GNNs), as a novel application. Moreover, as also discussed in [48], constructing an powerful AGGREGATION mechanism is essential for GNNs. To this end, we induce an advanced AGGREGATION function, using the universal transformer network [9] consisting of a self-attention mechanism [41] followed by a recurrent transition (TRANS) with adding residual connections [18] and layer normalization (LNORM) [1], as illustrated in Figure 1.

The residual connections [18] are used to add useful information learned in the lower layers to the higher layers, and more importantly, to allow gradients to directly pass through the layers to avoid vanishing gradient or exploding gradient problems. The layer normalization (LNORM) [1] is used to normalize the inputs across the feature dimensions to stabilize the network to enable smoother gradients and faster training. The residual connections and the layer normalization are commonly used in many architectures and thus are omitted in the paper for simplicity.
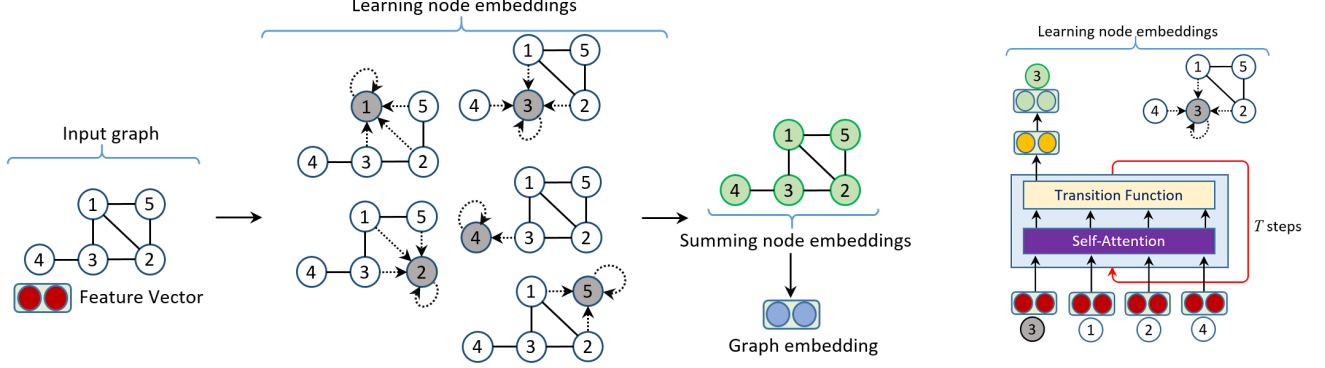
Figure 1: Illustration of our U2GNN.

Formally, given an input graph $\mathcal{G}$, we uniformly sample a set $\mathcal{N}_{\mathsf{v}}$ of neighbors for each $\mathsf{v} \in \mathcal{V}$ and then input $\mathcal{N}_{\mathsf{v}} \cup \{\mathsf{v}\}$ to the U2GNN learning process. Note that we sample a different $\mathcal{N}_{\mathsf{v}}$ for node v at each training batch. We also construct multiple layers stacked on top of each other in our U2GNN. Regarding the k-th layer, given a node $\mathsf{v} \in \mathcal{V}$, at each step t, we induce a transformer self-attention-based function to aggregate the vector representations for all nodes $\mathsf{u} \in \mathcal{N}_{\mathsf{v}} \cup \{\mathsf{v}\}$ as:

$$
(3.11) \qquad \mathbf{h}_{t,\mathsf{u}}^{(k)} = \text{TRANSFORMER-AGGREGATION}\left(\mathbf{h}_{t-1,\mathsf{u}}^{(k)}\right)
$$

In particular,

$$
(3.12) \qquad \mathbf{x}_{t,\mathsf{u}}^{(k)} = \text{LNORM}\left(\mathbf{h}_{t-1,\mathsf{u}}^{(k)} + \text{ATT}\left(\mathbf{h}_{t-1,\mathsf{u}}^{(k)}\right)\right)
$$

then,

$$
(3.13) \qquad \mathbf{h}_{t,\mathsf{u}}^{(k)} = \text{LNORM}\left(\mathbf{x}_{t,\mathsf{u}}^{(k)} + \text{TRANS}\left(\mathbf{x}_{t,\mathsf{u}}^{(k)}\right)\right)
$$

where $\mathbf{h}_{t,\mathsf{u}}^{(k)} \in \mathbb{R}^d$; TRANS(.) and ATT(.) denote a MLP network (i.e., two fully-connected layers) and a self-attention network respectively:

$$
(3.14)
$$
$$
\text{TRANS}\left(\mathbf{x}_{t,\mathsf{u}}^{(k)}\right) = \mathbf{W}_2^{(k)}\text{ReLU}\left(\mathbf{W}_1^{(k)}\mathbf{x}_{t,\mathsf{u}}^{(k)} + \mathbf{b}_1^{(k)}\right) + \mathbf{b}_2^{(k)}
$$

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

$$
(3.15) \quad \text{ATT}\left(\mathbf{h}_{t-1,\mathsf{u}}^{(k)}\right) = \sum_{\mathsf{u}' \in \mathcal{N}_{\mathsf{v}} \cup \{\mathsf{v}\}} \alpha_{\mathsf{u},\mathsf{u}'}^{(k)}\left(\mathbf{V}^{(k)}\mathbf{h}_{t-1,\mathsf{u}'}^{(k)}\right)
$$

where $\mathbf{V}^{(k)} \in \mathbb{R}^{d \times d}$ is a value-projection weight matrix; $\alpha_{\mathsf{u},\mathsf{u}'}$ is an attention weight, which is computed using the softmax function over scaled dot products between nodes u and u':

$$
(3.16)
$$
$$
\alpha_{\mathsf{u},\mathsf{u}'}^{(k)} = \text{softmax}\left(\frac{\left(\mathbf{Q}^{(k)}\mathbf{h}_{t-1,\mathsf{u}}^{(k)}\right) \cdot \left(\mathbf{K}^{(k)}\mathbf{h}_{t-1,\mathsf{u}'}^{(k)}\right)}{\sqrt{d}}\right)
$$

where $\mathbf{Q}^{(k)} \in \mathbb{R}^{d \times d}$ and $\mathbf{K}^{(k)} \in \mathbb{R}^{d \times d}$ are query-projection and key-projection matrices, respectively.

After T steps, we feed $\mathbf{h}_{T,\mathsf{v}}^{(k)} \in \mathbb{R}^d$ to the next $(k{+}1)$-th layer as:

$$
(3.17) \qquad \mathbf{h}_{0,\mathsf{v}}^{(k+1)} = \mathbf{h}_{T,\mathsf{v}}^{(k)}, \forall \mathsf{v} \in \mathcal{V}
$$

Note that $\mathbf{h}_{0,\mathsf{v}}^{(0)} = \mathbf{h}_{\mathsf{v}}^{(0)} \in \mathbb{R}^d$ is the feature vector of node v.

We apply the vector concatenation across the layers to obtain the vector representations $\mathbf{e}_{\mathsf{v}}$ of nodes v following Equation 3.9 as:

$$
(3.18) \qquad \mathbf{e}_{\mathsf{v}} = \left[\mathbf{h}_{0,\mathsf{v}}^{(1)}; \mathbf{h}_{0,\mathsf{v}}^{(2)}; ...; \mathbf{h}_{0,\mathsf{v}}^{(K)}\right], \forall \mathsf{v} \in \mathcal{V}
$$

where K is the number of layers. We use $\mathbf{e}_{\mathsf{v}}$ as the final embedding of node $\mathsf{v} \in \mathcal{V}$ and then sum all the final embeddings of nodes in $\mathcal{G}$ to get the final embedding $\mathbf{e}_{\mathcal{G}}$ of the entire graph $\mathcal{G}$. We feed $\mathbf{e}_{\mathcal{G}}$ to a single fully-connected layer followed by a softmax layer to predict the graph label as:

$$
(3.19) \qquad \hat{\mathbf{y}}_{\mathcal{G}} = \text{softmax}\left(\mathbf{W}\mathbf{e}_{\mathcal{G}} + \mathbf{b}\right)
$$

Finally, we learn the model parameters by minimizing the cross-entropy loss function. To sum up, we briefly present the learning process of our proposed U2GNN in Algorithm 1.

3.4 Discussion We discuss some findings in our proposed U2GNN as follows:

Algorithm 1 The U2GNN learning process.

Input: $\mathcal{G} = \left(\mathcal{V}, \mathcal{E}, \{\mathbf{h}_v^{(0)}\}_{v \in \mathcal{V}} \right)$ with its label y

for $k = 0, 1, ..., K-1$ **do**
    for $v \in \mathcal{V}$ **do**
        SAMPLE $\mathcal{N}_v$ for v
        for $t = 1, 2, ..., T$ **do**
            $\forall u \in \mathcal{N}_v \cup \{v\}$
            $\mathbf{x}_{t,u}^{(k)} \leftarrow \text{LNORM}\left(\mathbf{h}_{t-1,u}^{(k)} + \text{ATT}\left(\mathbf{h}_{t-1,u}^{(k)} \right) \right)$
            $\mathbf{h}_{t,u}^{(k)} \leftarrow \text{LNORM}\left(\mathbf{x}_{t,u}^{(k)} + \text{TRANS}\left(\mathbf{x}_{t,u}^{(k)} \right) \right)$
        $\mathbf{h}_{0,v}^{(k+1)} \leftarrow \mathbf{h}_{T,v}^{(k)} \in \mathbb{R}^d$

$\mathbf{e}_v \leftarrow \left[\mathbf{h}_{0,v}^{(1)}; \mathbf{h}_{0,v}^{(2)}; ...; \mathbf{h}_{0,v}^{(K)} \right], \forall v \in \mathcal{V}$ (w.r.t Equation 3.18)
$\mathbf{e}_{\mathcal{G}} \leftarrow \sum_{v \in \mathcal{V}} \mathbf{e}_v$
$\mathsf{y} \leftarrow \mathsf{softmax}\left(\mathbf{W}\mathbf{e}_{\mathcal{G}} + \mathbf{b} \right)$

- If we set T to 1, $\alpha_{u,u'}^{(k)}$ to 1, $\boldsymbol{V}^{(k)}$ to the identity matrix in Equation 3.15, and do not use both the residual connections and the layer normalization, we simplify our U2GNN aggregation function (from Equations 3.17 and 3.13) as:

$$
\begin{aligned}
\mathbf{h}_{1,v}^{(k+1)} &= \text{TRANS}\left(\text{ATT}\left(\mathbf{h}_{0,v}^{(k+1)} \right) \right) \\
&= \text{TRANS}\left(\sum_{u \in \mathcal{N}_v \cup \{v\}} \mathbf{h}_{0,u}^{(k+1)} \right) \\
(3.20) \quad &= \text{TRANS}\left(\sum_{u \in \mathcal{N}_v \cup \{v\}} \mathbf{h}_{1,u}^{(k)} \right)
\end{aligned}
$$

where TRANS(.) denotes the MLP network of two fully-connected layers (as defined in Equation 3.14). ==Thus, this implies that our U2GNN can be simplified (w.r.t Equation 3.20) to be equivalent to Graph Isomorphism Network (GIN-0) [48] (w.r.t Equation 3.8) – one of the recent state-of-the-art GNNs.== Experimental results presented in Section 5 show that U2GNN outperforms GIN-0 on benchmark datasets for the graph classification task.

- We probably could construct a complex architecture using a complicated graph-level pooling (such as hierarchical pooling [6]) followed by multiple fully-connected layers [7, 31] to predict the graph labels. However, we refrained from doing that, as our key purpose is to introduce a single, unified and effective model that can work well and produce competitive performances on the benchmark datasets. Therefore, using the sum pooling followed by a single fully-connected layer is reasonable for our U2GNN.

- As established empirically, our results shown in Section 5 imply that the U2GNN self-attention-based aggregation function is a powerful computation process compared to other existing functions.

4 Experimental setup

We use seven well-known datasets consisting of three social network datasets (COLLAB, IMDB-B, and IMDB-M) and four bioinformatics datasets (DD, MUTAG, PROTEINS, and PTC). The social network datasets do not have available node features; thus, we follow [34, 54] to use node degrees as features. Table 1 reports the statistics of these datasets.

Table 1: Statistics of the experimental benchmark datasets. **#G** denotes the numbers of graphs. **#Cls** denotes the number of class labels. **Avg#N** denotes the average number of nodes per graph. **Avg#E** denotes the average number of neighbors per node. d is the dimension of feature vectors. Note that d is also equal to the node embedding size at each U2GNN layer.

Dataset	#G	#Cls	Avg#N	Avg#E	d
COLLAB	5,000	3	74.5	65.9	–
IMDB-M	1,500	3	13.0	10.1	–
IMDB-B	1,000	2	19.8	9.8	–
DD	1,178	2	284.3	5.0	82
PROTEINS	1,113	2	39.1	3.7	3
PTC	344	2	25.6	2.0	19
MUTAG	188	2	17.9	2.2	7

- **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; each graph is labeled to a physics field that the researcher belongs 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; nodes correspond to actors/actresses, and each edge represents a co-appearance of two actors/actresses in the same movie; each graph is assigned to a genre.

- **Bioinformatics datasets:** DD [10] 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. MUTAG [8] is a collection of 188 nitro compound networks with 7 discrete node labels, where classes indicate a mutagenic effect on a bacterium. PROTEINS comprises

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

Model	COLLAB	IMDB-B	IMDB-M	DD	PROTEINS	MUTAG	PTC
GK [39]	72.84 ± 0.28	65.87 ± 0.98	43.89 ± 0.38	78.45 ± 0.26	71.67 ± 0.55	81.58 ± 2.11	57.26 ± 1.41
WL [38]	79.02 ± 1.77	73.40 ± 4.63	49.33 ± 4.75	79.78 ± 0.36	74.68 ± 0.49	82.05 ± 0.36	57.97 ± 0.49
PSCN [34]	72.60 ± 2.15	71.00 ± 2.29	45.23 ± 2.84	77.12 ± 2.41	75.89 ± 2.76	**92.63 ± 4.21**	62.29 ± 5.68
GCN [24]	**81.72 ± 1.64**	73.30 ± 5.29	51.20 ± 5.13	79.12 ± 3.07	75.65 ± 3.24	87.20 ± 5.11	–
GFN [7]	81.50 ± 2.42	73.00 ± 4.35	51.80 ± 5.16	78.78 ± 3.49	76.46 ± 4.06	90.84 ± 7.22	–
GraphSAGE [16]	79.70 ± 1.70	72.40 ± 3.60	49.90 ± 5.00	65.80 ± 4.90	65.90 ± 2.70	79.80 ± 13.9	–
GAT [42]	75.80 ± 1.60	70.50 ± 2.30	47.80 ± 3.10		74.70 ± 2.20	89.40 ± 6.10	66.70 ± 5.10
DGCNN [54]	73.76 ± 0.49	70.03 ± 0.86	47.83 ± 0.85	79.37 ± 0.94	75.54 ± 0.94	85.83 ± 1.66	58.59 ± 2.47
SAGPool [27]	–	–	–	76.45 ± 0.97	71.86 ± 0.97	–	–
PPGN [30]	81.38 ± 1.42	73.00 ± 5.77	50.46 ± 3.59	–	77.20 ± 4.73	90.55 ± 8.70	66.17 ± 6.54
CapsGNN [47]	79.62 ± 0.91	73.10 ± 4.83	50.27 ± 2.65	75.38 ± 4.17	76.28 ± 3.63	86.67 ± 6.88	–
DSGC [37]	79.20 ± 1.60	73.20 ± 4.90	48.50 ± 4.80	77.40 ± 6.40	74.20 ± 3.80	86.70 ± 7.60	–
GIN-0 [48]	80.20 ± 1.90	75.10 ± 5.10	52.30 ± 2.80	–	76.20 ± 2.80	89.40 ± 5.60	64.60 ± 7.00
GCAPS [43]	77.71 ± 2.51	71.69 ± 3.40	48.50 ± 4.10	77.62 ± 4.99	76.40 ± 4.17		66.01 ± 5.91
IEGN [31]	77.92 ± 1.70	71.27 ± 4.50	48.55 ± 3.90	–	75.19 ± 4.30	84.61 ± 10.0	59.47 ± 7.30
U2GNN	77.84 ± 1.48	**77.04 ± 3.45**	**53.60 ± 3.53**	**80.23 ± 1.48**	**78.53 ± 4.07**	89.97 ± 3.65	**69.63 ± 3.60**

1,113 graphs obtained from [5] to present secondary structure elements (SSEs). PTC [40] consists of 344 chemical compound networks with 19 discrete node labels where classes show carcinogenicity for male and female rats.

4.1 Training protocol We vary the number K of U2GNN layers in $\{1, 2, 3\}$, the number of steps T in $\{1, 2, 3, 4\}$, the number of neighbors ($|\mathcal{N}_v| = N$) sampled for each node in $\{4, 8, 16\}$, and the dimension s of TRANS(.) (in Equation 3.14) in $\{128, 256, 512, 1024\}$ (in Equation 3.14). We set the batch size to 4. We apply the Adam optimizer [23] to train our U2GNN and select the Adam initial learning rate $lr \in \{5e^{-5}, 1e^{-4}, 5e^{-4}, 1e^{-3}\}$. We run up to 50 epochs to evaluate our U2GNN.

4.2 Evaluation protocol We follow [48, 47, 30, 37, 7] to use the same data splits and the same 10-fold cross-validation scheme to calculate the classification performance for a fair comparison.

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

- **Unsupervised approaches:** Graphlet Kernel (GK) [39] and Weisfeiler-Lehman kernel (WL) [38].

- **Supervised approaches:** PATCHY-SAN (PSCN) [34], Graph Convolutional Network (GCN) [24], GraphSAGE [16], Graph Attention Network (GAT) [42], Deep Graph CNN (DGCNN)

[54], Graph Capsule Convolution Neural Network (GCAPS) [43], Capsule Graph Neural Network (CapsGNN) [47], Self-Attention Graph Pooling (SAGPool) [27], Graph Isomorphism Network (GIN-0) [48], Graph Feature Network (GFN) [7], Invariant-Equivariant Graph Network (IEGN) [31], Provably Powerful Graph Network (PPGN) [30], and Discriminative Structural Graph Classification (DSGC) [37].

We report the baselines taken from the original papers or published in [20, 43, 47, 12, 7, 37].

5 Experimental results

Table 2 presents the experimental results of U2GNN and other strong baseline models for the benchmark datasets, and Figure 3 shows the classification accuracies of our proposed U2GNN across 10 folds for each dataset. On the social network datasets, our U2GNN produces new state-of-the-art performances on IMDB-B and IMDB-M and gains a competitive score on COLLAB. Especially, U2GNN obtains 4+% absolute higher accuracies than all the supervised baseline models on IMDB-B and IMDB-M. Regarding COLLAB, we found the best results of GCN in [7], where, after obtaining the graph embeddings, [7] utilized two fully-connected layers to predict the graph labels for GCN and GFN. This is different from our U2GNN where we used a single fully-connected layer as we discussed in Section 3.4.

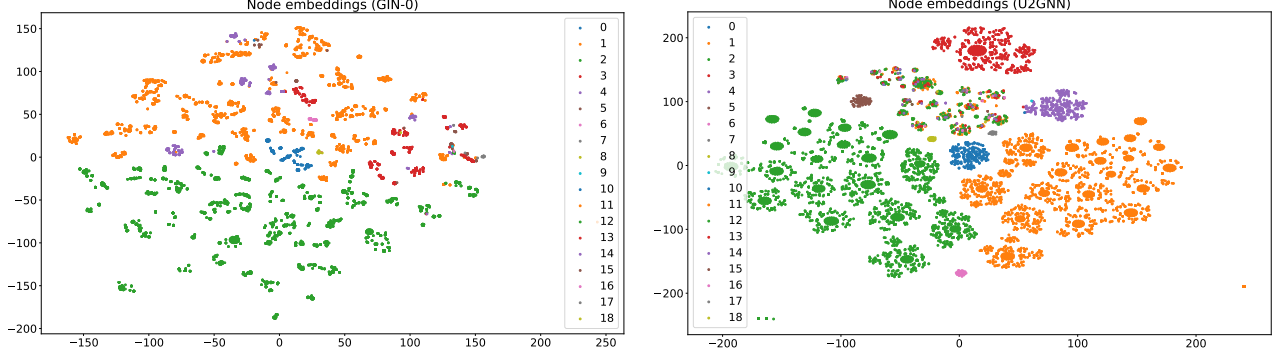On the bioinformatics datasets, our U2GNN obtains the highest accuracies on DD, PROTEINS, and PTC.

Figure 2: A t-SNE visualization of the node embeddings learned by GIN-0 and our U2GNN on the PTC dataset.

Moreover, U2GNN achieves a competitive accuracy compared with those of the baseline models on MUTAG. Additionally, there are no significant differences between our U2GNN and the baselines on MUTAG as this dataset only consists of 188 graphs, which explains the high variance in the results.
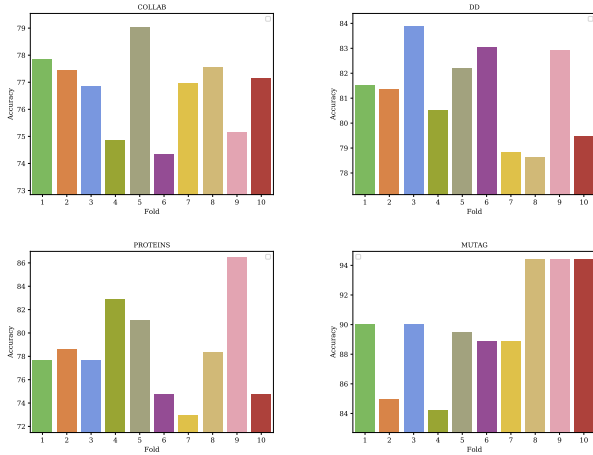

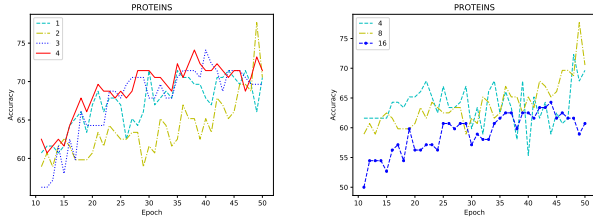Figure 3: Classification accuracies across 10 folds for each dataset.

Figure 4: Effects of the number of timesteps (T), and the number of neighbors sampled for each node ($N = |\mathcal{N}_v|$).

Hyper-parameter analysis. We investigate the effects of the number of timesteps (T) and the number of neighbors sampled for each node ($N = |\mathcal{N}_v|$) in Figure 4. In general, we find that higher T can help on most of the datasets as we may use more steps T to encode the graph structures better. Furthermore, the social network datasets are denser than the bioinformatics ones; hence we should use more sampled neighbors (i.e., using higher N) on the social network datasets rather than on the bioinformatics ones.

Visualization. To qualitatively demonstrate the effectiveness of our U2GNN, we use t-SNE [29] to visualize the node embeddings learned by GIN-0 and our U2GNN on PTC where the node labels are available. Compared to GIN-0, Figure 2 shows that our U2GNN produces a higher quality of learned node embeddings wherein the nodes are well-clustered according to the node labels.

In general, the superior performance of our method over the up-to-date baselines (especially, GIN-0) indicates that the U2GNN self-attention-based aggregation function is an advanced computation process to improve the classification performance of GNNs.

6   Conclusion

We introduce U2GNN, an advantageous graph neural network model. U2GNN induces a powerful aggregation function leveraging the transformer self-attention network to improve the graph classification performance. We evaluate our U2GNN using the same data splits and the same 10-fold cross-validation scheme on well-known benchmark datasets. Experimental results demonstrate that U2GNN outperforms up-to-date models and produces state-of-the-art accuracies on these datasets. [2]

---

[2]We propose a new unsupervised learning in Appendix A to train GNNs and hope that future GNN works can consider the unsupervised learning beside the supervised one.

References

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

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *International Conference on Learning Representations (ICLR)*, 2015.

[3] 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:1806.01261*, 2018.

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

[5] 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.

[6] Cătălina Cangea, Petar Veličković, Nikola Jovanović, Thomas Kipf, and Pietro Liò. Towards sparse hierarchical graph classifiers. *arXiv preprint arXiv:1811.01287*, 2018.

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

[8] 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.

[9] Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal Transformers. *International Conference on Learning Representations (ICLR)*, 2019.

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

[11] 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.

[12] Xiaolong Fan, Maoguo Gong, Yu Xie, Fenlong Jiang, and Hao Li. Structured self-attention architecture for graph-level representation learning. *Pattern Recognition*, 100, 2019.

[13] Hongyang Gao and Shuiwang Ji. Graph u-nets. In *International Conference on Machine Learning*, pages 2083–2092, 2019.

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

[15] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural Message Passing for Quantum Chemistry. In *International Conference on Machine Learning*, pages 1263–1272, 2017.

[16] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1024–1034, 2017.

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

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

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

[20] Sergey Ivanov and Evgeny Burnaev. Anonymous walk embeddings. In *International Conference on Machine Learning*, pages 2191–2200, 2018.

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

[22] Hisashi Kashima, Koji Tsuda, and Akihiro Inokuchi. Marginalized kernels between labeled graphs. In *International Conference on Machine Learning*, pages 321–328, 2003.

[23] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations (ICLR)*, 2015.

[24] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.

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

[26] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *International Conference on Machine Learning*, pages 1188–1196, 2014.

[27] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. Self-attention graph pooling. In *International Conference on Machine Learning*, 2019.

[28] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated Graph Sequence Neural Networks. *International Conference on Learning Representations (ICLR)*, 2016.

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

[30] Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. Provably powerful graph networks. In *Advances in Neural Information Processing Systems*, pages 2153–2164, 2019.

[31] Haggai Maron, Heli Ben-Hamu, Nadav Shamir, and Yaron Lipman. Invariant and equivariant graph networks. *International Conference on Learning Representations (ICLR)*, 2019.

[32] 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*, pages 3111–3119, 2013.

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

[34] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning Convolutional Neural Networks for Graphs. In *International Conference on Machine Learning*, pages 2014–2023, 2016.

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

[36] 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.

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

[38] 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.

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

[40] 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.

[41] 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*, pages 5998–6008, 2017.

[42] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. *International Conference on Learning Representations (ICLR)*, 2018.

[43] Saurabh Verma and Zhi-Li Zhang. Graph capsule convolutional neural networks. *The Joint ICML and IJCAI Workshop on Computational Biology*, 2018.

[44] 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.

[45] Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S Yu. Heterogeneous graph attention network. In *The World Wide Web Conference*, pages 2022–2032, 2019.

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

[47] Zhang Xinyi and Lihui Chen. Capsule Graph Neural Network. *International Conference on Learning Representations (ICLR)*, 2019.

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

[49] Pinar Yanardag and SVN Vishwanathan. Deep graph kernels. In *The ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 1365–1374, 2015.

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

[51] Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J Kim. Graph transformer networks. In *Advances in Neural Information Processing Systems*, pages 11960–11970, 2019.

[52] Daokun Zhang, Jie Yin, Xingquan Zhu, and Chengqi Zhang. Network representation learning: A survey. *IEEE Transactions on Big Data*, 6:3–28, 2020.

[53] Jiawei Zhang, Haopeng Zhang, Congying Xia, and Li Sun. Graph-bert: Only attention is needed for learning graph representations. *arXiv preprint arXiv:2001.05140*, 2020.

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

[55] Ruochi Zhang, Yuesong Zou, and Jian Ma. Hyper-SAGNN: a self-attention based graph neural network for hypergraphs. In *International Conference on Learning Representations (ICLR)*, 2020.

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

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

A Unsupervised Graph Neural Networks

The unsupervised learning is essential in both industry and academic applications, where expanding unsupervised GNN models is more suitable due to the limited availability of class labels. Therefore, we introduce a new unsupervised learning to train GNNs for the graph classification task.

A.1 Learning process Most of the recent approaches have focused on the supervised learning where they use the graph labels during the training process [47, 48, 7, 31, 37]. In a situation where no graph la-

Table 3: Graph classification results (% accuracy) in the unsupervised learning. The best scores are in bold. uGCN denotes our unsupervised GCN. Note that we do not make any direct comparison between the *unsupervised* approaches and the *supervised* ones because of the difference in the training data.

Model	COLLAB	IMDB-B	IMDB-M	DD	PROTEINS	MUTAG	PTC
DGK [49]	73.09 ± 0.25	66.96 ± 0.56	44.55 ± 0.52	73.50 ± 1.01	75.68 ± 0.54	87.44 ± 2.72	60.08 ± 2.55
AWE [20]	73.93 ± 1.94	74.45 ± 5.83	51.54 ± 3.61	71.51 ± 4.02	–	87.87 ± 9.76	–
uGCN	93.28 ± 0.99	94.50 ± 2.79	81.66 ± 3.16	94.31 ± 1.71	$\mathbf{89.09 \pm 3.25}$	$\mathbf{95.36 \pm 2.64}$	$\mathbf{92.67 \pm 4.60}$
U2GNN	$\mathbf{95.62 \pm 0.92}$	$\mathbf{96.41 \pm 1.94}$	$\mathbf{89.20 \pm 2.52}$	$\mathbf{95.67 \pm 1.89}$	80.01 ± 3.21	88.47 ± 7.13	91.81 ± 6.61

bels are available during training, some works (such as DGK [49], Graph2Vec [33], and AWE [20]) have considered the unsupervised learning, where *they can have access to all nodes from the entire dataset (i.e., additionally using all nodes in the test set during training)*. But they produce lower classification accuracies compared to the supervised approaches.

Algorithm 2 The unsupervised learning.

Input: $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \{\mathbf{h}_v\}_{v \in \mathcal{V}})$

for $k = 0, ..., K-1$ **do**

    for $v \in \mathcal{V}$ **do**

        $\mathbf{h}_v^{(k+1)} = \textsc{Aggregation}\left(\left\{\mathbf{h}_u^{(k)}\right\}_{u \in \mathcal{N}_v \cup \{v\}}\right)$

$\mathbf{e}_v \leftarrow \left[\mathbf{h}_v^{(1)}; \mathbf{h}_v^{(2)}; ...; \mathbf{h}_v^{(K)}\right], \forall v \in \mathcal{V}$ (with respect to Equation 3.9)

$\mathbf{o}_v \leftarrow \mathbf{e}_v$ (with respect to Equation A.1)

$\mathbf{e}_{\mathcal{G}} \leftarrow \sum_{v \in \mathcal{V}} \mathbf{o}_v$

To this end, we propose a new unsupervised learning to train GNNs for the graph classification task. We can see $\mathbf{e}_v$ in Equation 3.9 as a vector representation encoded for the substructure around node v. The goal of our unsupervised learning is to guide GNNs to recognize and distinguish the substructures within each graph, leading to improve the classification accuracies of unsupervised models. To achieve this goal, we consider a final embedding $\mathbf{o}_v$ for each node v, and make the similarity between $\mathbf{e}_v$ and $\mathbf{o}_v$ higher than that between $\mathbf{e}_v$ and the final embeddings of the other nodes, by minimizing the sampled softmax loss function [21] applied to node v as:

$$(A.1) \qquad \mathcal{L}_{\mathsf{U2GNN}}(v) = -\log \frac{\exp(\mathbf{o}_v \cdot \mathbf{e}_v)}{\sum_{v' \in \mathcal{V}'} \exp(\mathbf{o}_{v'} \cdot \mathbf{e}_v)}$$

where $\mathcal{V}'$ is a subset sampled from $\{\cup \mathcal{V}_m\}_{m=1}^M$. Node embeddings $\mathbf{o}_v$ are learned implicitly as model parameters. After that, we sum all the final embeddings $\mathbf{o}_v$ of nodes v in $\mathcal{G}$ to obtain the graph embedding $\mathbf{e}_{\mathcal{G}}$. We then use the logistic regression classifier [11] with setting the termination criterion to 0.001 to evaluate our

model. We outline the general process of our unsupervised learning in Algorithm 2.

A.2 Training protocol We follow some unsupervised approaches such as DGK [49] and AWE [20] to train our unsupervised U2GNN on all nodes from the entire dataset (i.e., consisting of all nodes from the test set during training) for a fair comparison. The hyperparameters are varied as same as in Section 4.1.

We also train our GCN baseline following our unsupervised learning. We set the batch size to 4 and vary the number of GCN layers in $\{1, 2, 3\}$ and the hidden layer size in $\{32, 64, 128, 256\}$. We also use the Adam optimizer [23] to train this unsupervised GCN up to 50 epochs.

A.3 Evaluation protocol We utilizes the logistic regression classifier [11] with using the 10-fold cross-validation scheme to evaluate our models. We compare our unsupervised GCN (denoted as uGCN) and U2GNN with the baselines: Deep Graph Kernel (DGK) [49] and Anonymous Walk Embedding (AWE) [20].

A.4 Experimental results Table 3 presents the experimental results in the unsupervised learning. We encourage a re-evaluation to examine the existing GNNs from the supervised learning to our new unsupervised learning to see negative results. For example, regarding the supervised learning, as shown in Table 2, our supervised U2GNN outperforms GCN on the bioinformatics datasets. However, regarding the unsupervised learning, as shown in Table 3, our uGCN works better than our unsupervised U2GNN on PROTEINS, MUTAG, and PTC. These three datasets are much sparse, and node neighbors have a similar effect on each other. Hence, without using the graph labels during training, U2GNN does not increase the similar effects on node neighbors, leading to be outperformed by our uGCN on these datasets.

In general, both our unsupervised U2GNN and uGCN obtain the state-of-the-art accuracies on the
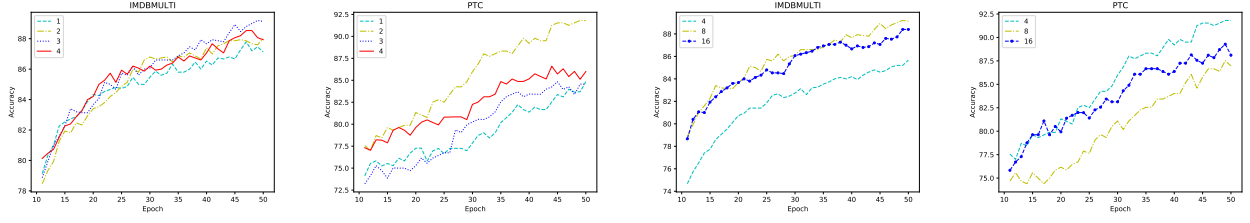
Figure 5: Effects of the number of timesteps (T) and the number of neighbors sampled for each node ($N = |\mathcal{N}_v|$) in the unsupervised learning.

benchmark datasets. The significant gains demonstrate a notable impact of our unsupervised learning. It aims to guide GNNs to identify the sub-graphs' structures for every node; hence, the models can memorize the structural differences among graphs to produce the plausible node and graph embeddings as visualized in Figure 6, leading to improve the unsupervised performance. We hope that future GNN works can consider the unsupervised learning beside the supervised one.

Hyper-parameter analysis. As shown in both Figures 4 and 5, we also see similar findings in both the supervised and unsupervised training settings.
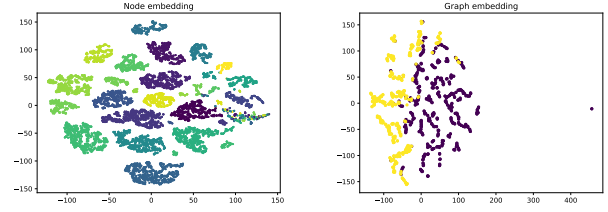

Figure 6: A visualization of the node and graph embeddings learned by our unsupervised U2GNN on the DD dataset.