Graph Cross Networks with Vertex Infomax Pooling

Maosen Li
Shanghai Jiao Tong University
maosen_li@sjtu.edu.cn

Siheng Chen
Mitsubishi Electric Laboratories
schen@merl.com

Ya Zhang
Shanghai Jiao Tong University
ya_zhang@sjtu.edu.cn

Ivor W. Tsang
University of Technology, Sydney
Ivor.Tsang@uts.edu.au

Abstract

We propose a novel *graph cross network* (GXN) to achieve comprehensive feature learning from multiple scales of a graph. Based on trainable hierarchical representations of a graph, GXN enables the interchange of intermediate features across scales to promote information flow. Two key ingredients of GXN include a novel *vertex infomax pooling* (VIPool), which creates multiscale graphs in a trainable manner, and a novel feature-crossing layer, enabling feature interchange across scales. The proposed VIPool selects the most informative subset of vertices based on the neural estimation of mutual information between vertex features and neighborhood features. The intuition behind is that a vertex is informative when it can maximally reflect its neighboring information. The proposed feature-crossing layer fuses intermediate features between two scales for mutual enhancement by improving information flow and enriching multiscale features at hidden layers. The cross shape of feature-crossing layer distinguishes GXN from many other multiscale architectures. Experimental results show that the proposed GXN improves the classification accuracy by 1.96% and 1.15% on average for graph classification and vertex classification, respectively. Based on the same network, the proposed VIPool consistently outperforms other graph-pooling methods.

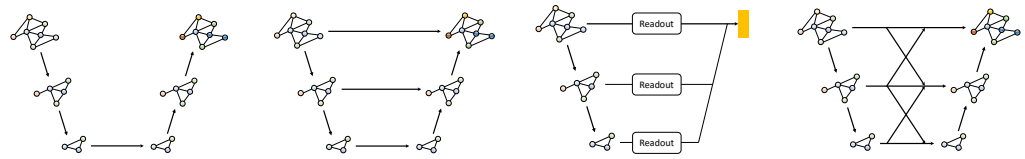
1 Introduction

Recently, there are explosive interests in studying graph neural networks (GNNs) [30, 23, 47, 29, 10, 17, 53, 50, 31, 33], which expand deep learning techniques to ubiquitous non-Euclidean graph data, such as social networks [49], bioinformatic networks [15] and human activities [33]. Achieving good performances on graph-related tasks, such as vertex classification [30, 23, 47] and graph classification [17, 53, 50], GNNs learn patterns from both graph structures and vertex information with feature extraction in spectral domain [5, 11, 30] or vertex domain [23, 37, 47, 32, 52, 10, 3]. Nevertheless, most GNN-based methods learn features of graphs with fixed scales, which might underestimate either local or global information. To address this issue, multiscale feature learning on graphs enables capturing more comprehensive graph features for downstream tasks [6, 20, 35].

Multiscale feature learning on graphs is a natural generalization from multiresolution analysis of images, whose related techniques, such as wavelets and pyramid representations, have been well studied in both theory and practice [24, 45, 41, 1, 54]. However, this generalization is technically nontrivial. While hierarchical representations and pixel-to-pixel associations across scales are straightforward for images with regular lattices, the highly irregular structures of graphs cause challenges in producing graphs at various scales [8] and aggregating features across scales.

To generate multiscale graphs, graph pooling methods are essential to compress large graphs into smaller ones. Conventional graph pooling methods [8, 42] leverage graph sampling theory and

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.



(a) Encoder-decoder [6]. (b) Graph U-net [20]. (c) Readout [31]. (d) GXN (ours).

Figure 1: Architectures of multiscale graph neural networks. Our architecture adopts intermediate fusion.

designed rules. Recently, some data-driven pooling methods are proposed, which automatically merge a fine-scale vertex subset to a coarsened vertex [13, 11, 43, 40, 34, 50]. The coarsened graph, however, might not have direct vertex-to-vertex association with the original scale. Some other graph pooling methods adaptively select vertices based on their importance over the entire graph [20, 31]; however, they fail to consider local information.

To aggregate features across multiple scales, existing attempts build encoder-decoder architecture [6, 34, 12] to learn graph features from the latent spaces, which might underestimate fine-scale information. Some other works gather the multiscale features in parallel and merge them as the final representation [35, 20, 19, 31], which might limit information flow across scales.

In this work, we design a new graph neural network to achieve multiscale feature learning on graphs, and our technical contributions are two-folds: a novel graph pooling operation to preserve informative vertices and a novel model architecture to exploit rich multiscale information.

A novel graph pooling operation: Vertex infomax pooling (VIPool). We propose a novel graph pooling operation by selecting and preserving those vertices that can maximally express their corresponding neighborhoods. The criterion of vertex-selection is based on the neural estimation of mutual information [2, 26, 48] between vertex and neighborhood features, thus we call the proposed pooling mechanism *vertex infomax pooling* (VIPool). Based on VIPool, we can implement graph pooling and unpooling to coarsen and refine multiple scales of a graph. Compared to the vertex-grouping-based methods [13, 11, 43, 40, 34, 50], the proposed VIPool provides the direct vertex-vertex association across scales and makes the coarsened graph structure and information fusion easier to achieve. Compared to other vertex-selection-based methods [20, 31], VIPool considers both local and global information on graphs by learning both vertex representation and graph structures.

A novel model architecture: Graph cross network (GXN). We propose a new model with a novel architecture called *graph cross network* (GXN) to achieve feature learning on multiscale graphs. Employing the trainable VIPool, our model creates multiscale graphs in data-driven manners. To learn features from all parallel scales, our model is built with a pyramid structure. To further promote information flow, we propose novel intermediate *feature-crossing layers* to interchange features across scales in each network layer. The intuition of feature-crossing is that the it improves information flow and exploits richer multiscale information in multiple network layers rather than only combine them in the last layer. Similar crossing structures have been explored for analyzing images [46, 45], but we cannot directly use those structures for irregular graphs. The proposed feature-crossing layer handles irregular graphs by providing the direct vertex-vertex associations across multiple graph scales and network layers; see typical multiscale architectures in Figure 1, where GXN is well distinguished because intermediate feature interchanging across scales forms a crossing shape.

Remark: In each individual scale, graph U-net [20] simply uses skip connections while GXN uses multiple graph propagation layers to extract features. The proposed feature-crossing layer is used to fuse intermediate features and cannot be directly applied to graph U-net.

To test our methods, we conduct extensive experiments on several standard datasets for both graph classification and vertex classification. Compared to state-of-the-art methods for these two tasks, GXN improves the average classification accuracies by 1.96% and 1.15%, respectively. Meanwhile, based on the same model architecture, our VIPool consistently outperforms previous graph pooling methods; and more intermediate connection leads to a better performance. [1]

2 Related Works

Multiscale graph neural networks with graph pooling. To comprehensively learn the multiscale graph representations, various multiscale network structures have been explored. Hierarchical encoder-

[1] The code could be downloaded at `https://github.com/limaosen0/GXN`

decoder structures [6, 34, 12] learn graph features just from much coarse scales. LancozsNet [35] designs various graph filters on the multiscale graphs. Graph U-net [20] and readout functions [19, 31] design pyramid structures with skip-connections and combines features from all scales in the last layer. Compared to previous works, the proposed GXN has two main differences. 1) Besides the common late fusion of features, GXN uses intermediate fusion across scales, where the features at various scales in each network layer are fused to embed richer multiscale information. 2) GXN extracts hierarchical multiscale features through a deep network, previous Graph U-net [20] extracts features only once in each scale and then uses skip-connections to fuse feature across scales.

To compress a graph into multiple coarser scales, various methods of graph pooling are proposed. Early graph pooling methods are usually designed based on graph sampling theory [8] or graph coarsening [42]. With the study of deep learning, some works down-scale graphs in data-driven manner. The graph-coarsening-based pooling methods [13, 11, 43, 40, 34, 50, 51] cluster vertices and merge each cluster to a coarsened vertex; however, there is not vertex-to-vertex association to preserve the original vertex information. The vertex-selection-based pooling methods [20, 31] preserve selected vertices based on their importance, but tend to loss the original graph structures. Compared to previous works, the proposed VIPool in GXN is trained given an explicit optimization for vertex selection, and the pooled graph effectively abstracts the original graph structure.

Mutual information estimation and maximization. Given two variables, to estimate their mutual information whose extact value is hard to compute, some models are constructed based on the parameterization of neural networks. [2] leverages trainable networks to depict a lower bound of mutual information, which could be optimized toward a precise mutual information estimation. [26] maximizes the pixel-image mutual information to promote to capture the most informative image patterns via self-supervision. [48] maximizes the mutual information between a graph and each single vertex, where the representative vertex features are obtained. Similarly, [44] applies the mutual information maximization on graph classification. Compared to these mutual-information-based studies, the proposed VIPool, which also leverages mutual information maximization on graphs, aims to obtain an optimization for vertex selection by finding the vertices that maximally represent their local neighborhood. We also note that, in VIPool, the data distribution is defined on a single graph, while previous works [48, 44] assume to train on the distribution of multiple graphs.

3 Vertex Infomax Pooling

Before introducing the overall model, we first propose a new graph pooling method to create multiple scales of a graph. In this graph pooling, we select and preserve a ratio of vertices and connect them based on the original graph structure. Since downscaling graphs would lose information, it is critical to preserve as much information as possible in the pooled graph, which could maximally represent the original graphs. To this end, we propose a novel *vertex infomax pooling* (VIPool), preserving the vertices that carry high mutual information with their surrounding neighborhoods by mutual information estimation and maximization. The preserved vertices well represent local subgraphs, and they also abstract the overall graph structure based on a vertex selection criterion.

Mathematically, let $G(\mathcal{V}, \mathbf{A})$ be a graph with a set of vertices $\mathcal{V} = \{v_1, \dots, v_N\}$ whose features are $\mathbf{X} = [\mathbf{x}_1 \cdots \mathbf{x}_N]^\top \in \mathbb{R}^{N\times d}$, and an adjacency matrix $\mathbf{A} \in \{0,1\}^{N\times N}$. We aim to select a subset $\Omega \subset \mathcal{V}$ that contains $|\Omega| = K$ vertices. Considering a criterion function $C(\cdot)$ to quantify the information of a vertex subset, we find the most informative subset through solving the problem,

$$\max_{\Omega\subset\mathcal{V}} \; C(\Omega), \quad \text{subject to} \;\; |\Omega| \;=\; K. \tag{1}$$

We design $C(\Omega)$ based on the mutual information between vertices and their corresponding neighborhoods, reflecting the vertices' abilities to express neighborhoods. In the following, we first introduce the computation of vertex-neighborhood mutual information, leading to the definition of $C(\cdot)$; we next select a vertex set by solving (1); we finally pool a fine graph based on the selected vertices.

Mutual information neural estimation. In a graph $G(\mathcal{V}, \mathbf{A})$, for any selected vertex v in $\Omega \subset \mathcal{V}$, we define v's neighborhood as $\mathcal{N}_v$, which is the subgraph containing the vertices in $\mathcal{V}$ whose geodesic distances to v are no greater than a threshold R according to the original $G(\mathcal{V}, \mathbf{A})$, i.e. $\mathcal{N}_v = G(\{u\}_{d(u,v)\le R}, \mathbf{A}_{\{u\},\{u\}})$. Let a random variable $\mathbf{v}$ be the feature of a randomly picked vertex in Ω, the distribution of $\mathbf{v}$ is $P_{\mathbf{v}} = P(\mathbf{v} = \mathbf{x}_v)$, where $\mathbf{x}_v$ is the outcome feature value when we pick vertex v. Similarly, let a random variable $\mathbf{n}$ be the neighborhood feature associated with a randomly

picked vertex in Ω, the distribution of $\mathbf{n}$ is $P_{\mathbf{n}} = P(\mathbf{n} = \mathbf{y}_{\mathcal{N}_u})$, where $\mathbf{y}_{\mathcal{N}_u}$ is the outcome feature value when we pick vertex u's neighborhood. The mutual information between selected vertices and neighborhoods is the KL-divergence between the joint distribution $P_{\mathbf{v},\mathbf{n}} = P(\mathbf{v} = \mathbf{x}_v, \mathbf{n} = \mathbf{y}_{\mathcal{N}_v})$ and the product of marginal distributions $P_{\mathbf{v}} \otimes P_{\mathbf{n}}$:

$$\begin{aligned} I^{(\Omega)}(\mathbf{v},\mathbf{n}) &= D_{\mathrm{KL}}\left(P_{\mathbf{v},\mathbf{n}} || P_{\mathbf{v}} \otimes P_{\mathbf{n}}\right) \\ &\overset{(a)}{\geq} \sup_{T\in\mathcal{T}} \left\{ \mathbb{E}_{\mathbf{x}_v,\mathbf{y}_{\mathcal{N}_v}\sim P_{\mathbf{v},\mathbf{n}}}\left[T(\mathbf{x}_v,\mathbf{y}_{\mathcal{N}_v})\right] - \mathbb{E}_{\mathbf{x}_v\sim P_{\mathbf{v}},\mathbf{y}_{\mathcal{N}_u}\sim P_{\mathbf{n}}}\left[e^{T(\mathbf{x}_v,\mathbf{y}_{\mathcal{N}_u})-1}\right]\right\}, \end{aligned}$$

where (a) follows from f-divergence representation based on KL divergence [2]; $T \in \mathcal{T}$ is an arbitrary function that maps features of a pair of vertex and neighborhood to a real value, here reflecting the dependency of two features. To achieve more flexibility and convenience in optimization, f-divergence representation based on a non-KL divergence can be adopted [38], which still measures the vertex-neighborhood dependency. Here we consider a GAN-like divergence.

$$I^{(\Omega)}_{\mathrm{GAN}}(\mathbf{v},\mathbf{n}) \geq \sup_{T\in\mathcal{T}} \left\{ \mathbb{E}_{P_{\mathbf{v},\mathbf{n}}}\left[\log\sigma\left(T(\mathbf{x}_v,\mathbf{y}_{\mathcal{N}_v})\right)\right] + \mathbb{E}_{P_{\mathbf{v}},P_{\mathbf{n}}}\left[\log\left(1-\sigma\left(T\left(\mathbf{x}_v,\mathbf{y}_{\mathcal{N}_u}\right)\right)\right)\right]\right\},$$

where $\sigma(\cdot)$ is the sigmoid function. In practice, we cannot go over the entire functional space $\mathcal{T}$ to evaluate the exact value of $I^{(\Omega)}_{\mathrm{GAN}}$. Instead, we parameterize $T(\cdot,\cdot)$ by a neural network $T_w(\cdot,\cdot)$, where the subscript w denotes the trainable parameters. Through optimizing over w, we obtain a neural estimation of the GAN-based mutual information, denoted as $\widehat{I}^{(\Omega)}_{\mathrm{GAN}}$. We can define our vertex selection criterion function to be this neural estimation; that is,

$$C(\Omega) \;=\; \widehat{I}^{(\Omega)}_{\mathrm{GAN}} \;=\; \max_{w} \frac{1}{|\Omega|}\sum_{v\in\Omega}\log\sigma\left(T_w(\mathbf{x}_v,\mathbf{y}_{\mathcal{N}_v})\right) + \frac{1}{|\Omega|^2}\sum_{(v,u)\in\Omega}\log\left(1-\sigma\left(T_w(\mathbf{x}_v,\mathbf{y}_{\mathcal{N}_u})\right)\right).$$

In $C(\Omega)$, the first term reflects the affinities between vertices and their own neighborhoods; and the second term reflects the differences between vertices and arbitrary neighborhoods. Notably, a higher C score indicates that vertices maximally reflect their own neighborhoods and meanwhile minimally reflect arbitrary neighborhoods. To specify T_w, we consider $T_w\left(\mathbf{x}_v,\mathbf{y}_{\mathcal{N}_u}\right) = \mathcal{S}_w\left(\mathcal{E}_w(\mathbf{x}_v),\mathcal{P}_w(\mathbf{y}_{\mathcal{N}_u})\right)$, where the subscript w indicates the associated functions are trainable[2], $\mathcal{E}_w(\cdot)$ and $\mathcal{P}_w(\cdot)$ are embedding functions of vertices and neighborhoods, respectively, and $\mathcal{S}_w(\cdot,\cdot)$ is an affinity function to quantify the affinity between vertices and neighborhoods; see an illustration in Figure 2. We implement $\mathcal{E}_w(\cdot)$ and $\mathcal{S}_w(\cdot,\cdot)$ by multi-layer perceptrons (MLPs), and implement $\mathcal{P}_w(\cdot)$ by aggregating vertex features and neighborhood connectivities in $\mathbf{y}_{\mathcal{N}_u}$; that is

$$\mathcal{P}_w(\mathbf{y}_{\mathcal{N}_u}) \;=\; \frac{1}{R}\sum_{r=0}^{R}\sum_{\nu\in\mathcal{N}_u}\left((\widetilde{\mathbf{D}}^{-1/2}\widetilde{\mathbf{A}}\widetilde{\mathbf{D}}^{-1/2})^r\right)_{\nu,u}\mathbf{W}^{(r)}\mathcal{E}_w(\mathbf{x}_\nu), \quad \forall u\in\Omega, \tag{2}$$

where $\widetilde{\mathbf{A}} = \mathbf{A} + \mathbf{I} \in \{0,1\}^{N\times N}$ denotes the self-connected graph adjacency matrix and $\widetilde{\mathbf{D}}$ is the degree matrix of $\widetilde{\mathbf{A}}$; $\mathbf{W}^{(r)}$ is the trainable weight associated with the rth hop of neighbors; $\mathcal{P}_w(\cdot)$. The detailed derivation is presented in Appendix A.

When we maximize $C(\Omega)$ by training $\mathcal{E}_w(\cdot)$, $\mathcal{P}_w(\cdot)$ and $\mathcal{S}_w(\cdot,\cdot)$, we estimate the mutual information between vertices in Ω and their neighborhoods. This is similar to deep graph infomax (DGI) [48], which estimates the mutual information between any vertex feature and a global graph embedding. Both DGI and the proposed VIPool apply the techniques of mutual information neural estimation [2, 26] to the graph domain; however, there are two major differences. First, DGI aims to train a graph embedding function while VIPool aims to evaluate the importance of a vertex via its affinity to its neighborhood. Second, DGI considers the relationship between a vertex and an entire graph while VIPool learns the dependency between a vertex and a neighborhood. By varying the neighbor-hop R of $\mathcal{N}_u$ in Eq. (2), VIPool is able to tradeoff local and global information.

Solutions for vertex selection. To solve the vertex selection problem (1), we consider the submodularity of mutual information [9] and employ a greedy algorithm: we select the first vertex with maximum $C(\Omega)$ with $|\Omega| = 1$; and we next add a new vertex sequentially by maximizing $C(\Omega)$ greedily; however, it is computationally expensive to evaluate $C(\Omega)$ for two reasons: (i) for any

[2] The trainable parameters in $\mathcal{E}_w(\cdot)$, $\mathcal{P}_w(\cdot)$, and $\mathcal{S}_w(\cdot,\cdot)$ are not weight-shared.

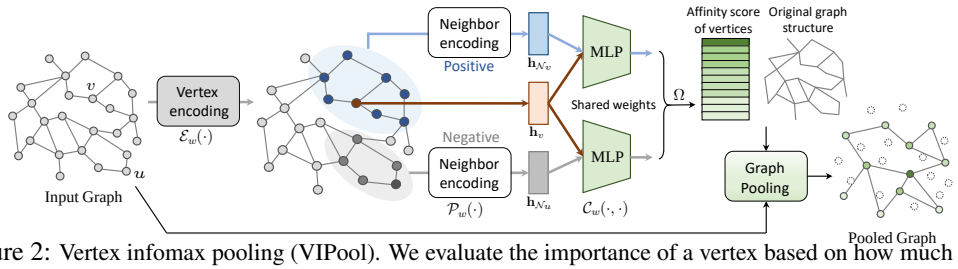


Figure 2: Vertex infomax pooling (VIPool). We evaluate the importance of a vertex based on how much it can reflect its own neighborhood and how much it can discriminate from an arbitrary neighborhood.

vertex set Ω, we need to solve an individual optimization problem; and (ii) the second term of $C(\Omega)$ includes all the pairwise interactions involved with quadratic computational cost. To address issue (i), we set the vertex set to all the vertices in the graph, maximize $\widehat{I}^{(\mathcal{V})}_{\mathrm{GAN}}$ to train $\mathcal{E}_w(\cdot)$, $\mathcal{P}_w(\cdot)$ and $\mathcal{S}_w(\cdot,\cdot)$. We then fix those three functions and evaluate $\widehat{I}^{(\Omega)}_{\mathrm{GAN}}$. To address issue (ii), we perform negative sampling, approximating the second term [36], where we sample negative neighborhoods $\mathcal{N}_u$ from the entire graph, whose number equals the number of positive vertex samples; that is, $|\Omega|$.

Graph pooling and unpooling. After solving problem. (1), we obtain Ω that contains K unique vertices selected from $\mathcal{V}$. To implement *graph pooling*, we further consider the distinct importance of different vertices in Ω, we compute an affinity score for each vertex based on its ability to describe its neighborhood. For vertex v with feature $\mathbf{x}_v$ and neighborhood feature $\mathbf{y}_{\mathcal{N}_v}$, the affinity score is

$$a_v \;=\; \sigma\left(\mathcal{S}_w(\mathcal{E}_w(\mathbf{x}_v), \mathcal{P}_w(\mathbf{y}_{\mathcal{N}_v}))\right) \in [0,1], \quad \forall v \in \Omega. \tag{3}$$

Eq. (3) considers the affinity only between a vertex and its own neighborhood, showing the degree of vertex-neighborhood information dependency. We collect a_v for $\forall v \in \Omega$ to form an affinity vector $\mathbf{a} \in [0,1]^K$. For graph data pooling, the pooled vertex feature $\mathbf{X}_\Omega = \mathbf{X}(\mathrm{id},:) \odot (\mathbf{a}\mathbf{1}^\top) \in \mathbb{R}^{K\times d}$, where id denotes selected vertices's indices that are originally in $\mathcal{V}$, $\mathbf{1}$ is an all-one vector, and $\odot$ denotes the element-wise multiplication. With the affinity vector $\mathbf{a}$, we assign an importance to each vertex and provide a path for back-propagation to flow gradients. As for graph structure pooling, we calculate $\mathbf{A}_\Omega = \mathrm{Pool}_A(\mathbf{A})$, and we consider three approaches to implement $\mathrm{Pool}_A(\cdot)$:

• Edge removal, i.e. $\mathbf{A}_\Omega = \mathbf{A}(\mathrm{id},\mathrm{id})$. This is simple, but loses significant structural information;

• Kron reduction [16], which is the Schur complement of the graph Laplacian matrix and preserves the graph spectral properties, but it is computationally expensive due to the matrix inversion;

• Cluster-connection, i.e. $\mathbf{A}_\Omega = \mathbf{S}\mathbf{A}\mathbf{S}^\top$ with $\mathbf{S} = \mathrm{softmax}(\mathbf{A}(\mathrm{id},:)) \in [0,1]^{K\times N}$. Each row of $\mathbf{S}$ represents the neighborhood of a selected vertex and the softmax function is applied for normalization. The intuition is to merge the neighboring information to the selected vertices [50].

Cluster-connection is our default implementation of the graph structure pooling. Figure 2 illustrates the overall process of vertex selection and graph pooling process.

To implement *graph unpooling*, inspired by [20], we design an inverse process against graph pooling. We initialize a zero matrix for the unpooled graph data, $\mathbf{X}' = \mathbf{O} \in \{0\}^{N\times d}$; and then, fill it by fetching the vertex features according to the original indices of retrained vertices; that is, $\mathbf{X}'(\mathrm{id},:) = \mathbf{X}_\Omega$. We then interpolate it through a graph propagation layer (implemented by graph convolution [30]) to propagate information from the vertices in Ω to the padded ones via the original graph structure.

4 Graph Cross Network

In this section, we propose the architecture of our *graph cross network* (GXN) for multiscale graph feature learning; see an exemplar model with 3 scales and 4 feature-crossing layers in Figure 3. The graph pooling/unpooling operations apply VIPool proposed in Section 3 and the graph propagation layers adopt the graph convolution layers [30]. A key ingredient of GXN is that we design *feature-crossing layers* to enhance multiscale information fusion. The entire GXN includes three stages: multiscale graphs generation, multiscale features extraction and multiscale readout.

Multiscale graphs generation. Given an input graph $G(\mathcal{V},\mathbf{A})$ with vertex features, $\mathbf{X} \in \mathbb{R}^{N\times d}$, we aim to create graph representations at multiple scales. We first employ a graph propagation layer on the input graph to initially embed the finest scale of graph as $G_0(\mathcal{V}_0,\mathbf{A}_0)$ with $\mathcal{V}_0 = \mathcal{V}$, $\mathbf{A}_0 = \mathbf{A}$ and vertex representations $\mathbf{X}_0$, where the graph propagation layer is implemented by a graph convolution

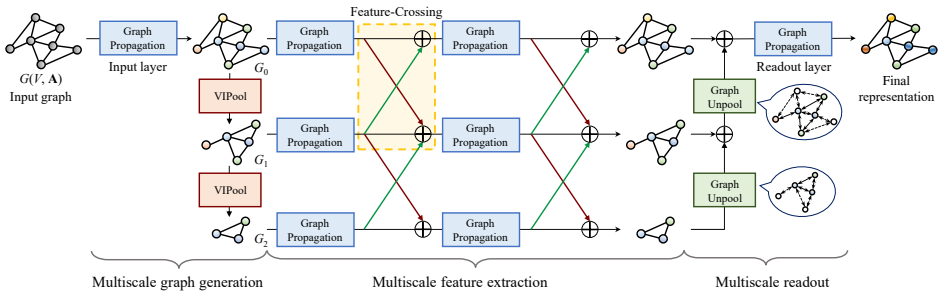


Figure 3: GXN architecture. We show an exemplar model with 3 scales and 4 feature-crossing layers.

layer [30]. We then recursively apply VIPool for S times to obtain a series of coarser scales of graph $G_1(\mathcal{V}_1, \mathbf{A}_1), \ldots, G_S(\mathcal{V}_S, \mathbf{A}_S)$ and corresponding vertex features $\mathbf{X}_1, \ldots, \mathbf{X}_S$ from G_0 and $\mathbf{X}_0$, respectively, where $|\mathcal{V}_s| > |\mathcal{V}_{s'}|$ for $\forall\, 1 \leq s < s' \leq S$.

Multiscale features extraction. Given multiscale graphs, we build a graph neural network at each scale to extract features. Each network consists of a sequence of graph propagation layers. To further enhance the information flow across scales, we propose feature-crossing layers between two consecutive scales at various network layers, which allows multiscale features to communicate and merge in the intermediate layers. Mathematically, at scale s and the any network layer, let the feature matrix of graph G_s be $\mathbf{X}_{\mathrm{h},s}$, the feature of graph G_{s-1} after pooling be $\mathbf{X}_{\mathrm{h},s}^{(\mathrm{p})}$, and the feature of graph G_{s+1} after unpooling be $\mathbf{X}_{\mathrm{h},s}^{(\mathrm{up})}$, the obtained vertex feature $\mathbf{X}'_{\mathrm{h},s}$ is formulated as

$$\mathbf{X}'_{\mathrm{h},s} = \mathbf{X}_{\mathrm{h},s} + \mathbf{X}_{\mathrm{h},s}^{(\mathrm{p})} + \mathbf{X}_{\mathrm{h},s}^{(\mathrm{up})}, \quad 0 < s < S.$$

For $s = 0$ or S, $\mathbf{X}_{\mathrm{h},s}$ is not fused by features from finer or coarser scales; see Figure 3. The graph pooling/unpooling here uses the same vertices as those obtained in multiscale graph generation to associate the vertices at different layers, but the affinity score $\mathbf{a}$ in each feature-crossing layer is trained independently to reflect the vertex importance at different levels. Note that the vertex-to-vertex association across scales is important here for feature-crossing and VIPool nicely fits it.

Multiscale readout. After multiscale feature extraction, we combine deep features at all the scales together to obtain the final representation. To align features at different scales, we adopt a sequence of graph unpooling operations implemented in the VIPool to transform all features to the original scale; see Figure 3. We finally leverage a readout graph propagation layer to further embed the fused multiscale features and generate the readout graph representation for various downstream tasks. In this work, we consider both graph classification and vertex classification.

Model Training. To train GXN, we consider the training loss with two terms: a graph-pooling loss $\mathcal{L}_{\mathrm{pool}} = -\widehat{I}_{\mathrm{GAN}}^{(\mathcal{V})}$ and a task-driven loss $\mathcal{L}_{\mathrm{task}}$. For graph classification, the task-driven loss is the cross-entropy loss between the predicted and ground-truth graph labels, $\mathcal{L}_{\mathrm{task}} = -\mathbf{y}^\top \log(\hat{\mathbf{y}})$, where $\mathbf{y}$ and $\hat{\mathbf{y}}$ are ground-truth label and predicted label of a graph; for vertex classification, it is the cross-entropy loss between the predictions and ground-truth vertex labels, $\mathcal{L}_{\mathrm{task}} = -\sum_{v \in \mathcal{V}_L} \mathbf{y}_v^\top \log(\hat{\mathbf{y}}_v)$, where $\mathbf{y}_v$ and $\hat{\mathbf{y}}_v$ are ground-truth and predicted vertex labels, and $\mathcal{V}_L$ contains labeled vertices. We finally define the overall loss as $\mathcal{L} = \mathcal{L}_{\mathrm{task}} + \alpha \mathcal{L}_{\mathrm{pool}}$, where the hyper-parameter α linearly decays per epoch from 2 to 0 during training, balancing a final task and vertex pooling[3].

5 Experimental Results

5.1 Datasets and Experiment Setup

Datasets. To test our GXN, we conduct extensive experiments for graph classification and vertex classification on several datasets. For **graph classification**, we use social network datasets: IMDB-B, IMDB-M and COLLAB [49], and bioinformatic datasets: D&D [15], PROTEINS [18], and ENZYMES [4]. Table 1 shows the dataset information. Note that no vertex feature is provided in three social network datasets, and we use one-hot vectors to encode the vertex degrees as vertex features, explicitly utilizing some structural information. We use the same dataset separation as in [20], perform 10-fold cross-validation, and show the average accuracy for evaluation. For **vertex**

[3] VIPool is trained through both $\mathcal{L}_{\mathrm{pool}}$ and $\mathcal{L}_{\mathrm{task}}$, which makes graph pooling adapt to a specific task.

Table 1: Graph classification accuracies (%) of different methods on different datasets. GXN (gPool) and GXN (SAGPool) denote that we apply previous pooling operations, gPool [20] and SAGPool [31] in our GXN framework, respectively. Various fashions of feature-crossing are presented, including fusion of coarse-to-fine (↑), fine-to-coarse (↓), no feature-crossing (noCross), and feature-crossing at early, late and all layers of networks.

Dataset	IMDB-B	IMDB-M	COLLAB	D&D	PROTEINS	ENZYMES
# Graphs (Classes)	1000 (2)	1500 (3)	5000 (3)	1178 (2)	1113 (2)	600 (6)
Avg. # Vertices	19.77	13.00	74.49	284.32	39.06	32.63
PatchySAN [37]	76.27 ± 2.6	69.70 ± 2.2	43.33 ± 2.8	72.60 ± 2.2	75.00 ± 2.8	-
ECC [43]	-	-	67.79	72.54	72.65	53.50
Set2Set [21]	-	-	71.75	78.12	74.29	60.15
DGCNN [53]	70.00 ± 0.9	47.83 ± 0.9	73.76 ± 0.5	79.37 ± 0.9	73.68 ± 0.9	-
DiffPool [50]	70.40	47.83	75.84	80.64	76.25	**62.53**
Graph U-Net [20]	72.10	48.33	77.56	82.43	77.68	58.57
SAGPool [31]	72.80	49.43	76.92	78.35	78.28	60.23
AttPool [27]	73.60	50.67	77.04	79.20	76.50	59.76
GXN	**77.60 ± 0.8**	**54.73 ± 0.9**	**79.19 ± 0.8**	**83.28 ± 1.3**	**79.38 ± 1.2**	58.63 ± 1.0
GXN (gPool)	76.40 ± 1.0	53.16 ± 0.6	77.73 ± 1.1	82.44 ± 1.4	78.44 ± 0.8	57.74 ± 1.3
GXN (SAGPool)	76.90 ± 0.7	52.74 ± 0.8	78.10 ± 0.8	83.07 ± 1.3	78.58 ± 1.1	57.87 ± 1.1
GXN (AttPool)	76.85 ± 0.9	53.62 ± 0.9	78.52 ± 0.9	82.93 ± 1.0	78.09 ± 1.3	57.45 ± 1.0
GXN (↑)	77.10 ± 0.6	54.22 ± 1.0	78.41 ± 0.8	82.65 ± 1.0	78.87 ± 0.8	57.48 ± 0.8
GXN (↓)	76.80 ± 1.1	54.08 ± 0.7	78.28 ± 1.0	82.30 ± 1.2	78.64 ± 1.2	56.95 ± 1.3
GXN (noCross)	74.80 ± 1.1	52.68 ± 0.9	77.58 ± 0.7	82.60 ± 0.9	78.26 ± 0.9	57.37 ± 1.2
GXN (early)	77.10 ± 0.6	54.27 ± 0.6	78.18 ± 0.8	83.13 ± 1.0	79.20 ± 1.0	58.13 ± 1.0
GXN (late)	76.30 ± 0.9	53.83 ± 1.0	77.88 ± 1.1	82.58 ± 1.5	79.03 ± 1.2	57.84 ± 0.9
GXN (all)	**77.60 ± 0.8**	**54.73 ± 0.9**	**79.19 ± 0.8**	**83.28 ± 1.3**	**79.38 ± 1.2**	58.63 ± 1.0

classification, we use three classical citation networks: Cora, Citeseer and Pubmed [30]. We perform both full-supervised and semi-supervised vertex classification; that is, for full-supervised classification, we label all the vertices in training sets for model training, while for semi-supervised, we only label a few vertices (around 7% on average) in training sets. We use the default separations of training/validation/test subsets. See more information of all used datasets in Appendix.

Model configuration. We implement GXN with PyTorch 1.0 on one GTX-1080Ti GPU. For **graph classification**, we consider three scales, which preserve 50% to 100% vertices from the original scales, respectively. For both input and readout layers, we use 1-layer GCNs; for multiscale feature extraction, we use 2 GCN layers followed by ReLUs at each scale and feature-crossing layers between any two consecutive scales at any layers. After the readout layers, we unify the embeddings of various graphs to the same dimension by using the same SortPool in DGCNN [53], AttPool [27] and Graph U-Net [20]. In the VIPool, we use a 2-layer MLP and R-layer GCN ($R = 1$ or 2) as $\mathcal{E}_w(\cdot)$ and $\mathcal{P}_w(\cdot)$, and use a linear layer as $\mathcal{S}_w(\cdot, \cdot)$. The hidden dimensions are 32. To improve the efficiency of solving problem (1), we modify $C(\Omega)$ by preserving only the first term. In this way, we effectively reduce the computational costs to sovle (1) from $\mathcal{O}(|V|^2)$ to $\mathcal{O}(|V|)$, and each vertex contributes the vertex set independently. The optimal solution is **top-**K vertices. We compare the outcomes of $C(\Omega)$ by the greedy algorithm and top-k method in Figure 5. For **vertex classification**, we use similar architecture as in graph classification, while the hidden feature are 128-dimension. We directly use the readout layer for vertex classification. In the loss function $\mathcal{L}$, α decays from 2 to 0 during training, where the VIPool needs fast convergence for vertex selection; and the model gradually focuses more on tasks based on the effective VIPool. We use Adam optimizer [14] and the learining rates range from 0.0001 to 0.001 for different datasets.

5.2 Comparison

Graph classification. We compare the proposed GXN to representative GNN-based methods, including PatchySAN [37], ECC [43], Set2Set [21], DGCNN [53], DiffPool [50], Graph U-Net [20], SAGPool [31], AttPool [27], and StructPool [51], where most of them performed multiscale graph feature learning. Additionally, we design several variants of GXN: 1) to test the superiority of VIPool, we apply gPool [20], SAGPool [31] and AttPool [27] in the same architecture of GXN, denoted as 'GXN (gPool)', 'GXN (SAGPool)' and 'GXN (AttPool)', respectively; 2) we investigate different feature-crossing mechanism, including various crossing directions and crossing positions. Table 1 compares the accuracies of various methods for graph classification. We see that our model outperforms the state-of-the-art methods on 5 out of 6 datasets, achieving an improvement by 1.96% on average accuracies. Besides, VIPool and more feature-crossing lead to better performance. We also show the qualitative results of vertex selection of different graph pooling methods in Appendix.

Table 2: Vertex classification accuracies (%) of different methods, where 'full-sup.' and 'semi-sup.' denote the scenarios of full-supervised and semi-supervised vertex classification, respectively.

Dataset	Cora		Citeseer		Pubmed	
# Vertices (Classes)	2708 (7)		3327 (6)		19717 (3)	
Supervision	full-sup.	semi-sup.	full-sup.	semi-sup.	full-sup.	semi-sup.
DeepWalk [39]	78.4 ± 1.7	67.2 ± 2.0	68.5 ± 1.8	43.2 ± 1.6	79.8 ± 1.1	65.3 ± 1.1
ChebNet [11]	86.4 ± 0.5	81.2 ± 0.5	78.9 ± 0.4	69.8 ± 0.5	88.7 ± 0.3	74.4 ± 0.4
GCN [30]	86.6 ± 0.4	81.5 ± 0.5	79.3 ± 0.5	70.3 ± 0.5	90.2 ± 0.3	79.0 ± 0.3
GAT [47]	87.8 ± 0.7	83.0 ± 0.7	80.2 ± 0.6	73.5 ± 0.7	90.6 ± 0.4	79.0 ± 0.3
FastGCN [7]	85.0 ± 0.8	80.8 ± 1.0	77.6 ± 0.8	69.4 ± 0.8	88.0 ± 0.6	78.5 ± 0.7
ASGCN [28]	87.4 ± 0.3	-	79.6 ± 0.2	-	90.6 ± 0.3	-
Graph U-Net [20]	-	84.4	-	73.2	-	79.6
GXN	**88.9 ± 0.4**	**85.1 ± 0.6**	**80.9 ± 0.4**	**74.8 ± 0.4**	**91.8 ± 0.3**	**80.2 ± 0.3**
GXN (noCross)	87.3 ± 0.4	83.2 ± 0.5	79.5 ± 0.4	73.7 ± 0.3	91.1 ± 0.2	79.6 ± 0.3

Vertex classification. We compare GXN to state-of-the-art methods: DeepWalk [39], GCN [30], GraphSAGE [23], FastGCN [7], ASGCN [28], and Graph U-Net [20] for vertex classification. We reproduce these methods for both full-supervised and semi-supervised learning based on their official codes. Table 8 compares the vertex classification accuracies of various methods. Considering both full-supervised and semi-supervised settings, we see that our model achieves higher average accuracy by 1.15%. We also test a degraded GXN without any feature-crossing layer, and we see that the feature-crossing layers improves the accuracies by 1.10% on average; see more results in Appendix.

5.3 Model Analysis

We further conduct detailed analysis about the GXN architecture and VIPool.

GXN architectures. We test the architecture with various graph scales and feature-crossing layers. Base on the dataset of IMDB-B for graph classification, we vary the number of graph scales from 1 to 5 and the number of feature-crossing layers from 1 to 3. We present the vertex classification results in Table 3. We see that the architecture with 3 graph scales and 2 feature-crossing layers leads to the best performance. Compared to use only 1 graph scale, using 2 graph scales significantly improve the graph classification accuracy by 2.53% on average, indicating the importance of multiscale representations. When we use more than 3 scales, the classification results tend to be stable, indicating the redundant scales. To keep the model efficiency and effectiveness, we adopt 3 scales of graphs. As for the number of feature-crossing layers, only using 1 feature-crossing layer do not provide sufficient information for graph classification; while using more than 2 feature-crossing layers tends to damage model performance due to the higher model complexity.

Hops of neighborhood in VIPool. To validate the effects of different ranges of neighborhood information in VIPool, we vary the neighbor-hops R in $\mathcal{P}_w(\cdot)$ from 1 to 5 and perform graph classification on D&D and IMDB-B. When R increases, we push a vertex to represent a bigger neighborhood with more global information. Figure 4 shows the graph classification accuracies with various R on the two datasets. We see that, for D&D, which include graphs with relatively larger sizes (see Table 1, line 3), when $R = 2$, the model achieves the best performance, reflecting that vertices could express their neighborhoods within $R = 2$; while for IMDB-B with smaller graphs, vertices tend to express their 1-hop neighbors better. This reflects that VIPool achieves a flexible trade-off between local and global information through varying R to adapt to various graphs.

Approximation of C function. In VIPool, to optimize problem (1) more efficiently, we substitute the original $C(\Omega)$ by only preserving the positive term $C_+(\Omega) = \sum_{v\in\Omega} \log \sigma(\mathcal{S}_w(\mathbf{h}_v, \mathbf{h}_{\mathcal{N}_v}))$ and maximize $C_+(\Omega)$ by selecting 'Top-K', which also obtains the optimal solution. To see the performance gap between the original and the accelerated versions, we compare the exact value of $C(\Omega)$

		# feature-cross layers		
	Accuracy	1	2	3
# scales	1	73.20	74.10	73.80
	2	76.00	76.50	76.20
	3	76.80	**77.30**	77.10
	4	76.70	77.20	77.20
	5	76.80	**77.30**	77.00

Table 3: Graph classification accuracies (%) with various scales and feature-crossing layers on IMDB-B.

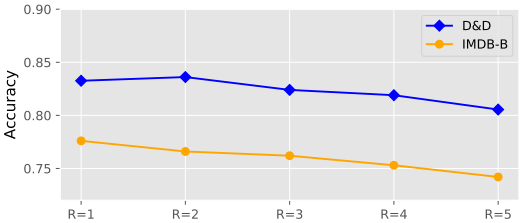
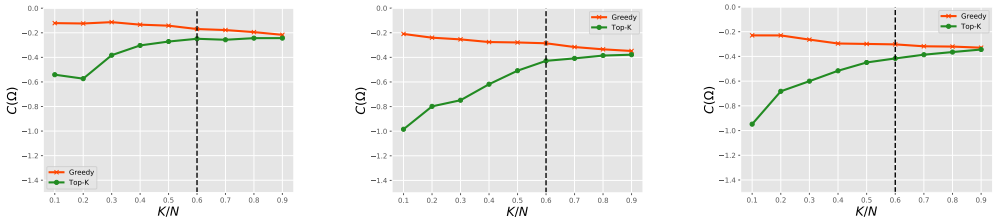


Figure 4: Graph classification accuracies (%) with neighbor-hops R from 1 to 5 on D&D and IMDB-B.



(a) PROTEINS (bioinformatic). (b) COLLAB (social). (c) Cora (citation).

Figure 5: Comparison of C values on different types of graph datasets.

Table 4: Vertex classification accuracies and time costs per epoch with different $\mathrm{Pool}_A(\cdot)$ on Cora.

$\mathrm{Pool}_A(\cdot)$	Accuracy (%)	Time (s)
Edge-Remove	84.6	0.44
Kron-Reduce	85.3	8.36
Clus-Connect	85.1	1.28

Table 5: Graph classification accuracies and time costs per epoch with different $\mathrm{Pool}_A(\cdot)$ on IMDB-B.

$\mathrm{Pool}_A(\cdot)$	Accuracy	Time (s)
Edge-Remove	77.20	1.97
Kron-Reduce	77.50	13.44
Clus-Connect	77.60	3.75

with the selected Ω by optimizing $C(\Omega)$ with greedy algorithm and optimizing $C_+(\Omega)$ with 'Top-K' method, respectively, on different types of datasets: bioinformatic, social and citation networks. We vary the ratio of the vertex selection among the global graph from 0.1 to 0.9. Figure 5 compares the $C(\Omega)$ as a function of selection ratio with two algorithms on 3 datasets, and the vertical dash lines denotes the boundaries where the value gaps equal to 10%. We see that, when we select small percentages (e.g. $< 60\%$) of vertices, the C value obtained by the greedy algorithm is much higher than 'Top-K' method; when we select more vertices, there are very small gaps between the two optimization algorithms, indicating two similar solutions of vertex selection. In GXN, we set the selection ratio above 60% in each graph pooling. More results about the model performances varying with ratios of vertices selection are presented in Appendix.

Graph structure pooling. In VIPool, we consider three implementations for graph structure pooling $\mathrm{Pool}_A(\cdot)$: edge-removal, Kron reduction and cluster-connection. We test these three operations for semi-supervised vertex classification on Cora and graph classification on IMDB-B, and we show the classification accuracies and time costs of the three graph structure pooling operations (denoted as 'Edge-Remove', 'Kron-Reduce' and 'Clus-Connect', respectively) in Tables 4 and 5. We see that Kron reduction or cluster-connection tend to provide the best accuracies on different datasets, but Kron reduction is significantly more expensive than the other two methods due to the matrix inversion. On the other hand, cluster-connection provides a better tradeoff between effectiveness and efficiency and we thus consider cluster-connection as our default choice.

6 Conclusions

This paper proposes a novel model *graph cross network* (GXN), where we construct parallel networks for feature learning at multiple scales of a graph and design novel feature-crossing layers to fuse intermediate features across multiple scales. To downsample graphs into various scales, we propose vertex infomax pooling (VIPool), selecting those vertices that maximally describe their neighborhood information. Based on the selected vertices, we coarsen graph structures and the corresponding graph data. VIPool is optimized based on the neural estimation of the mutual information between vertices and neighborhoods. Extensive experiments show that (i) GXN outperforms most state-of-the-art methods on graph classification and vertex classification; (ii) VIPool outperforms the other pooling methods; and (iii) more intermediate fusion across scales leads to better performances.

Broader Impact of Our Work

In this work, we aim to propose a method for multiscale feature learning on graphs, achieving two basic but challenging tasks: graph classification and vertex classification. This work has the following potential possible impacts to the society and the research community.

This work could be effectively used in many practical and important scenarios such as drug molecular analysis, social network mining, biometrics, human action recognition and motion prediction, etc., making our daily life more convenient and efficient. Due to the ubiquitous graph data, in most cases,

we can try to construct multiscale graphs to comprehensively obtain rich detailed, abstract, and even global feature representations, and effectively improve downstream tasks.

Our network structure can not only solve problem of feature learning with multiple graph scales, but also can be applied to the pattern learning of heterogeneous graphs, or other cross-modal or cross-view machine learning scenarios. This is of great significance for improving the ability of pattern recognition, feature transfer, and knowledge distillation to improve the computational efficiency.

At the same time, this work may have some negative consequences. For example, in social networks, it is uncomfortable even dangerous to use the models based on this work to over-mine the behavior of users, because the user's personal privacy and information security are crucial; companies should avoid mining too much users' personal information when building social platforms, keeping a safe internet environment.

References

[1] V. Badrinarayanan, A. Kendall, and R. Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence (IEEE TPAMI)*, 39(12):2481–2495, December 2017. ISSN 1939-3539. doi: 10.1109/TPAMI.2016.2644615.

[2] Mohamed Ishmael Belghazi, Aristide Baratin, Sai Rajeshwar, Sherjil Ozair, Yoshua Bengio, Aaron Courville, and Devon Hjelm. Mutual information neural estimation. In *Proceedings of the International Conference on Machine Learning (ICML)*, July 2018.

[3] Monica. Bianchini, Maria Cristina Gori, and Franco Scarselli. Processing directed acyclic graphs with recursive neural networks. *IEEE Transactions on Neural Networks (TNN)*, 12(6):1464–1470, February 2001. ISSN 1045-9227. doi: 10.1109/72.963781.

[4] Karsten M. Borgwardt, Cheng Soon Ong, Stefan Schönauer, S. V. N. Vishwanathan, Alex J. Smola, and Hans-Peter Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 21(1):i47–i56, March 2005. doi: 10.1093/bioinformatics/bti1007.

[5] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. In *International Conference on Learning Representations (ICLR)*, April 2014.

[6] Haochen Chen, Bryan Perozzi, Yifan Hu, and Steven Skiena. Harp: Hierarchical representation learning for networks. In *AAAI Conference on Artificial Intelligence (AAAI)*, February 2018.

[7] Jie Chen, Tengfei Ma, and Cao Xiao. FastGCN: Fast learning with graph convolutional networks via importance sampling. In *International Conference on Learning Representations (ICLR)*, April 2018.

[8] Siheng Chen, Rohan Varma, Aliaksei Sandryhaila, and Jelena Kovačević. Discrete signal processing on graphs: Sampling theory. *IEEE Trans. Signal Process.*, 63(24):6510–6523, 2015.

[9] Yuxin Chen, S. Hamed Hassani, Amin Karbasi, and Andreas Krause. Sequential information maximization: When is greedy near-optimal? In *Proceedings of The 28th Conference on Learning Theory, COLT 2015, Paris, France, July 3-6, 2015*, volume 40 of *JMLR Workshop and Conference Proceedings*, pages 338–363. JMLR.org, 2015.

[10] Hanjun Dai, Bo Dai, and Le Song. Discriminative embeddings of latent variable models for structured data. In *Proceedings of the International Conference on Machine Learning (ICML)*, June 2016.

[11] Michael Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems (NeurIPS)*. December 2016.

[12] Chenhui Deng, Zhiqiang Zhao, Yongyu Wang, Zhiru Zhang, and Zhuo Feng. Graphzoom: A multi-level spectral approach for accurate and scalable graph embedding. In *International Conference on Learning Representations (ICLR)*, April 2020.

[13] Inderjit S. Dhillon, Yuqiang Guan, and Brian Kulis. Weighted graph cuts without eigenvectors a multilevel approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence (IEEE TPAMI)*, 29(11): 1944–1957, November 2007. ISSN 1939-3539. doi: 10.1109/TPAMI.2007.1115.

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

[15] Paul D. Dobson and Andrew J. Doig. Distinguishing enzyme structures from non-enzymes without alignments. *Journal of Molecular Biology (JMB)*, 330(4):771–783, July 2003. doi: 10.1016/S0022-2836(03)00628-4.

[16] Florian Dörfler and Francesco Bullo. Kron reduction of graphs with applications to electrical networks. *IEEE Trans. Circuits Syst. I Regul. Pap.*, 60-I(1):150–163, 2013.

[17] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alan Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in Neural Information Processing Systems (NeurIPS)*. December 2015.

[18] Aasa Feragen, Niklas Kasenburg, Jens Petersen, Marleen de Bruijne, and Karsten Borgwardt. Scalable kernels for graphs with continuous attributes. In *Advances in Neural Information Processing Systems 26*. December 2013.

[19] Guoji Fu, Chengbin Hou, and Xin Yao. Learning topological representation for networks via hierarchical sampling. *CoRR*, abs/1902.06684, 2019.

[20] Hongyang Gao and Shuiwang Ji. Graph u-nets. In *Proceedings of the International Conference on Machine Learning (ICML)*, July 2019.

[21] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *Proceedings of the International Conference on Machine Learning (ICML)*, August 2017.

[22] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 2672–2680. December 2014.

[23] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems (NeurIPS)*. Decemeber 2017.

[24] Junjun He, Zhongying Deng, and Yu Qiao. Dynamic multi-scale filters for semantic segmentation. In *The IEEE International Conference on Computer Vision (ICCV)*, October 2019.

[25] Jean-Baptiste Hiriart-Urruty and Claude Lemarechal. Fundamentals of convex analysis. Springer, 2012.

[26] R Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Phil Bachman, Adam Trischler, and Yoshua Bengio. Learning deep representations by mutual information estimation and maximization. In *International Conference on Learning Representations (ICLR)*, April 2019.

[27] Jingjia Huang, Zhangheng Li, Nannan Li, Shan Liu, and Ge Li. Attpool: Towards hierarchical feature representation in graph convolutional networks via attention mechanism. In *The IEEE International Conference on Computer Vision (ICCV)*, October 2019.

[28] Wenbing Huang, Tong Zhang, Yu Rong, and Junzhou Huang. Adaptive sampling towards fast graph representation learning. In *Advances in Neural Information Processing Systems (NeurIPS)*. December 2018.

[29] Thomas Kipf and Max Welling. Variational graph auto-encoders. In *Advances in Neural Information Processing Systems (NeurIPS), Workshop*, December 2016.

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

[31] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. Self-attention graph pooling. In *Proceedings of the International Conference on Machine Learning (ICML)*, July 2019.

[32] Tao Lei, Wengong Jin, Regina Barzilay, and Tommi Jaakkola. Deriving neural architectures from sequence and graph kernels. In *Proceedings of the International Conference on Machine Learning (ICML)*.

[33] Maosen Li, Siheng Chen, Xu Chen, Ya Zhang, Yanfeng Wang, and Qi Tian. Actional-structural graph convolutional networks for skeleton-based action recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[34] Jiongqian Liang, Saket Gurukar, and Srinivasan Parthasarathy. Mile: A multi-level framework for scalable graph embedding. *CoRR*, abs/1802.09612, 2018.

[35] Renjie Liao, Zhizhen Zhao, Raquel Urtasun, and Richard Zemel. Lanczosnet: Multi-scale deep graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, April 2019.

[36] 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 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*, pages 3111–3119, 2013.

[37] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkovl. Learning convolutional neural networks for graphs. In *Proceedings of the International Conference on Machine Learning (ICML)*, June 2016.

[38] Sebastian Nowozin, Botond Cseke, and Ryota Tomioka. f-gan: Training generative neural samplers using variational divergence minimization. In *Advances in Neural Information Processing Systems (NeurIPS)*. December 2016.

[39] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. *CoRR*, abs/1403.6652, 2014.

[40] Sungmin Rhee, Seokjun Seo, and Sun Kim. Hybrid approach of relation network and localized graph convolutional filtering for breast cancer subtype classification. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, July 2018.

[41] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. Unet: Convolutional networks for biomedical image segmentation. In *In International Conference on Medical Image Computing and Computer-assisted Intervention (MICCAI)*, October 2015.

[42] Ilya Safro, Peter Sanders, and Christian Schulz. Advanced coarsening schemes for graph partitioning. *ACM Journal of Experimental Algorithmics*, 19(1), 2014.

[43] Martin Simonovsky and Nikos Komodakis. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3693–3702, July 2017.

[44] Fan-Yun Sun, Jordan Hoffman, Vikas Verma, and Jian Tang. Infograph: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization. In *International Conference on Learning Representations (ICLR)*, April 2020.

[45] Ke Sun, Bin Xiao, Dong Liu, and Jingdong Wang. Deep high-resolution representation learning for human pose estimation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[46] Petar Veličković, Duo Wang, Nicholas D. Lane, and Pietro Liò. X-cnn: Cross-modal convolutional neural networks for sparse datasets. In *IEEE Symposium Series on Computational Intelligence (SSCI)*, December 2016.

[47] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations (ICLR)*, April 2018.

[48] Petar Veličković, William Fedus, William L. Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. Deep graph infomax. In *International Conference on Learning Representations (ICLR)*, April 2019.

[49] Pinar Yanardag and S.V. N. Vishwanathan. A structural smoothing framework for robust graph comparison. In *Advances in Neural Information Processing Systems 28*. December 2015.

[50] Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. In *Advances in Neural Information Processing Systems (NeurIPS)*. December 2018.

[51] Hao Yuan and Shuiwang Ji. Structpool: Structured graph pooling via conditional random fields. In *International Conference on Learning Representations (ICLR)*, April 2020.

[52] Marc Brockschmidt Richard Zemel Yujia Li, Daniel Tarlow. Gated graph sequence neural networks. In *International Conference on Learning Representations (ICLR)*, May 2016.

[53] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *AAAI Conference on Artificial Intelligence (AAAI)*, February 2018.

[54] Rui Zhang, Sheng Tang, Yongdong Zhang, Jintao Li, and Shuicheng Yan. Scale-adaptive convolutions for scene parsing. In *The IEEE International Conference on Computer Vision (ICCV)*, October 2017.

Appendix A. Mutual Information Neural Estimation for Vertex Selection

An individual vertex is fully identified through its feature, which works as the vertex attribute. Given an vertex set $\mathcal{V}$ that contains all the vertices on the graph and a vertex subset $\Omega \subset \mathcal{V}$ which contains the selected vertices, we let a random variable $\mathbf{v}$ to represent the vertex feature when we randomly pick a vertex from Ω. Then we define the probability distribution of $\mathbf{v}$ as

$$P_{\mathbf{v}} \;=\; P(\mathbf{v} = \mathbf{x}_v), \quad \forall v \in \Omega$$

where $\mathbf{x}_v$ is the feature value when we pick vertex v.

The neighborhood of any vertex $u \in \Omega$ is defined as $\mathcal{N}_u$, which is the subgraph containing the vertices in $\mathcal{V}$ whose geodesic distances to the central vertex u are no greater than a threshold R, i.e. $\mathcal{N}_v = G(\{u\}_{d(u,v)\leq R}, \mathbf{A}_{\{u\},\{u\}})$. Let a random variable $\mathbf{n}$ be the neighborhood features when we randomly pick a central vertex from Ω, then we define the probability distribution of $\mathbf{n}$ as

$$P_{\mathbf{n}} \;=\; P(\mathbf{n} = \mathbf{y}_{\mathcal{N}_u}), \quad \forall u \in \Omega$$

where $\mathbf{y}_{\mathcal{N}_u} = [\mathbf{A}_{\mathcal{N}_u,\mathcal{N}_u}, \{\mathbf{x}_\nu\}_{\nu\in\mathcal{N}_u}]$ denotes the neighborhood feature value when we pick vertex u's neighborhood, including both the internal connectivity information and contained vertex features in the neighborhood $\mathcal{N}_u$.

Therefore, we define the joint distribution of the random variables of vertex features and neighborhood features, which is formulated as

$$P_{\mathbf{v},\mathbf{n}} \;=\; P(\mathbf{v} = \mathbf{x}_v, \mathbf{n} = \mathbf{y}_{\mathcal{N}_v}), \quad \forall v \in \Omega$$

where the joint distribution reflects probability that we randomly pick the corresponding vertex feature and neighborhood feature of the same vertex v together.

The *mutual information* between the vertex features and the neighborhood features is defined as the KL-divergence between the joint distribution $P_{\mathbf{v},\mathbf{n}}$ and the product of the marginal distributions of two random variable, $P_{\mathbf{v}} \otimes P_{\mathbf{n}}$; that is

$$I^{(\Omega)}(\mathbf{v},\mathbf{n}) \quad = \quad D_{\mathrm{KL}}\left(P_{\mathbf{v},\mathbf{n}} || P_{\mathbf{v}} \otimes P_{\mathbf{n}}\right),$$

This mutual information measures of the mutual dependency between vertices and neighborhoods in the selected vertex subset Ω. The KL divergence admits the f-representation [2],

$$D_{\mathrm{KL}}\left(P_{\mathbf{v},\mathbf{n}} || P_{\mathbf{v}} \otimes P_{\mathbf{n}}\right) \;\geq\; \sup_{T\in\mathcal{T}} \left\{ \mathbb{E}_{\mathbf{x}_v,\mathbf{y}_{\mathcal{N}_v}\sim P_{\mathbf{v},\mathbf{n}}}\left[T(\mathbf{x}_v,\mathbf{y}_{\mathcal{N}_v})\right] - \mathbb{E}_{\mathbf{x}_v\sim P_{\mathbf{v}},\mathbf{y}_{\mathcal{N}_u}\sim P_{\mathbf{n}}}\left[e^{T(\mathbf{x}_v,\mathbf{y}_{\mathcal{N}_u})-1}\right] \right\}, \quad (4)$$

where $\mathcal{T}$ is an arbitrary class of functions that maps a pair of vertex features and neighborhood features to a real value, and here we use $T(\cdot,\cdot)$ to compute the dependency of two features. It could be a tight lower-bound of mutual information if we search any possible function $T \in \mathcal{T}$.

Note that the main target here is to propose a vertex-selection criterion based on quantifying the dependency between vertices and neighborhood. Therefore instead of computing the exact mutual information based on KL divergence, we can use non-KL divergences to achieve favourable flexibility and convenience in optimization. Both non-KL and KL divergences can be formulated based on the same f-representation framework. Here we start from the general f-divergence between the joint distribution and the product of marginal distributions of vertices and neighborhoods.

$$D_f\left(P_{\mathbf{v},\mathbf{n}} || P_{\mathbf{v}} \otimes P_{\mathbf{n}}\right) \;=\; \int P_{\mathbf{v}} P_{\mathbf{n}} f\left(\frac{P_{\mathbf{v},\mathbf{n}}}{P_{\mathbf{v}} P_{\mathbf{n}}}\right) d\mathbf{x}_v d\mathbf{y}_{\mathcal{N}_v}$$

where $f(\cdot)$ is a convex and lower-semicontinuous divergence function. when $f(x) = x\log x$, the f-divergence is specificed as KL divergence. The function $f(\cdot)$ has a convex conjugate function $f^*(\cdot)$, i.e. $f^*(t) = \sup_{x\in\mathrm{dom}_f}\{xt - f(x)\}$, where dom_f is the definition domain of $f(\cdot)$. Note that the two functions $f(\cdot)$ and $f^*(\cdot)$ is dual to each other. According to the Fenchel conjugate [25], the f-divergence can be modified as

$$\begin{aligned} D_f(P_{\mathbf{v},\mathbf{n}} || P_{\mathbf{v}} \otimes P_{\mathbf{n}}) \quad &= \quad \int P_{\mathbf{x}} P_{\mathbf{n}} \sup_{t\in\mathrm{dom}_{f^*}} \left\{ t\frac{P_{\mathbf{x},\mathbf{n}}}{P_{\mathbf{v}} P_{\mathbf{n}}} - f^*(t) \right\} \\ &\geq \quad \sup_{T\in\mathcal{T}} \left\{ \mathbb{E}_{P_{\mathbf{v},\mathbf{n}}}[T(\mathbf{x}_v,\mathbf{y}_{\mathcal{N}_v})] - \mathbb{E}_{P_{\mathbf{v}},P_{\mathbf{n}}}[f^*(T(\mathbf{x}_v,\mathbf{y}_{\mathcal{N}_u}))] \right\} \end{aligned}$$

where $\mathcal{T}$ denotes any functions that map vertex and neighborhood features to a scalar, and the function $T(\cdot,\cdot)$ works as a variational representation of t. We further use an activation function $a:\mathbb{R}\to \mathrm{dom}_{f^*}$ to constrain the function value; that is $T(\cdot,\cdot)\to a(T(\cdot,\cdot))$. Therefore, we have

$$D_f(P_{\mathbf{v},\mathbf{n}}||P_{\mathbf{v}}\otimes P_{\mathbf{n}}) \quad \geq \quad \sup_{T\in\mathcal{T}}\left\{\mathbb{E}_{P_{\mathbf{v},\mathbf{n}}}[a(T(\mathbf{x}_v,\mathbf{y}_{\mathcal{N}_v}))]-\mathbb{E}_{P_{\mathbf{v}},P_{\mathbf{n}}}[f^*(a(T(\mathbf{x}_v,\mathbf{y}_{\mathcal{N}_u})))]\right\}$$

since the $a(T(\cdot,\cdot))$ is also in $\mathcal{T}$ and its value is in dom_{f^*}, the optimal solution satisfies the equation. Suppose that the divergence function is $f(x)=x\log x$, the conjugate divergence function is $f^*(t)=\exp(t-1)$ and the activation function is $a(x)=x$, we can obtain the f-representation of KL divergence; see Eq. (4). Note that the form of activation function is not unique, and we aim to find a proper one that helps to derivation and computation.

Here, we consider another form of divergence based on f-representation; that is, GAN-like divergence, where we have specific form of divergence function $f(x)=x\log x-(x+1)\log(x+1)$ and conjugate divergence function $f^*(t)=-\log(1-\exp(t))$ [38]. We let the activation be $a(\cdot)=-\log(1+\exp(\cdot))$. The GAN-like divergence is formulated as

$$\begin{aligned}
& D_{\mathrm{GAN}}\left(P_{\mathbf{v},\mathbf{n}}||P_{\mathbf{v}}\otimes P_{\mathbf{n}}\right)\\
\geq\ & \sup_{T\in\mathcal{T}}\left\{\mathbb{E}_{P_{\mathbf{v},\mathbf{n}}}[a(T(\mathbf{x}_v,\mathbf{y}_{\mathcal{N}_v}))]-\mathbb{E}_{P_{\mathbf{v}},P_{\mathbf{n}}}[f^*(a(T(\mathbf{x}_v,\mathbf{y}_{\mathcal{N}_u})))]\right\}\\
=\ & \sup_{T\in\mathcal{T}}\left\{\mathbb{E}_{P_{\mathbf{v},\mathbf{n}}}[-\log(1+\exp(-T(\mathbf{x}_v,y_{\mathcal{N}_v})))]+\mathbb{E}_{P_{\mathbf{v}},P_{\mathbf{n}}}\log(1-\exp(-\log(1+e^{T(\mathbf{x}_v,\mathbf{y}_{\mathcal{N}_u})})))\right\}\\
=\ & \sup_{T\in\mathcal{T}}\left\{\mathbb{E}_{P_{\mathbf{v},\mathbf{n}}}\log\frac{1}{1+e^{-T(\mathbf{x}_v,y_{\mathcal{N}_v})}}+\mathbb{E}_{P_{\mathbf{v}},P_{\mathbf{n}}}\log(1-\frac{1}{1+e^{-T(\mathbf{x}_v,\mathbf{y}_{\mathcal{N}_u})}})\right\}\\
=\ & \sup_{T\in\mathcal{T}}\left\{\mathbb{E}_{P_{\mathbf{v},\mathbf{n}}}\left[\log\sigma\left(T(\mathbf{x}_v,\mathbf{y}_{\mathcal{N}_v})\right)\right]+\mathbb{E}_{P_{\mathbf{v}},P_{\mathbf{n}}}\left[\log\left(1-\sigma\left(T\left(\mathbf{x}_v,\mathbf{y}_{\mathcal{N}_u}\right)\right)\right)\right]\right\}
\end{aligned}$$

where $\sigma(\cdot)$ is the sigmoid function that maps a real value into the range of $(0,1)$. Eventually, the GAN-like divergence converts the f-divergence to a binary cross entropy, which is similar to the objective function to train the discriminator in GAN [22].

To determine the form of the function $T(\cdot,\cdot)$, we parameterized $T(\cdot,\cdot)$ by trainable neural networks rather than design it manually. The parameterized function is denoted as $T_w(\cdot,\cdot)$, where w generally denotes the parameterization. In this work, $T_w(\cdot,\cdot)$ is constructed with three trainable functions: 1) A vertex embedding function $\mathcal{E}_w(\cdot)$; 2) A neighborhood embedding function $P_w(\cdot)$; and 3) a vertex-neighborhood affinity function $C_w(\cdot,\cdot)$; which are formulated as

$$\begin{aligned}
T_w(\mathbf{x}_v,\mathbf{y}_{\mathcal{N}_u}) &= \mathcal{S}_w(\mathcal{E}_w(\mathbf{x}_v),\mathcal{P}_w(\mathbf{y}_{\mathcal{N}_u}))\\
&= \mathcal{S}_w\left(\mathcal{E}_w(\mathbf{x}_v),\frac{1}{R}\sum_{r=0}^{R}\sum_{\nu\in\mathcal{N}_u}\left((\widetilde{\mathbf{D}}^{-1/2}\widetilde{\mathbf{A}}\widetilde{\mathbf{D}}^{-1/2})^r\right)_{\nu,u}\mathbf{W}^{(r)}\mathcal{E}_w(\mathbf{x}_\nu)\right).
\end{aligned}$$

where $\mathcal{E}_w(\cdot)$ is modeled by a Multi-layer perceptron (MLP), $\mathcal{P}_w(\cdot)$ is modeled by a R-hop graph convolution layer and $\mathcal{S}_w(\cdot,\cdot)$ is also modeled by an MLP. In $\mathcal{P}_w(\cdot)$, $\widetilde{\mathbf{A}}=\mathbf{A}+\mathbf{I}$ is the self-connected graph adjacency matrix and $\widetilde{\mathbf{D}}$ is the degree matrix of $\widetilde{\mathbf{A}}$; $\mathbf{W}^{(r)}\in\mathbb{R}^{d\times d}$ is the trainable weight matrix associated with the rth hop of neighborhood. The neighborhood embedding function $\mathcal{P}_w(\cdot)$ aggregates neighborhood information with in a geodesic distance threshold R. Note that $\mathcal{P}_w(\cdot)$ separately use neighborhood features $\mathbf{y}_{\mathcal{N}_u}$ in form of connectivity information and vertex features.

In this way, the GAN-like-divergence-based mutual information between graph vertices and neighborhoods can be represented with the parameterized GAN-like divergence, which is a variational divergence and works as a lower bound of of the theorical GAN-like-divergence-based mutual information; that is,

$$\begin{aligned}
I^{(\Omega)}_{\mathrm{GAN}}(\mathbf{v},\mathbf{n}) &= D_{\mathrm{GAN}}(P_{\mathbf{v},\mathbf{n}}||P_{\mathbf{v}}\otimes P_{\mathbf{n}}) \ \geq\ \widehat{I}^{(\Omega)}_{\mathrm{GAN}}(\mathbf{v},\mathbf{n})\\
&= \max_w\left\{\mathbb{E}_{P_{\mathbf{v},\mathbf{n}}}\left[\log\sigma\left(T_w(\mathbf{x}_v,\mathbf{y}_{\mathcal{N}_v})\right)\right]+\mathbb{E}_{P_{\mathbf{v}},P_{\mathbf{n}}}\left[\log\left(1-\sigma\left(T_w\left(\mathbf{x}_v,\mathbf{y}_{\mathcal{N}_u}\right)\right)\right)\right]\right\}\\
&= \max_w\frac{1}{|\Omega|}\sum_{v\in\Omega}\log\sigma(\mathcal{T}_w(\mathbf{x}_v,\mathbf{y}_{\mathcal{N}_v}))+\frac{1}{|\Omega|^2}\sum_{(v,u)\in\Omega}\log(1-\sigma(\mathcal{T}_w(\mathbf{x}_v,\mathbf{y}_{\mathcal{N}_u})))
\end{aligned}$$

Table 6: The detailed information of graph datasets used in the experiments of graph classification

Dataset	IMDB-B	IMDB-M	COLLAB	D&D	PROTEINS	ENZYMES
# Graphs	1000	1500	5000	1178	1113	600
# Classes	2	3	3	2	2	6
Max # Vertices	139	89	492	5748	620	126
Min # Vertices	12	7	32	30	4	2
Avg. # Vertices	19.77	13.00	74.49	284.32	39.06	32.63
# Train Graphs	900	1350	4501	1061	1002	540
# Test Graphs	100	150	499	111 117		60
Vertex Dimensions	1	1	1	82	3	3
Max Degrees	66	60	370	-	-	-

Table 7: The detailed information of graph datasets used in the experiments of vertex classification

Dataset	Cora	Citeseer	Pubmed
# Vertices	2708	3327	19717
# Edges	5429	4732	44338
# Classes	7	6	3
Vertex Dimension	1433	3703	500
# Train Vertices (full-sup.)	1208	1827	18217
# Train Vertices (semi-sup.)	140	120	60
# Valid. Vertices	500	500	500
# Test Vertices	1000	1000	1000

Since we consider the dependency between vertices and neighborhoods within a specific vertex set, the possible outcomes of the joint distribution and the two marginal distributions are countable. We thus use the summation to aggregate all the possible cases. To maximize $\widehat{I}^{(\Omega)}_{\mathrm{GAN}}(\mathbf{v}, \mathbf{n})$ by training the internal function in $T_w(\cdot, \cdot)$, that is, $\mathcal{E}_w(\cdot)$, $\mathcal{P}_w(\cdot)$, and $\mathcal{S}_w(\cdot, \cdot)$, we can maximally approximate the mutual information between individual vertex and neighborhood for vertex selection in our VIPool. Note that the value of $\widehat{I}^{(\Omega)}_{\mathrm{GAN}}(\mathbf{v}, \mathbf{n})$ is not very close to the exact KL-divergence-based mutual information, but it has the consistency to $I^{(\Omega)}(\mathbf{v}, \mathbf{n})$ to reflect the pair of vertex-neighborhood with high or low mutual information, leading to effective vertex selection.

Appendix B. Detailed Information of Experimental Graph Datasets

Here we show more details about the graph datasets used in our experiments of both graph classification and vertex classification. We first show the six datasets for graph classification in Table 6. We see that, we show the numbers of graphs, graph classes, vertices, numbers of graphs in training/test datasets and feature dimensions of all the six datasets. Note that, three social network datasets, IMDB-B, IMDB-M and COLLAB do not provide specific vertex features, where the vertex dimension is denoted as 1 and the maximum vertex degrees are shown in addition. In our experiments, we use one-hot vectors to encode the vertex degrees in these three datasets as their vertex features which explicitly contains the structure information.

We then show the details of three citation network datasets used in the experiments of vertex classification in Table 7. We see that, we present the numbers of vertices, edges, vertex classes and feature dimensions of the three datasets, as well as we show the separations of training/validation/test sets, where '# Train Vertices (full-sup.)' denotes the number of training vertices for full-supervised vertex classification and '# Train Vertices (semi-sup.)' denotes the number of training vertices for semi-supervised vertex classification.

Appendix C. More GXN Variants for Vertex Classification

Here we show more results of vertex classification of more variants of the proposed GXN associated with different pooling methods; that is, we test different pooling methods with the same GXN model framework, where the pooling methods include gPool [20], SAGPool [31] and AttPool [27]. The full-supervised and semi-supervised vertex classification accuracies of different algorithms on three

Table 8: Vertex classification accuracies (%) of different methods, where 'full-sup.' and 'semi-sup.' denote the scenarios of full-supervised and semi-supervised vertex classification, respectively.

Dataset	Cora		Citeseer		Pubmed	
# Vertices (Classes)	2708 (7)		3327 (6)		19717 (3)	
Supervision	full-sup.	semi-sup.	full-sup.	semi-sup.	full-sup.	semi-sup.
DeepWalk [39]	78.4 ± 1.7	67.2 ± 2.0	68.5 ± 1.8	43.2 ± 1.6	79.8 ± 1.1	65.3 ± 1.1
ChebNet [11]	86.4 ± 0.5	81.2 ± 0.5	78.9 ± 0.4	69.8 ± 0.5	88.7 ± 0.3	74.4 ± 0.4
GCN [30]	86.6 ± 0.4	81.5 ± 0.5	79.3 ± 0.5	70.3 ± 0.5	90.2 ± 0.3	79.0 ± 0.3
GAT [47]	87.8 ± 0.7	83.0 ± 0.7	80.2 ± 0.6	73.5 ± 0.7	90.6 ± 0.4	79.0 ± 0.3
FastGCN [7]	85.0 ± 0.8	80.8 ± 1.0	77.6 ± 0.8	69.4 ± 0.8	88.0 ± 0.6	78.5 ± 0.7
ASGCN [28]	87.4 ± 0.3	-	79.6 ± 0.2	-	90.6 ± 0.3	-
Graph U-Net [20]	-	84.4	-	73.2	-	79.6
GXN	**88.9 ± 0.4**	**85.1 ± 0.6**	**80.9 ± 0.4**	**74.8 ± 0.4**	**91.8 ± 0.3**	**80.2 ± 0.3**
GXN (gPool)	88.0 ± 0.4	84.4 ± 0.6	79.7 ± 0.5	74.4 ± 0.6	90.6 ± 0.4	79.8 ± 0.4
GXN (SAGPool)	87.8 ± 0.6	84.7 ± 0.4	80.0 ± 0.5	74.2 ± 0.4	90.9 ± 0.3	80.1 ± 0.3
GXN (AttPool)	88.4 ± 0.3	84.6 ± 0.5	80.6 ± 0.4	74.5 ± 0.5	91.3 ± 0.3	**80.2 ± 0.4**

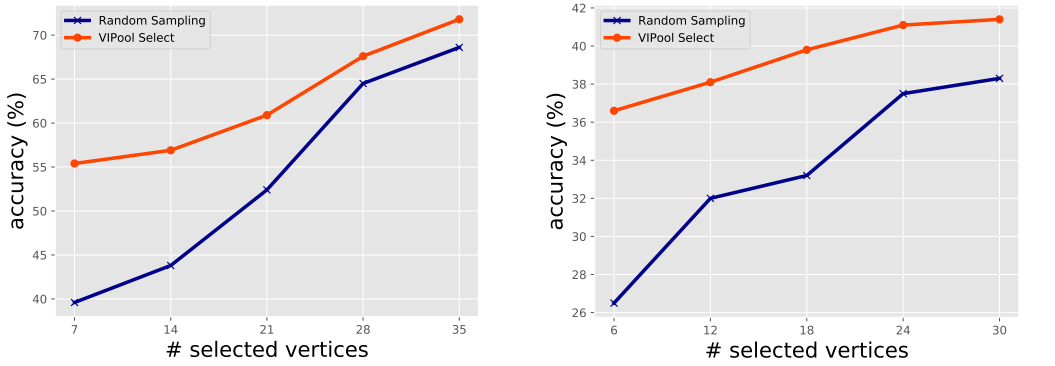
Table 9: Graph classification accuracices of GXN models with different ratios of vertex selection in VIPool and different optimization algorithms to select informative vertex subset on IMDB-B dataset. Note that some entries are missing due to too expensive computational cost.

Ratios of vertex selection		Classification accuracy (%)	
ratio of scale 1	ratio of scale 2	greedy algorithm	top-K method
0.9	0.7	-	76.6 ± 0.8
0.9	0.5	-	76.7 ± 1.0
0.9	0.3	-	76.4 ± 0.8
0.9	0.1	-	76.3 ± 0.9
0.8	0.7	-	77.5 ± 0.9
0.8	0.5	-	**77.6 ± 0.9**
0.8	0.3	-	77.3 ± 0.9
0.8	0.1	-	76.4 ± 0.9
0.7	0.5	77.2 ± 0.9	76.6 ± 1.0
0.7	0.3	77.1 ± 0.8	76.3 ± 0.9
0.7	0.1	76.8 ± 0.9	75.8 ± 0.7
0.5	0.3	77.0 ± 0.7	76.1 ± 0.7
0.5	0.1	76.8 ± 0.8	75.6 ± 1.0
0.3	0.1	76.7 ± 1.0	74.9 ± 0.8
Only one scale		74.1 ± 0.5	

citation networks are shown in Table 8. We see that, comprared to the previous pooling methods, the proposed GXN which uses VIPool could provide higher average classification accuracies for both full-supervised and semi-supervised vertex classification. Different GXN variants with different pooling methods tend to consistently outperform most state-of-the-art models for vertex classification, reflecting the effectiveness of the proposed GXN architecture.

Appendix D. Different Ratios of Vertex Selection

Here we investigate the effects of the different ratios of vertex selection and preservation in the proposed VIPool operation. We conduct experiments on the tasks of graph classification. In the proposed GXN model, we vary the ratios of vertex selection in two coarsened scales of graphs from 0.9 to 0.1, where the ratio of scale 1 is larger than the ratio scale 2. We employ greedy algorithm to maximize $C(\Omega)$ or top-K method to maximize $C_+(\Omega)$ for vertex selection. We note that, for any scale that preserve more than 70% vertices of the original graph, we do not use greedy algorithm on $C(\Omega)$ due to the much high time costs, where the greedy algorithm, specifically, spends more than 30 times of running time than top-K method. Additionally, we present the classification accuracy of a naive baseline; that is, we use the only the original scale for graph classification, which reflects the lowest bound of the model performance. Given the different ratios of vertex selection, we perform tasks of graph convolution on IMDB-B dataset, where the classification accuracies are presented in Table 9. We see that if we preserve more vertices in the coarsened scales of graphs, the entire model tends to obtain a higher graph classification accuracy, where we use not only greedy algorithm but also top-K method to optimize $C(\Omega)$ and $C_+(\Omega)$, respectively, for vertex selection; in contrast, the



(a) Semi-supervised vertex classification on Cora. (b) Semi-supervised vertex classification on Citeseer.

Figure 6: Comparison of semi-supervised vertex classification accuracies with a few selected and labeled data by using different vertex selection methods.

classification accuracies decrease with the removal of more and more vertices. The reason is that a much coarser graph cannot provides as much information as a coarsened graph which still has a large proportion of vertices. Another phenomenon is that, as the number of removed nodes increases, using greedy algorithm to select vertices tends to achieve increasingly higher classification accuracy than using top-K method, indicating that, greedy algorithm obtain more informative vertices when we select a few ones. Note that, when the ratio of scale 1 and ratio of scale 2 are respectively 0.8 and 0.5, we run the GXN model with the default configurations in our main text.

Appendix E. Use A Few Selected Vertices for Semi-supervised Vertex Classification Training

Here we consider active-sample-based semi-supervised classification, where we are allowed to select a few vertices and obtain their corresponding labels as supervision to train a classifier for vertex classification. In other words, we actively select training data in a semi-supervised classification task. Intuitively, since a graph structure is highly irregular, selecting a few informative vertices would potentially significantly improve the overall classification accuracy. Here we compare the proposed VIPool to random sampling. Note that for this task, we cannot compare with other graph pooling methods. The reason is that previous pooling pooling methods need a subsequent task to provide an explicit supervision; however, the vertex selection here should be blind to the final classification labels. The proposed VIPool is rooted in mutual information neural estimation and can be trained in either an unsupervised or supervised setting.

Specifically, given a graph, such as a citation network, Cora or Citeseer, we aim to show the classification accuracy as a function of the number of selected vertices. For example, there are 7 classes in Cora, we can select $7, 14, 21, 28$ and 35 vertices ($1, 2, 3, 4$ and 5 times of 7) and use their ground-truth labels as supervision for semi-supervised vertex classification. As for Citeseer, there are 6 classes and we can select $6, 12, 18, 24$ and 30 vertices. During evaluation, we test the performances on all the unselected vertices. We compare two method for vertex selection and classification: 1) the proposed **VIPool** method, where we use greedy algorithm to optimize $C(\Omega)$ for vertex selection; and 2) **Random Sampling**, where we randomly select each vertex with the same probability on the whole graph. We conduct semi-supervised vertex classification on the datasets of Cora and Citeseer. Figure 6 shows the the classification accuracies varying with the numbers of selected vertices for two vertex selection methods. We see that, when we select only a few vertices, such as fewer than 3 times of the number of vertex classes (i.e. 21 for Cora and 18 for Citeseer), the proposed VIPool method could select much more informative vertices than randomly sampling the same number of vertices, leading to over 10% higher vertex classification accuracies. If we select more vertices by using the two vertex selection methods, the classification results corresponding to the two methods become closer to each other, indicating that a large number of selected vertices tend to potentially provide sufficient information to represent the rich patterns of the graphs and we could obtain more similar classification results than only selecting a few vertices.

Appendix F. Illustration of Vertex Selection

To show the pooling effects of different pooling algorithms, we conduct a toy experiments to reconstruct three spatial mesh graphs with an encoder-decoder model. The encoder employs different pooling methods to squeeze the original graph into a few vertices (10 vertices) and the decoder attempt to reconstruct the original graphs based on the pooled vertex features and graph structures. To train the encoder-decoder model, we use an L2-norm loss to measure the distances between the vertex coordinates of reconstructed graphs and ground-truth graphs.

The three mesh graphs have vertex features as the 2D Euclidean coordinates and the specific vertex distributions are that 1) 88 vertices uniformly distribute in a circle region; 2) 503 vertices distribute in a hollow square region, where the vertices densely distribute around the center and sparsely distribute near the margins; 3) 310 vertices distribute in a circle, where the vertices densely distribute near the center and sparsely distribute around. The specific topologies are shown in the first row of Figure 7.

We compare the proposed VIPool operation with several baseline methods: random sampling, gPool [20], SAGPool [31] and AttPool [27]. The selected vertices are colored blue and illustrated in Figure 7. We see that, VIPool can abstract the original graphs more properly, where the preserved vertices distribute dispersely in both dense and sparse regions to cover the overall graphs. As for the baselines, we see that, 1) random sampling tends to select more vertices in dense regions, since each vertex is sampled with equal probability and the dense regions include more vertices and chances for vertex selection; 2) gPool and SAGpool calculate the importance weight for each vertices mainly based on vertex information itself without topological constraints, thus the selected vertices tends to distributed concentrated in local regions. 3) AttPool considers to model the local attentions and select more representative vertices, thus it can abstract graph structures to some extent, but the vertex distributions still slightly collapse the dense region.

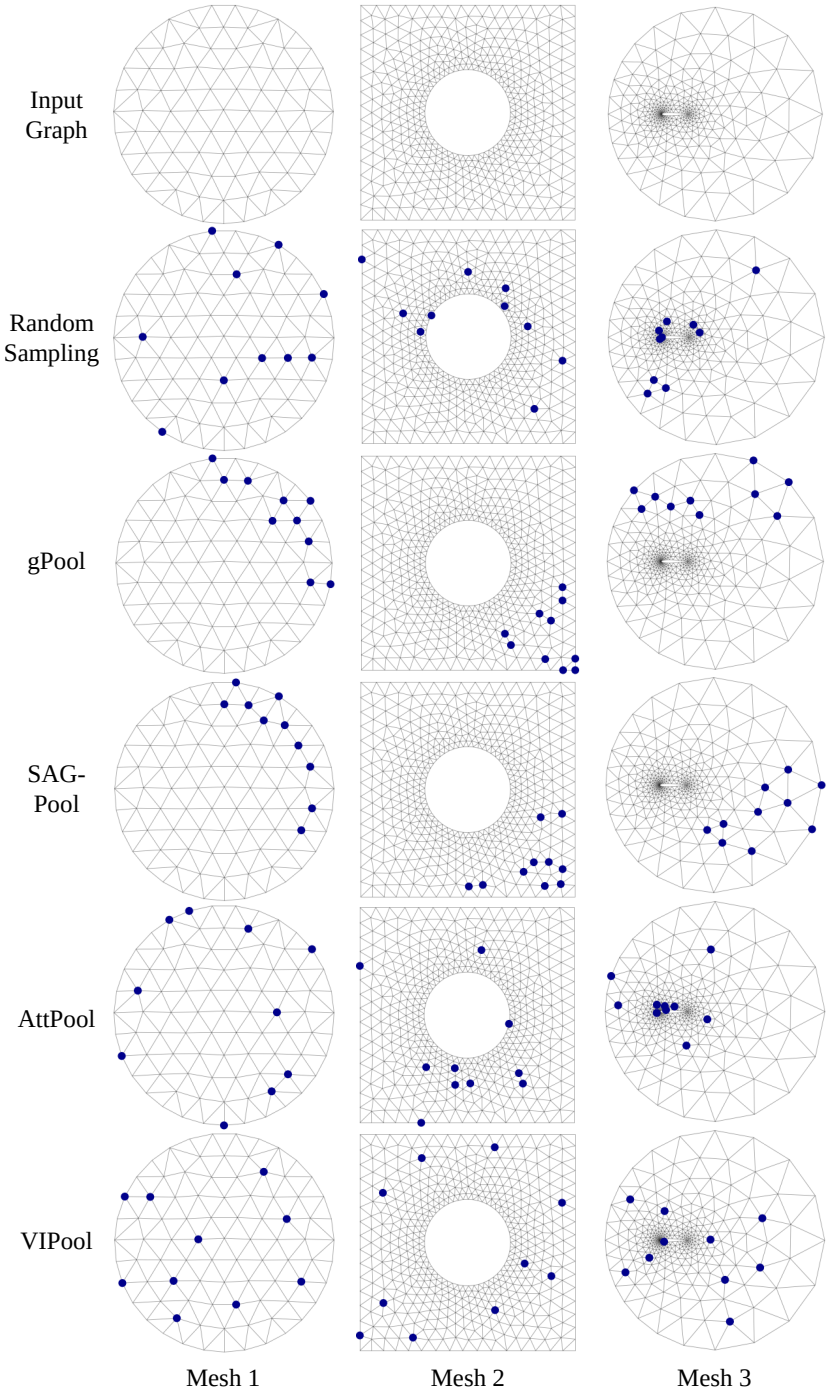


Figure 7: Vertex selection by using different pooling algorithms on three spatial mesh graph (better viewed on a color screen).