Distributed, Egocentric Representations of Graphs for Detecting Critical Structures: Supplementary Metarials

1. Further Related Work

Here, we give an in-depth review of existing graph embedding models.

Graph Kernels. The Weisfeiler-Lehman kernel (Shervashidze et al., 2011) grows the coverage of each node by collecting information from neighbors, which is conceptually similar to our method, but differs from our model in that WL kernel collects only node labels, while our method collects the complete labeled neighborhood graphs from neighbors. Deep Graph Kernels (Yanardag & Vishwanathan, 2015) and Subgraph2vec (Narayanan et al., 2016), which are inspired by word2vec (Mikolov et al., 2013), embed the graph structure by predicting neighbors' structures given a node. Multiscale Laplacian Graph Kernels (Kondor & Pan, 2016) compare graphs at multiple scales by recursively comparing graphs based on the comparison of subgraphs, which takes $O(LN^5)$ where L represents the number of comparing scales and is inefficient. All the above graph kernels have a common drawback in that the embeddings are generated in a unsupervised manner. The critical structure cannot be jointly detected at the generation of embeddings.

Graphical Models. Assuming that the edges of a graph express the conditional dependency between random variables (nodes), Structure2vec (Dai et al., 2016) introduces a novel layer that makes the optimization procedures of approximated inference directly trainable by SGD. It is efficient on large graphs with time complexity linear to number of nodes. However, it's weak on identifying critical structures because the approximated inference makes too much simplification on the graph structure. For example, the mean-field approximation assumes that variables are independent with each other. As a result, Structure2vec can only identify critical structures of very simple shape.

Convolution-based Methods. Recently, many work are proposed to embed graphs by borrowing the concept of CNN. Figure 1 summarizes the definitions of filters and neighborhoods in these work. The Spatial Graph Convolutional Network (GCN) was proposed by (Bruna et al., 2013). The design of Spatial GCN (Figure 1(a)) is very different from other convolution-based methods (and ours) since its goal is to perform hierarchical clustering of nodes. A neighborhood is defined as a cluster. However, the filter is not aim to scan for local patterns but to learn the connectivity of all clusters. This means each filter is of size $O(N^2)$ if there are N clusters. Also, a filter requires the global-scale information, i.e. the features of all clusters to train, so it's very inefficient on large graphs. Hence, in the same paper, (Bruna et al., 2013) proposed another version, the Spectrum GCN, to perform hierarchical clustering in the spectrum domain. The computation of Spectrum GCN is later improved by (Defferrard et al., 2016). However, the major drawback is that the graph spectrum is weak at identifying structures as it is only interpretable for very special graph families (e.g., complete graphs and star-like trees). A recent variant of Spectrum GCN (Kipf & Welling, 2017) uses the filters to detect the propagated feature of each node (equivalent to weighted-sum its neighbors' features). This variant also cannot detect the critical structures precisely.

Diffusion Convolutional Neural Networks (DCNN) (Atwood & Towsley, 2016) embed the graph by detecting patterns in the diffusion of each node. The neighborhood is defined as the diffusion, which are paths starting from a node to other nodes in m hops. The diffusion can be represented by an $M \times N$ diffusion matrix $\boldsymbol{D}$, where each element $D_{m,n}$ indicates if the current node connects to a node n in m hops. Filters in a DCNN scan through each node's diffusion matrix. So, DCNN can detect useful diffusion patterns. But it cannot detect critical structures since the diffusion patterns cannot precisely describe the location and the shape of structures. DCNN reported impressive results on the node classification. But it is inefficient on graph classification tasks since their notion of neighborhood is at the global-scale, which takes $O(MN^2)$ to embed a graph with N nodes. Also, their definition of neighborhood makes the embedding model shallow—there is only one single layer in a DCNN.

Patchy-San (Niepert et al., 2016) uses filters to detects patterns in the adjacency matrix of the K nearest neighbors of each node. The neighborhood of a node n is defined as the $K \times K$ adjacency matrix $\boldsymbol{A}^{(n)}$ of the K nearest neighbors of the node. Filters $\boldsymbol{W}^{(d)} \in \mathbb{R}^{K \times K}$, $d = 1, 2, \cdots, D$, scan through the adjacency matrix of each node to generate the graph embedding $\boldsymbol{H} \in \mathbb{R}^{N \times D}$, where $H_{n,d} = \sigma(\bar{\boldsymbol{A}}^{(n)} \circledast \boldsymbol{W}^{(d)} + b_d)$ is the output of an activation function σ, b_d is the bias term, and $\circledast$ is the Frobenius

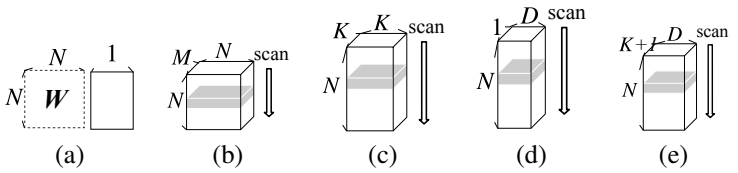

Figure 1. Filters and neighborhood in (a) Spatial GCN (Bruna et al., 2013). A filter $\boldsymbol{W}$ is a sparse matrix, which is not aimed to detect local patterns, but to learn the connectivity of clusters. (b) DCNN (Atwood & Towsley, 2016) scans through the $M \times N$ diffusion matrix of each node. (c) Patchy-San (Niepert et al., 2016) scans the $K \times K$ adjacency matrix of local neighborhood of each node. (d) Neural Fingerprints (Duvenaud et al., 2015) that scans the d-dimensional approximated neighborhood (specially, the summation of neighbors' embeddings) of each node. (e) Ego-Convolution scans $(K+1) \times D$ egocentric neighborhoods of each node.

inner product defined as $\boldsymbol{X} \circledast \boldsymbol{Y} = \sum_{i,j} X_{i,j} Y_{i,j}$. Note that a filter scans through $\bar{\boldsymbol{A}}^{(n)}$, the normalized version of $\boldsymbol{A}^{(n)}$ (Niepert et al., 2016), in order to be invariant under different vertex permutations. Patchy-San can detect precise structures (via the filters). However, to detect critical structures at the global level, each $\bar{\boldsymbol{A}}^{(n)}$ needs to have the size of $N \times N$, making the filters $\boldsymbol{W}^{(d)} \in \mathbb{R}^{N \times N}$ hard to learn. There is no discussion on how to generalize the local neighborhood defined by Patchy-San at a deep layer.

The Message-Passing NNs (Duvenaud et al., 2015; Li et al., 2016; Pham et al., 2017; Gilmer et al., 2017; Velickovic et al., 2018; Ying et al., 2018) scan through the approximated neighborhoods of each node and supports multiple layers. At each layer l, the neighborhood of a node is defined as a D-dimensional vector representing the aggregation (e.g., summation (Duvenaud et al., 2015)) of the D-dimensional hidden representation at layer $l-1$ of the l-hop neighbors. The summation avoids the vertex-ordering problem of the adjacency matrix in Patchy-San. On the other hand, the Message-Passing NNs loses the ability of detecting precise critical structures.

Note that a Message-Passing NN, called the 1-head-attention graph attention network (1-head GAT) (Velickovic et al., 2018), is a special case of Ego-CNN where the $\boldsymbol{W}^{(l,d)}$ in Eq. (3) in the main paper is replaced by a rank-1 matrix $\boldsymbol{C}^{(l,d)}$. Specifically, it models the d-th dimension of the embedding of node n at the l-th layer as $H^{(l)}_{n,d} = \sigma\left(\left[\boldsymbol{H}^{(l-1)}_{Nbr(n,1),:}, \cdots, \boldsymbol{H}^{(l-1)}_{Nbr(n,K),:}\right]^\top \circledast \boldsymbol{C}^{(l,d)}\right)$, where $\boldsymbol{C}^{(l,d)} = \boldsymbol{\alpha}\boldsymbol{W}_{d,:}$ is the outer-product of the edge importance vector $\boldsymbol{\alpha} \in \mathbb{R}^K$ and the d-th row of their weight matrix $\boldsymbol{W}$. The 1-head GAT was proposed for node classification problems. When it is applied to graph learning tasks, requiring the $\boldsymbol{C}^{(l,d)}$ to be a rank-1 matrix severely limits the model capability and leads to inferior performance.

2. Detailed Visualization Steps

In this section, we detail how the critical structure is visualized using an Ego-CNN. Like standard CNN, each neuron in Ego-CNN represents the matched result for a specific pattern (subgraph) with size upper-bounded by K^L nodes. To visualize the detected critical structure, the first step is to find the most important neighborhoods at the deepest Ego-Convolution layer. The importance score $\gamma^{(n)}$ for each node n can be calculated using the Attention layer described in Section 3.2 of the main paper. Next, we compute the importance-adjusted node embedding $\hat{\boldsymbol{H}}^{(l)}_{n,:} = \gamma^{(n)} \hat{\boldsymbol{H}}^{(l)}_{n,:} \in \mathbb{R}^D$ before applying the Transposed Convolution layer-by-layer (from the deepest to the shallowest) to plot the critical subgraphs. At each layer, we (1) follow the Transposed Convolution to re-construct each node's importance-adjusted neighboring embeddings $\hat{\boldsymbol{E}}^{(n,l)} = \sum_d \hat{\boldsymbol{H}}^{(l)}_{n,d} \boldsymbol{W}^{(l,d)} \in \mathbb{R}^{(K+1)\times D}$, and then (2) reconstruct $\hat{\boldsymbol{H}}^{(l-1)}_{n,:}$ by "undoing" $\boldsymbol{E}^{(n,l)} = \left[\boldsymbol{H}^{(l-1)}_{n,:}, \boldsymbol{H}^{(l-1)}_{Nbr(n,1),:}, \cdots, \boldsymbol{H}^{(l-1)}_{Nbr(n,K),:}\right]^\top$ in Eq. (3) in the main paper. We do so by letting $\hat{\boldsymbol{H}}^{(l-1)}_{n,:} = \sum_{i:\text{node } n \text{ is } i\text{'s } k \text{ nearest neightbor, } k \le K} \hat{\boldsymbol{E}}^{(i,l)}_{k+1,:}$. Repeating the above steps, we end up reconstructing an importance-adjusted adjacency matrix $\hat{\boldsymbol{H}}^{(0)} \in \mathbb{R}^{k \times k}$ that "undoes" Patchy-San. Our Figures 5 and 6 in the main paper plot the graphs using edge widths proportional the values in $\hat{\boldsymbol{H}}^{(0)}$.

3. More Experiments

In this section, we conduct more experiments to further investigate the performance of Ego-CNNs. First, we compare the Ego-CNN with a naive extension of Patch-San using standard CNN layers. The architecture (6 layers) and numbers of parameters of the two models are roughly the same. The results are shown in Figure 2. As we can see, the Ego-CNN with Ego-Convolutions outperforms standard CNN convolutions on graph classification problems.

Next, we compare the Ego-CNNs with and without the scale-free regularizer mentioned in Section 4.3 of the main

Model Type	MUTAG	PTC	PROTEINS	NCI1	IMDB (B)	REDDIT (B)
Ego-CNN	**93.1** (6s)	**63.8** (20s)	**73.8** (244s)	**80.7** (854s)	**72.3**	87.8
Patch-San + Std. Convolution Layers	89.4	61.5	70.7	71.0	67.1	81.0
Ego-CNN with Scale-Free Regularizer	84.5 (13s)	59.5 (22s)	73.2 (281s)	77.8 (368s)	71.5	**88.4**

Figure 2. More experiments.

paper. As we can see in Figure 2, the scale-free regularizer does not boost performance on bioinformatic datasets (MUTAG/PTC/PROTEINS/NCI1) because the graphs have no scale-free property. This motivates a test like the one shown in Figure 5 in the main paper—one should verify if the target graphs indeed have scale-free properties before applying the scale-free regularizer.

4. Relation to Kronecker Graphs

In this section, we briefly show how Kronecker graph (Leskovec et al., 2010) is a special case of Ego-CNN with scale-free regularizer. Suppose only 1 pattern $A^{(0)} \in \mathbb{R}^{K\times K}$ is captured in the node embedding layer and there is only 1 filter $W \in \mathbb{R}^{(K+1)\times 1}$ in the weight-tying Ego-Convolution layers. Then, the pattern captured in the l-th weight-tying Ego-Convolution filter is $A^{(l)} = f(A^{(l-1)} \bigotimes W)$, where f is a function that converts a list of adjacency matrices of length $K+1$ into a unified adjacency matrix in which elements that belong to the same physical nodes are merged.

References

Atwood, J. and Towsley, D. Diffusion-convolutional neural networks. In *Proceedings of NIPS*, 2016.

Bruna, J., Zaremba, W., Szlam, A., and LeCun, Y. Spectral networks and locally connected networks on graphs. In *Proceedings of ICLR*, 2013.

Dai, H., Dai, B., and Song, L. Discriminative embeddings of latent variable models for structured data. In *Proceedings of ICML*, 2016.

Defferrard, M., Bresson, X., and Vandergheynst, P. Convolutional neural networks on graphs with fast localized spectral filtering. In *Proceedings of NIPS*, 2016.

Duvenaud, D. K., Maclaurin, D., Iparraguirre, J., Bombarell, R., Hirzel, T., Aspuru-Guzik, A., and Adams, R. P. Convolutional networks on graphs for learning molecular fingerprints. In *Proceedings of NIPS*, 2015.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. In *Proceedings of ICML*, 2017.

Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. In *Proceedings of ICLR*, 2017.

Kondor, R. and Pan, H. The multiscale laplacian graph kernel. In *Proceedings of NIPS*, 2016.

Leskovec, J., Chakrabarti, D., Kleinberg, J., Faloutsos, C., and Ghahramani, Z. Kronecker graphs: An approach to modeling networks. *Journal of Machine Learning Research*, 11(Feb):985–1042, 2010.

Li, Y., Tarlow, D., Brockschmidt, M., and Zemel, R. Gated graph sequence neural networks. In *Proceedings of ICLR*, 2016.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. Efficient estimation of word representations in vector space. 2013.

Narayanan, A., Chandramohan, M., Chen, L., Liu, Y., and Saminathan, S. subgraph2vec: Learning distributed representations of rooted sub-graphs from large graphs. *In Workshop on Mining and Learning with Graphs*, 2016.

Niepert, M., Ahmed, M., and Kutzkov, K. Learning convolutional neural networks for graphs. In *Proceedings of ICML*, 2016.

Pham, T., Tran, T., Phung, D. Q., and Venkatesh, S. Column networks for collective classification. In *Proceedings of AAAI*, 2017.

Shervashidze, N., Schweitzer, P., Leeuwen, E. J. v., Mehlhorn, K., and Borgwardt, K. M. Weisfeiler-lehman graph kernels. *JMLR*, 12(Sep):2539–2561, 2011.

Velickovic, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. Graph attention networks. In *Proceedings of ICLR*, 2018.

Yanardag, P. and Vishwanathan, S. Deep graph kernels. In *Proceedings of SIGKDD*. ACM, 2015.

Ying, Z., You, J., Morris, C., Ren, X., Hamilton, W., and Leskovec, J. Hierarchical graph representation learning with differentiable pooling. In *Proceedings of NIPS*, pp. 4805–4815, 2018.