Universal Self-Attention Network for Graph Classification

Dai Quoc Nguyen DAI.NGUYEN@MONASH.EDU
Monash University, Australia

Tu Dinh Nguyen NGUYENDINHTU@GMAIL.COM
Australia

Dinh Phung DINH.PHUNG@MONASH.EDU
Monash University, Australia

Abstract

We consider a limitation in using graph neural networks (GNNs) for graph classification: the lack of mechanism to exploit dependencies among nodes, often due to the lack in efficiency of aggregating nodes' neighbors. To this end, we present U2GNN – a novel embedding model leveraging the transformer self-attention network – to learn plausible node and graph embeddings. In particular, our U2GNN induces a powerful aggregation function, using a self-attention mechanism followed by a recurrent transition, to update vector representation of each node from its neighbors. As a consequence, U2GNN effectively infer the potential dependencies among nodes, leading to better modeling of graph structures. Experimental results show that the proposed U2GNN achieves state-of-the-art accuracies on benchmark datasets for the graph classification task. Our code is available at: https://github.com/daiquocnguyen/Graph-Transformer.

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 (Battaglia et al., 2018; Zhang et al., 2018b). 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 (Yanardag and Vishwanathan, 2015). Learning graph representations is one of the most important topics for the graph-structured data (Hamilton et al., 2017b; Zhou et al., 2018; Wu et al., 2019; Zhang et al., 2020a), where we aim to construct vector embeddings for nodes and graphs.

Early approaches focus on computing similarities among graphs to build a graph kernel for graph classification (Gärtner et al., 2003; Kashima et al., 2003; Borgwardt and Kriegel, 2005; Shervashidze et al., 2009; Vishwanathan et al., 2010; Shervashidze et al., 2011; Yanardag and Vishwanathan, 2015). Some graph kernel-based approaches split a given graph into "atomic substructures" (such as subtree structures, random walks, and shortest paths) to obtain a frequency vector to represent the entire graph. Some other approaches such as Deep Graph Kernel (DGK) (Yanardag and Vishwanathan, 2015), Graph2Vec (Narayanan

et al., 2017) and Anonymous Walk Embedding (AWE) (Ivanov and Burnaev, 2018) apply word embedding models such as Word2Vec (Mikolov et al., 2013) and Doc2Vec (Le and Mikolov, 2014) to learn embeddings for the atomic substructures or the entire graphs.

Recently, graph neural network (GNN)-based approaches become an essential strand to learn low-dimensional continuous embeddings of the entire graphs to predict graph labels (Scarselli et al., 2009; Hamilton et al., 2017b; Zhou et al., 2018; Wu et al., 2019; Zhang et al., 2020a). These GNN-based approaches often consist of two typical phases: *the aggregating phase* and *the readout phase* (Scarselli et al., 2009; Li et al., 2016; Gilmer et al., 2017; Xu et al., 2019; Xinyi and Chen, 2019; Maron et al., 2019b; Chen et al., 2019). The former phase aims to update the vector representation of each node by transforming and aggregating the vector representations of its neighbors (Kipf and Welling, 2017; Hamilton et al., 2017a; Veličković et al., 2018). Then the latter stage applies a pooling function (i.e., a ReadOut operation such as a simple sum pooling (Xu et al., 2019), or advanced poolings (Zhang et al., 2018a; Cangea et al., 2018; Ying et al., 2018)) to obtain an embedding for each entire graph to predict its label. We see that the GNN-based approaches have been obtaining state-of-the-art classification performances. Nonetheless, the dependency aspect among nodes has not been exploited effectively because of the lack of advanced computations in the aggregating phase.

The transformer self-attention network (Vaswani et al., 2017; Dehghani et al., 2019) has successfully utilized in NLP tasks such as question answering, machine translation, and language modeling. Inspired by this self-attention network, we present U2GNN – a new GNN model to learn the plausible node and graph embeddings. Our U2GNN induces an effective aggregation process – using a self-attention mechanism (Vaswani et al., 2017) followed by a recurrent transition – to capture the implicit dependencies among nodes effectively. Our main contributions in this paper are as follows:

- We propose U2GNN, using an advanced computation process in the aggregating phase based on the transformer self-attention network, to capture the graph structures effectively to produce the plausible node and graph embeddings.
- Experimental results show that U2GNN obtains state-of-the-art accuracies on benchmark datasets for the graph classification task in the supervised setting.
- We see that 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 propose new unsupervised learning to train a GNN model for the graph classification task. Our unsupervised results imply that our unsupervised learning can recognize and distinguish the substructures within each graph, leading to identify the structural differences among graphs effectively.

2. Related work

Early approaches aim to decompose each graph into "atomic substructures" (e.g., graphlets, subtree structures, random walks, or shortest paths) to measure the similarities among graphs (Gärtner et al., 2003). Therefore, we can view each atomic substructure as a word token 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) (Hofmann et al., 2008) to measure the classification performance. 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 such as Word2Vec (Mikolov et al., 2013) and Doc2Vec (Le and Mikolov, 2014), several works have used them for the graph classification task. Deep Graph Kernel (DGK) (Yanardag and Vishwanathan, 2015) applies Word2Vec to learn embeddings for the atomic substructures to create the kernel matrix. Graph2Vec (Narayanan et al., 2017) employs Doc2Vec to obtain the graph embeddings to train a SVM classifier to perform classification. Anonymous Walk Embedding (AWE) (Ivanov and Burnaev, 2018) maps random walks into "anonymous walks", views each anonymous walk as a word token, and utilizes Doc2Vec to achieve the graph embeddings to construct the kernel matrix.

Recent works have focused on using graph neural networks (GNNs) to perform the graph classification task (Scarselli et al., 2009; Li et al., 2016; Niepert et al., 2016; Gilmer et al., 2017; Zhang et al., 2018a; Ying et al., 2018; Verma and Zhang, 2018; Xu et al., 2019). In general, GNNs aim to update the vector representation of each node by propagating the vector representations of its neighbors recursively through using the recurrent Aggregation function until convergence (Scarselli et al., 2009). The Aggregation function can be a neural network such as gated recurrent units (GRU) (Li et al., 2016) and multi-layer perceptrons (MLPs) (Xu et al., 2019). GCN, GraphSAGE, and GAT can be seen as variants of this function. GNNs then apply the graph-level pooling layer (i.e., the ReadOut operation) to obtain the graph embeddings, which are fed to multiple fully-connected layers followed by a $\mathsf{softmax}$ layer to predict the graph labels. Other graph embedding models are summarized in (Zhou et al., 2018; Zhang et al., 2018b; Wu et al., 2019).

Regarding the aggregation of node representations, GCN (Kipf and Welling, 2017) 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:

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

where k is the layer index; $\mathbf{W}^{(k)}$ is a weight matrix of the k-th layer; $\mathbf{h}_{\mathsf{u}}^{(0)}$ 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 (Hamilton et al., 2017a) is an extension of GCN as:

$$\mathbf{h}_{\mathsf{v}}^{(k)} = \mathsf{g}\left(\mathbf{W}^{(k)} \left[\mathbf{h}_{\mathsf{v}}^{(k-1)}; \mathbf{h}_{\mathcal{N}_{\mathsf{v}}}^{(k)}\right]\right), \forall \mathsf{v} \in \mathcal{V} \tag{2}$$

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

$$\mathbf{h}_{\mathcal{N}_{\mathsf{v}}}^{(k)} = \mathsf{max}\left(\left\{\mathsf{g}\left(\mathbf{W}_{pool} \mathbf{h}_{\mathsf{u}}^{(k-1)} + \mathbf{b}\right), \forall \mathsf{u} \in \mathcal{N}_{\mathsf{v}}\right\}\right) \tag{3}$$

where $\mathcal{N}_{\mathsf{v}}$ is defined as a fixed-size, uniformly drawn from the set of all neighbor nodes of v, and uniformly sampled differently through each stacked layer.

Graph Attention Network (GAT) (Veličković et al., 2018) extends GCN in assigning importance weights to neighbors of a given node by applying the standard attention technique (Bahdanau et al., 2015). The vector representation $\mathbf{h}_{\mathsf{v}}^{(k)}$ of node v is aggregated from its neighbors as:

$$\mathbf{h}_{\mathsf{v}}^{(k)} = \mathsf{g}\left(\sum_{\mathsf{u}\in\mathcal{N}_{\mathsf{v}}\cup\{\mathsf{v}\}} \tau_{\mathsf{v},\mathsf{u}}^{(k)} \mathbf{W}^{(k)} \mathbf{h}_{\mathsf{u}}^{(k-1)}\right), \forall \mathsf{v} \in \mathcal{V} \tag{4}$$

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

$$\tau_{\mathsf{v},\mathsf{u}}^{(k)} = \mathsf{softmax}\left(\mathsf{LeakyReLU}\left(\boldsymbol{a}^{(k)} \cdot \left[\mathbf{W}^{(k)}\mathbf{h}_{\mathsf{v}}^{(k-1)}; \mathbf{W}^{(k)}\mathbf{h}_{\mathsf{u}}^{(k-1)}\right]\right)\right) \tag{5}$$

3. The proposed U2GNN

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

Graph classification. We represent each graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \{\mathbf{h}_{\mathsf{v}}\}_{\mathsf{v}\in\mathcal{V}})$, where $\mathcal{V}$ is a set of nodes, $\mathcal{E}$ is a set of edges, and $\mathbf{h}_{\mathsf{v}} \in \mathbb{R}^d$ represents the feature vector of node $\mathsf{v} \in \mathcal{V}$. Given a set of M graphs $\{\mathcal{G}_m\}_{m=1}^{M}$ and their corresponding class labels $\{\mathsf{y}_m\}_{m=1}^{M} \subseteq \mathcal{Y}$, 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.

Graph neural networks. Recent works about graph representation learning have focused on using graph neural networks (GNNs). In general, GNNs aim to update the vector representation of each node by recursively aggregating and transforming the vector representations of its neighbors (Kipf and Welling, 2017; Hamilton et al., 2017a; Veličković et al., 2018). After that, GNNs use a ReadOut pooling function to obtain the vector representations of the entire graphs (Gilmer et al., 2017; Zhang et al., 2018a; Ying et al., 2018; Verma and Zhang, 2018; Xu et al., 2019). Mathematically, given a graph $\mathcal{G}$, we can formalize GNNs as follows:

$$\mathbf{h}_{\mathsf{v}}^{(k)} = \textsc{Aggregation}\left(\left\{\mathbf{h}_{\mathsf{u}}^{(k-1)}\right\}_{\mathsf{u}\in\mathcal{N}_{\mathsf{v}}\cup\{\mathsf{v}\}}\right) \tag{6}$$

$$\mathbf{e}_{\mathcal{G}} = \textsc{ReadOut}\left(\{\mathbf{e}_{\mathsf{v}}\}_{\forall \mathsf{v}\in\mathcal{V}}\right) \tag{7}$$

where $\mathbf{h}_{\mathsf{v}}^{(k)}$ is the vector representation of node v at the k-th iteration/layer, $\mathcal{N}_{\mathsf{v}}$ is the set of neighbors of node v, and $\mathbf{h}_{\mathsf{v}}^{(0)} = \mathbf{h}_{\mathsf{v}}$.

Many methods have been proposed to construct the Aggregation function. Recently, Graph Isomorphism Network (GIN-0) (Xu et al., 2019) uses a more powerful Aggregation function based on a multi-layer perceptron (MLP) network of two fully-connected layers as:

$$\mathbf{h}_{\mathsf{v}}^{(k)} = \mathsf{MLP}^{(\mathsf{k})}\left(\sum_{\mathsf{u}\in\mathcal{N}_{\mathsf{v}}\cup\{\mathsf{v}\}} \mathbf{h}_{\mathsf{u}}^{(k-1)}\right), \forall \mathsf{v} \in \mathcal{V} \tag{8}$$

Following (Xu et al., 2019), we employ a concatenation over the vector representations of node v at the different layers to construct a vector representation $\mathbf{e}_{\mathsf{v}}$ for each node v as:

$$\mathbf{e}_{\mathsf{v}} = \left[\mathbf{h}_{\mathsf{v}}^{(1)}; \mathbf{h}_{\mathsf{v}}^{(2)}; ...; \mathbf{h}_{\mathsf{v}}^{(K)}\right], \forall \mathsf{v} \in \mathcal{V} \tag{9}$$

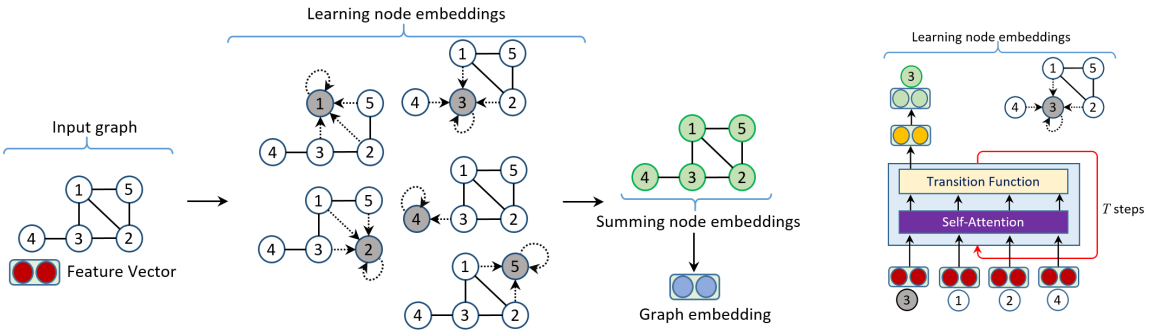

Figure 1: Illustration of our U2GNN.

where K is the index of the last layer. The graph-level ReadOut function can be a simple sum pooling (Xu et al., 2019) or a more advanced pooling such as sort pooling (Zhang et al., 2018a), hierarchical pooling (Cangea et al., 2018), and differentiable pooling (Ying et al., 2018). As the sum pooling can produce competitive results (Xu et al., 2019), we use the sum pooling to obtain the embedding $\mathbf{e}_{\mathcal{G}}$ of the entire graph $\mathcal{G}$ as:

$$\mathbf{e}_{\mathcal{G}} = \textsc{ReadOut}\left(\{\mathbf{e}_{\mathsf{v}}\}_{\forall \mathsf{v} \in \mathcal{V}}\right) = \sum_{\mathsf{v} \in \mathcal{V}} \mathbf{e}_{\mathsf{v}} = \sum_{\mathsf{v} \in \mathcal{V}} \left[\mathbf{h}_{\mathsf{v}}^{(1)}; \mathbf{h}_{\mathsf{v}}^{(2)}; ...; \mathbf{h}_{\mathsf{v}}^{(K)}\right] \tag{10}$$

The proposed U2GNN. We consider the lack of mechanism to exploit dependencies among nodes in recent graph neural networks (GNNs), often due to the lack in efficiency of aggregating nodes' neighbors (especially, in most of the existing Aggregation functions). Intuitively, we aim to construct our U2GNN as an extension of the GIN-0 architecture (Xu et al., 2019) in Equation 8. In particular, in U2GNN, we induce an advanced Aggregation function, using the universal self-attention network (Dehghani et al., 2019) consisting of a self-attention mechanism (Vaswani et al., 2017) followed by a recurrent transition (Trans) with adding residual connection (He et al., 2016) and layer normalization (LNorm) (Ba et al., 2016), as illustrated in Figure 1.

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 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 use 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:

$$\mathbf{h}_{t,\mathsf{u}}^{(k)} = \textsc{Graph-Transformer}\left(\mathbf{h}_{t-1,\mathsf{u}}^{(k)}\right) \tag{11}$$

$$\text{In particular, } \mathbf{h}_{t,\mathsf{u}}^{(k)} = \textsc{LNorm}\left(\mathbf{x}_{t,\mathsf{u}}^{(k)} + \textsc{Trans}\left(\mathbf{x}_{t,\mathsf{u}}^{(k)}\right)\right) \tag{12}$$

$$\text{with } \mathbf{x}_{t,\mathsf{u}}^{(k)} = \textsc{LNorm}\left(\mathbf{h}_{t-1,\mathsf{u}}^{(k)} + \text{ATT}\left(\mathbf{h}_{t-1,\mathsf{u}}^{(k)}\right)\right) \tag{13}$$

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

$$\textsc{Trans}\left(\mathbf{x}_{t,\mathsf{u}}^{(k)}\right) = \mathbf{W}_2^{(k)} \text{ReLU}\left(\mathbf{W}_1^{(k)} \mathbf{x}_{t,\mathsf{u}}^{(k)} + \mathbf{b}_1^{(k)}\right) + \mathbf{b}_2^{(k)} \tag{14}$$

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:

$$\text{ATT}\left(\mathbf{h}_{t-1,\mathsf{u}}^{(k)}\right) = \sum_{\mathsf{u}' \in \mathcal{N}_\mathsf{v} \cup \{\mathsf{v}\}} \alpha_{\mathsf{u},\mathsf{u}'}^{(k)} \left(\boldsymbol{V}^{(k)} \mathbf{h}_{t-1,\mathsf{u}'}^{(k)}\right) \tag{15}$$

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

$$\alpha_{\mathsf{u},\mathsf{u}'}^{(k)} = \mathsf{softmax}\left(\frac{\left(\boldsymbol{Q}^{(k)} \mathbf{h}_{t-1,\mathsf{u}}^{(k)}\right) \cdot \left(\boldsymbol{K}^{(k)} \mathbf{h}_{t-1,\mathsf{u}'}^{(k)}\right)}{\sqrt{d}}\right) \tag{16}$$

where $\boldsymbol{Q}^{(k)} \in \mathbb{R}^{d \times d}$ and $\boldsymbol{K}^{(k)} \in \mathbb{R}^{d \times d}$ are query-projection and key-projection matrices, respectively. [1]

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

$$\mathbf{h}_{0,\mathsf{v}}^{(k+1)} = \mathbf{h}_{T,\mathsf{v}}^{(k)}, \forall \mathsf{v} \in \mathcal{V} \tag{17}$$

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

Algorithm 1 The U2GNN learning process.

Input: $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$ with its label y
for $k = 0, 1, ..., K-1$ **do**
  for $\mathsf{v} \in \mathcal{V}$ **do**
    Sample $\mathcal{N}_\mathsf{v}$ for v
    for $t = 1, 2, ..., T$ **do**
      $\forall \mathsf{u} \in \mathcal{N}_\mathsf{v} \cup \{\mathsf{v}\}$
      $\mathbf{x}_{t,\mathsf{u}}^{(k)} \leftarrow \text{LNorm}\left(\mathbf{h}_{t-1,\mathsf{u}}^{(k)} + \text{ATT}\left(\mathbf{h}_{t-1,\mathsf{u}}^{(k)}\right)\right)$
      $\mathbf{h}_{t,\mathsf{u}}^{(k)} \leftarrow \text{LNorm}\left(\mathbf{x}_{t,\mathsf{u}}^{(k)} + \text{Trans}\left(\mathbf{x}_{t,\mathsf{u}}^{(k)}\right)\right)$
    $\mathbf{h}_{0,\mathsf{v}}^{(k+1)} \leftarrow \mathbf{h}_{T,\mathsf{v}}^{(k)} \in \mathbb{R}^d$
$\mathbf{e}_\mathsf{v} \leftarrow \left[\mathbf{h}_{0,\mathsf{v}}^{(1)}; \mathbf{h}_{0,\mathsf{v}}^{(2)}; ...; \mathbf{h}_{0,\mathsf{v}}^{(K)}\right], \forall \mathsf{v} \in \mathcal{V}$
$\mathbf{e}_\mathcal{G} \leftarrow \sum_{\mathsf{v} \in \mathcal{V}} \mathbf{e}_\mathsf{v}$
$\mathsf{y} \leftarrow \mathsf{softmax}\left(\mathbf{W}\mathbf{e}_\mathcal{G} + \mathbf{b}\right)$

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

$$\mathbf{e}_\mathsf{v} = \left[\mathbf{h}_{0,\mathsf{v}}^{(1)}; \mathbf{h}_{0,\mathsf{v}}^{(2)}; ...; \mathbf{h}_{0,\mathsf{v}}^{(K)}\right], \forall \mathsf{v} \in \mathcal{V} \tag{18}$$

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

1. If we do not share the weight matrices in both the self-attention and transition functions across all positions and timesteps, T becomes the number of self-attention layers within each U2GNN layer.

$\mathcal{G}$. We feed $\mathbf{e}_{\mathcal{G}}$ to a single fully-connected layer followed by a $\mathsf{softmax}$ layer to predict the graph labels as:

$$\hat{\mathbf{y}}_{\mathcal{G}} = \mathsf{softmax}\left(\mathbf{W}\mathbf{e}_{\mathcal{G}} + \mathbf{b}\right) \tag{19}$$

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.

Discussion. We compare our proposed U2GNN with related works as follows:

- If we set T to 1, $\alpha^{(k)}_{\mathsf{u},\mathsf{u}'}$ to 1, $\boldsymbol{V}^{(k)}$ to the identity matrix in Equation 15, and do not use both the residual connection and the layer normalization, we simplify our U2GNN aggregation function (from Equations 17 and 12) as:

$$\begin{aligned}\mathbf{h}^{(k)}_{1,\mathsf{v}} = \textsc{Trans}\left(\mathrm{ATT}\left(\mathbf{h}^{(k)}_{0,\mathsf{v}}\right)\right) &= \textsc{Trans}\left(\sum_{\mathsf{u}\in\mathcal{N}_{\mathsf{v}}\cup\{\mathsf{v}\}} \mathbf{h}^{(k)}_{0,\mathsf{u}}\right) \\ &= \textsc{Trans}\left(\sum_{\mathsf{u}\in\mathcal{N}_{\mathsf{v}}\cup\{\mathsf{v}\}} \mathbf{h}^{(k-1)}_{1,\mathsf{u}}\right)\end{aligned} \tag{20}$$

  where $\textsc{Trans}(.)$ is the network of two fully-connected layers (defined in Equation 14); thus, Equation 20 is now equivalent to Equation 8 of Graph Isomorphism Network (GIN-0) (Xu et al., 2019) – one of the recent state-of-the-art GNNs. This shows that our U2GNN uses a more advanced aggregation process, which is an extension of GIN-0. Experimental results presented in Section 5.3 show that U2GNN outperforms GIN-0 on benchmark datasets for the graph classification task.

- GAT (Veličković et al., 2018) (in Equation 5) 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 non-linearity function followed by the $\mathsf{softmax}$ 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) (Yun et al., 2019) identifies useful meta-paths (Wang et al., 2019) to transform the graph structures and applies GCN (Kipf and Welling, 2017) to learn the node embeddings for the node classification task on heterogeneous graphs. Self-Attention Graph Pooling (SAGPool) (Lee et al., 2019) uses GCN as an attention mechanism to weight the nodes, employs a node selection method (Gao and Ji, 2019) to retain a portion of the nodes, and applies the existing graph-level $\textsc{ReadOut}$ pooling layers (consisting of global pooling (Zhang et al., 2018a) and hierarchical pooling (Cangea et al., 2018)) to obtain the graph embeddings.

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

- A concurrent work, Hyper-SAGNN (Zhang et al., 2020c), utilizes the transformer self-attention network for hypergraphs that have diverse and different structures, hence required a different solution. Besides, the *later* and closely related work, Graph-BERT (Zhang et al., 2020b), is an extension of our U2GNN for the semi-supervised node classification task.
- We probably could construct an extended architecture using an advanced graph-level pooling (such as global pooling and hierarchical pooling). However, we refrained to do that as we have to design different architectures for different datasets, whilst our key purpose is to introduce a single, unified framework that can work well and produce competitive performances on the benchmark datasets; hence the sum pooling is a reasonable choice as it can produce competitive results as shown in (Xu et al., 2019).
- As established empirically, our results shown in Section 5.3 imply that the U2GNN self-attention-based aggregation function is a powerful computation process compared to other existing functions.

Algorithm 2 The U2GNN unsupervised learning.

Input: $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$
for $k = 0, 1, ..., K-1$ **do**
  for $\mathsf{v} \in \mathcal{V}$ **do**
    SAMPLE $\mathcal{N}_{\mathsf{v}}$ for v
    for $t = 1, 2, ..., T$ **do**
      $\forall \mathsf{u} \in \mathcal{N}_{\mathsf{v}} \cup \{\mathsf{v}\}$
      $\mathbf{x}_{t,\mathsf{u}}^{(k)} \leftarrow \text{LNORM}\left(\mathbf{h}_{t-1,\mathsf{u}}^{(k)} + \text{ATT}\left(\mathbf{h}_{t-1,\mathsf{u}}^{(k)}\right)\right)$
      $\mathbf{h}_{t,\mathsf{u}}^{(k)} \leftarrow \text{LNORM}\left(\mathbf{x}_{t,\mathsf{u}}^{(k)} + \text{TRANS}\left(\mathbf{x}_{t,\mathsf{u}}^{(k)}\right)\right)$
    $\mathbf{h}_{0,\mathsf{v}}^{(k+1)} \leftarrow \mathbf{h}_{T,\mathsf{v}}^{(k)} \in \mathbb{R}^d$
$\mathbf{e}_{\mathsf{v}} \leftarrow \left[\mathbf{h}_{0,\mathsf{v}}^{(1)}; \mathbf{h}_{0,\mathsf{v}}^{(2)}; ...; \mathbf{h}_{0,\mathsf{v}}^{(K)}\right], \forall \mathsf{v} \in \mathcal{V}$
$\mathbf{o}_{\mathsf{v}} \leftarrow \mathbf{e}_{\mathsf{v}}$ (with respect to Equation 21)
$\mathbf{e}_{\mathcal{G}} \leftarrow \sum_{\mathsf{v} \in \mathcal{V}} \mathbf{o}_{\mathsf{v}}$

Unsupervised graph neural networks. Most of the recent approaches have focused on the supervised learning where they use the graph labels during the training process (Xinyi and Chen, 2019; Xu et al., 2019; Chen et al., 2019; Maron et al., 2019b; Seo et al., 2019). In a situation where *no graph labels are available during training*, some work (such as DGK, Graph2Vec, and AWE) 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)* (Yanardag and Vishwanathan, 2015; Narayanan et al., 2017; Ivanov and Burnaev, 2018). But they produce lower classification accuracies compared to the supervised approaches. To this end, we propose new unsupervised learning to train a GNN model for the graph classification task.

We can see $\mathbf{e}_{\mathsf{v}}$ in Equation 9 as a vector representation encoded for the substructure around node v. Our unsupervised learning aims to guide a GNN model to recognize and

distinguish the substructures within each graph, leading to infer the structural differences among graphs effectively. To achieve this goal, we consider a final embedding $\mathbf{o}_{\mathsf{v}}$ for each node v, and make the similarity between $\mathbf{e}_{\mathsf{v}}$ and $\mathbf{o}_{\mathsf{v}}$ higher than that between $\mathbf{e}_{\mathsf{v}}$ and the final embeddings of the other nodes, by minimizing the sampled $\mathsf{softmax}$ loss function (Jean et al., 2015) applied to node v as:

$$\mathcal{L}_{\mathsf{U2GNN}}(\mathsf{v}) = -\log \frac{\exp(\mathbf{o}_{\mathsf{v}} \cdot \mathbf{e}_{\mathsf{v}})}{\sum_{\mathsf{v}' \in \mathcal{V}'} \exp(\mathbf{o}_{\mathsf{v}'} \cdot \mathbf{e}_{\mathsf{v}})} \tag{21}$$

where $\mathcal{V}'$ is a subset sampled from $\{\cup \mathcal{V}_m\}_{m=1}^{M}$. Node embeddings $\mathbf{o}_{\mathsf{v}}$ are learned implicitly as model parameters. After that, we sum all the final embeddings $\mathbf{o}_{\mathsf{v}}$ of nodes v in $\mathcal{G}$ to obtain the graph embedding $\mathbf{e}_{\mathcal{G}}$. We then use the logistic regression classifier (Fan et al., 2008) with setting the termination criterion to 0.001 to evaluate our model.

4. Experimental datasets

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 (Niepert et al., 2016; Zhang et al., 2018a) to use node degrees as features. Table 1 reports the statistics of these datasets.

Table 1: Statistics of the experimental benchmark datasets. #Avg.NG denotes the average number of nodes per graph. #Avg.NN denotes the average number of neighbors per node. d is the dimension of feature vectors.

Dataset	**#Graphs**	**#Classes**	**#Avg.NG**	**#Avg.NN**	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 (Dobson and 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. 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. PROTEINS comprises 1,113 graphs obtained from (Borgwardt et al., 2005) to present secondary structure elements (SSEs). 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.

5. Supervised graph neural networks

5.1. Training protocol

We vary the number K of U2GNN layers in $\{1, 2, 3\}$, the number of steps T in $\{1, 2, 3, 4\}$, and the number of neighbors ($|\mathcal{N}_v| = N$) sampled for each node in $\{4, 8, 16\}$. We set the hidden size s of the feed-forward network to 1024 (in Equation 14) and the batch size to 4. We apply the Adam optimizer (Kingma and Ba, 2015) to train our U2GNN and select the Adam initial learning rate $lr \in \left\{5e^{-5}, 1e^{-4}, 5e^{-4}, 1e^{-3}\right\}$. We run up to 50 epochs to evaluate our U2GNN.

5.2. Evaluation protocol

We follow (Xu et al., 2019; Xinyi and Chen, 2019; Maron et al., 2019a; Seo et al., 2019; Chen et al., 2019) 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: PATCHY-SAN (PSCN) (Niepert et al., 2016), Graph Convolutional Network (GCN) (Kipf and Welling, 2017), GraphSAGE (Hamilton et al., 2017a), Graph Attention Network (GAT) (Veličković et al., 2018), Deep Graph CNN (DGCNN) (Zhang et al., 2018a), Graph Capsule Convolution Neural Network (GCAPS) (Verma and Zhang, 2018), Capsule Graph Neural Network (CapsGNN) (Xinyi and Chen, 2019), Self-Attention Graph Pooling (SAGPool) (Lee et al., 2019), Graph Isomorphism Network (GIN-0) (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 baselines taken from the original papers or published in (Ivanov and Burnaev, 2018; Verma and Zhang, 2018; Xinyi and Chen, 2019; Fan et al., 2019; Chen et al., 2019; Seo et al., 2019).

5.3. Experimental results

Table 2 presents the experimental results of U2GNN and other strong baseline models. 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. On the bioinformatics datasets, our U2GNN obtains the highest accuracies on DD, PROTEINS, and PTC. 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 some baselines (e.g., GFN, GIN-0, and PPGN) on MUTAG as this dataset only consists of 188 graphs, which explains the high variance in the results.

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

Model	COLLAB	IMDB-B	IMDB-M	DD	PROTEINS	MUTAG	PTC
PSCN (2016)	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 (2017)	**81.72 ± 1.64**	73.30 ± 5.29	51.20 ± 5.13	79.12 ± 3.07	75.65 ± 3.24	87.20 ± 5.11	–
GraphSAGE (2017a)	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 (2018)	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 (2018a)	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
GCAPS (2018)	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 (2019b)	77.92 ± 1.70	71.27 ± 4.50	48.55 ± 3.90	–	75.19 ± 4.30	84.61 ± 10.0	59.47 ± 7.30
CapsGNN (2019)	79.62 ± 0.91	73.10 ± 4.83	50.27 ± 2.65	75.38 ± 4.17	76.28 ± 3.63	86.67 ± 6.88	–
SAGPool (2019)	–	–	–	76.45 ± 0.97	71.86 ± 0.97	–	–
DSGC (2019)	79.20 ± 1.60	73.20 ± 4.90	48.50 ± 4.80	77.40 ± 6.40	74.20 ± 3.80	86.70 ± 7.60	–
GFN (2019)	81.50 ± 2.42	73.00 ± 4.35	51.80 ± 5.16	78.78 ± 3.49	76.46 ± 4.06	90.84 ± 7.22	–
PPGN (2019a)	81.38 ± 1.42	73.00 ± 5.77	50.46 ± 3.59	–	77.20 ± 4.73	90.55 ± 8.70	66.17 ± 6.54
GIN-0 (2019)	80.20 ± 1.90	75.10 ± 5.10	52.30 ± 2.80	–	76.20 ± 2.80	89.40 ± 5.60	64.60 ± 7.00
U2GNN	77.84 ± 1.48	**79.40 ± 4.35**	**56.20 ± 3.35**	**81.24 ± 1.84**	**78.53 ± 4.07**	89.97 ± 3.65	**79.36 ± 4.06**

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 a more advanced computation process to infer the potential dependencies among nodes, leading to better modeling of the graph structures.

5.4. Visualization

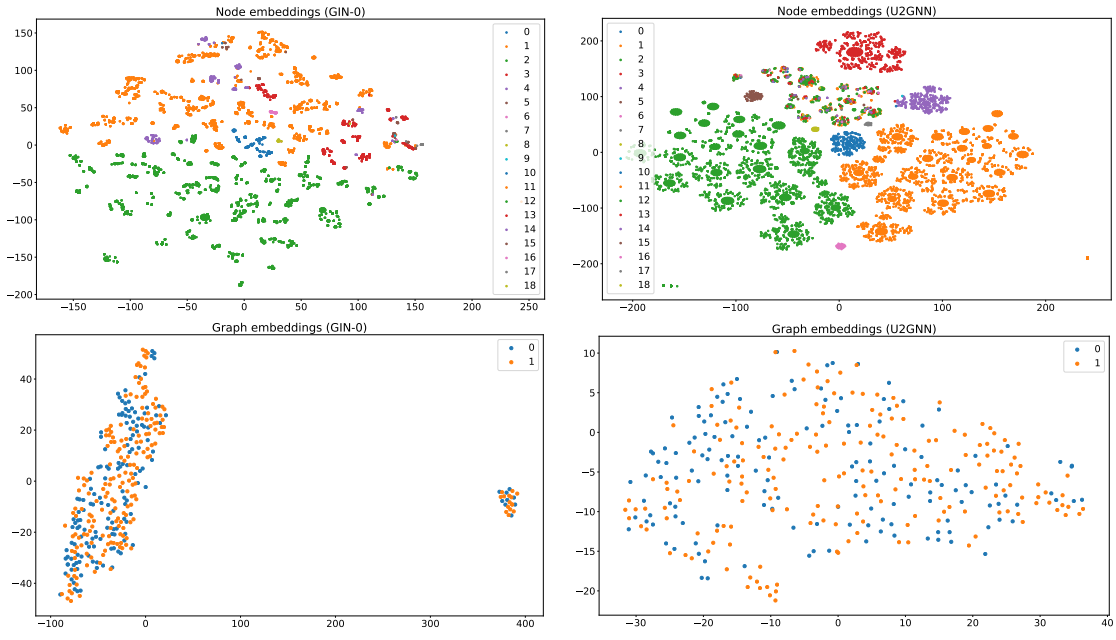

Figure 2: A visualization of the node and graph embeddings learned by GIN-0 and our U2GNN on the PTC dataset.

To demonstrate the effectiveness of capturing the node and graph properties, we use t-SNE (Maaten and Hinton, 2008) to visualize the node and graph embeddings learned by GIN-0 and our U2GNN on the PTC dataset where the node labels are available. Compared to GIN-0, Figure 2 shows that our U2GNN effectively capture the local structures wherein the nodes are well-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. Unsupervised graph neural networks

Note that we do not make a direct comparison between the unsupervised and supervised approaches because of the difference in the training data.

6.1. Training protocol

We follow some unsupervised approaches (Yanardag and Vishwanathan, 2015; Narayanan et al., 2017; Ivanov and Burnaev, 2018) to train our unsupervised U2GNN on all nodes from the entire dataset (i.e., using all nodes from the test set during training). The hyper-parameters are varied as same as in Section 5.1.

GCN is one of the state-of-the-art GNNs for semi-supervised node classification. Besides, as shown in (Xinyi and Chen, 2019; Fan et al., 2019; Chen et al., 2019; Seo et al., 2019), GCN outperforms GAT and GraphSAGE in the downstream task of graph classification and provides competitive accuracies compared to other existing models as shown in Section 5.3 for the supervised learning. Hence, for the unsupervised learning, we construct and train a GCN baseline – following our unsupervised learning – to learn the node and graph embeddings. 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 (Kingma and Ba, 2015) to train this GCN model up to 50 epochs to compute the classification accuracy.

We compare our unsupervised U2GNN with the baselines: Deep Graph Kernel (DGK) (Yanardag and Vishwanathan, 2015), Anonymous Walk Embedding (AWE) (Ivanov and Burnaev, 2018), and our unsupervised GCN model.

6.2. Experimental results

Table 3: Graph classification results (% accuracy) in the unsupervised learning. The best scores are in bold. GCN (2017) denotes our unsupervised GCN baseline.

Model	COLLAB	IMDB-B	IMDB-M	DD	PROTEINS	MUTAG	PTC
DGK (2015)	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 (2018)	73.93 ± 1.94	74.45 ± 5.83	51.54 ± 3.61	71.51 ± 4.02	–	87.87 ± 9.76	–
GCN (2017)	93.28 ± 0.99	94.50 ± 2.79	81.66 ± 3.16	94.31 ± 1.71	**89.09 ± 3.25**	**95.36 ± 2.64**	**92.67 ± 4.60**
U2GNN	**95.62 ± 0.92**	**96.41 ± 1.94**	**89.20 ± 2.52**	**95.67 ± 1.89**	80.01 ± 3.21	88.47 ± 7.13	91.81 ± 6.61

Table 3 presents the experimental results in the unsupervised learning on the benchmark datasets. Again, we note that we follow some unsupervised approaches such as DGK (Yanardag and Vishwanathan, 2015) and AWE (Ivanov and Burnaev, 2018) to use all nodes

from the entire dataset to train our unsupervised U2GNN and GCN models. Our unsupervised models obtain the state-of-the-art accuracies on the benchmark datasets. The significant gains demonstrate a notable impact of our unsupervised learning. It aims to guide the GNN models to identify the structures of the sub-graphs 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 3.

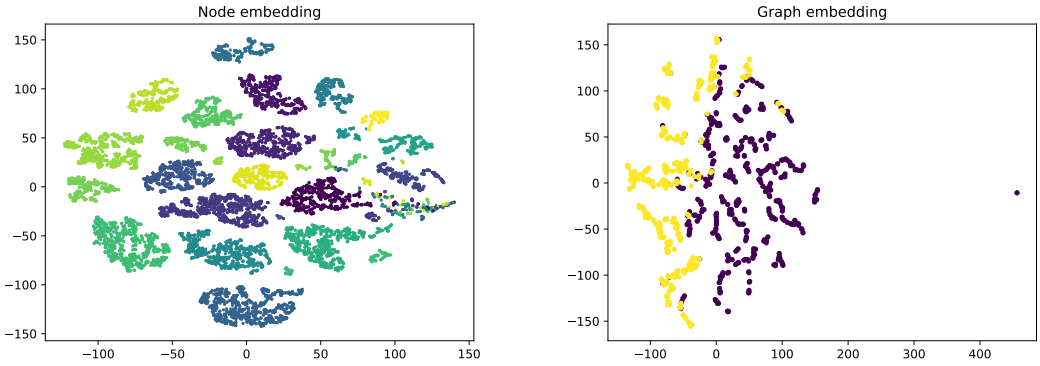

Figure 3: A visualization of the node and graph embeddings learned by our unsupervised U2GNN on the DD dataset.

6.3. 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. Note that similar findings also occur in the supervised training setting.

7. Conclusion

In this paper, we introduce U2GNN – a novel graph neural network for the graph classification task. U2GNN induces an advanced aggregation process based on the transformer self-attention network to effectively capture the graph structures. We evaluate our U2GNN using the same data splits and the same 10-fold cross-validation scheme on the well-known benchmark datasets. Experimental results show that the proposed U2GNN produces state-of-the-art accuracies on these datasets.

Furthermore, we hope that future GNN works should consider the unsupervised learning beside the supervised one.

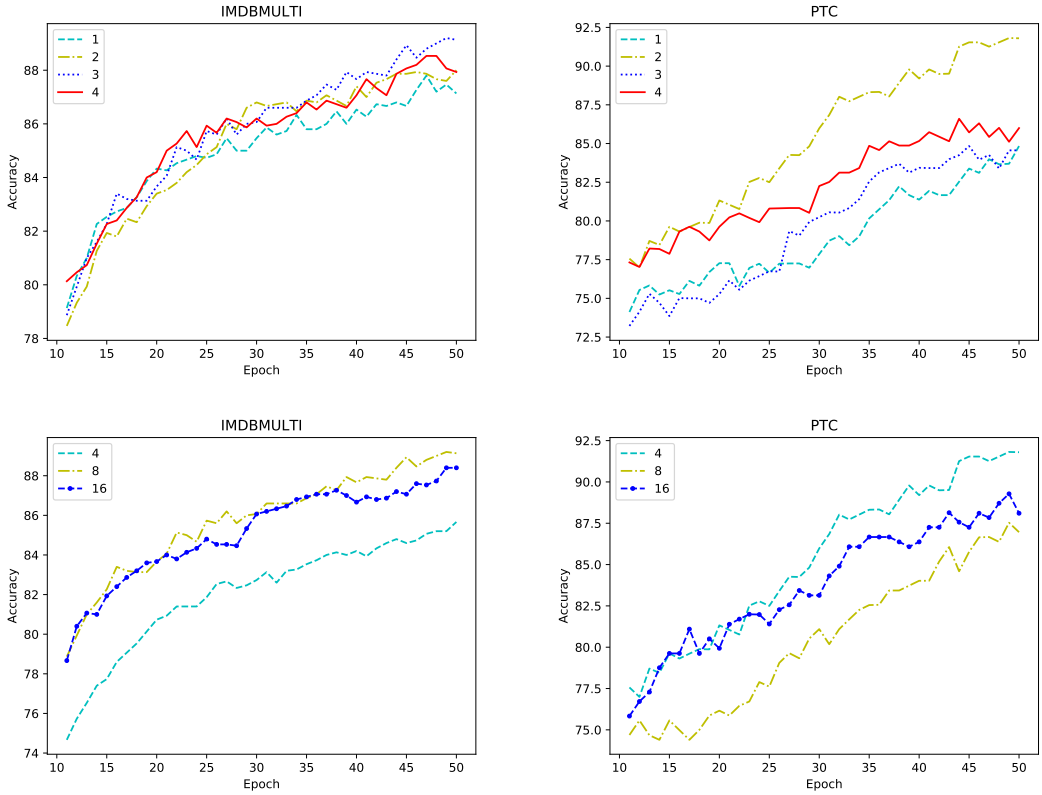

Figure 4: Effects of the number of timesteps (T) and the number of neighbors sampled for each node ($N = |\mathcal{N}_v|$) in the unsupervised learning.

References

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

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *International Conference on Learning Representations (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:1806.01261*, 2018.

Karsten M. Borgwardt and Hans-Peter Kriegel. Shortest-Path Kernels on Graphs. In *ICDM*, pages 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.

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

Ting Chen, Song Bian, and Yizhou Sun. Are powerful graph neural nets necessary? a dissection on graph classification. *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 (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.

Xiaolong Fan, Maoguo Gong, Yu Xie, Fenlong Jiang, and Hao Li. Structured self-attention architecture for graph-level representation learning. *Pattern Recognition*, 100, 2019. ISSN 0031-3203.

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

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.

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.

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

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

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.

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

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

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.

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

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

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

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

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

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

Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated Graph Sequence Neural Networks. *International Conference on Learning Representations (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. In *Advances in Neural Information Processing Systems*, pages 2153–2164, 2019a.

Haggai Maron, Heli Ben-Hamu, Nadav Shamir, and Yaron Lipman. Invariant and equivariant graph networks. *International Conference on Learning Representations (ICLR)*, 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*, pages 3111–3119, 2013.

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

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

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

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:1905.13422*, 2019.

Nino Shervashidze, SVN Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten Borgwardt. Efficient Graphlet Kernels for Large Graph Comparison. In *AISTATS*, pages 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*, pages 5998–6008, 2017.

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

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

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.

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.

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

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

Pinar Yanardag and SVN Vishwanathan. Deep graph kernels. In *The ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 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*, pages 4805–4815, 2018.

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.

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

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

Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An End-to-End Deep Learning Architecture for Graph Classification. In *The AAAI Conference on Artificial Intelligence*, 2018a.

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)*, 2020c.

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

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.