TOPOLOGY-AWARE GRAPH SIGNAL SAMPLING FOR POOLING IN GRAPH NEURAL NETWORKS

A PREPRINT

Amirhossein Nouranizadeh
Department of Computer Engineering
Amirkabir University of Technology

Mohammadjavad Matinkia
Department of Computer Engineering
Amirkabir University of Technology

Mohammad Rahmati
Department of Computer Engineering
Amirkabir University of Technology

November 11, 2020

ABSTRACT

As a generalization of convolutional neural networks to graph-structured data, graph convolutional networks learn feature embeddings based on the information of each node's local neighborhood. However, due to the inherent irregularity of such data, extracting hierarchical representations of a graph becomes a challenging task. Several pooling approaches have been introduced to address this issue. In this paper, we propose a novel topology-aware graph signal sampling method to specify the nodes that represent the communities of a graph. Our method selects the sampling set based on the local variation of the signal of each node while considering vertex-domain distances of the nodes in the sampling set. In addition to the interpretability of the sampled nodes provided by our method, the experimental results both on stochastic block models and real-world dataset benchmarks show that our method achieves competitive results compared to the state-of-the-art in the graph classification task.

***K*eywords** Graph Classification · Graph Neural Networks · Graph Poolings · Graph Signal Sampling

1 Intorduction

Considering artificial neural networks (especially CNNs) and deep learning's great success in diverse applications such as computer vision [1], speech recognition [2] and natural language processing [3, 4], where the data is regular ,i.e., the data points are represented in Euclidean space, applying these neural models on graph-structured data seems a tempting idea.

However, a vast range of data are graph-structured which encode the relationship between different data objects. Such data do not exhibit the regular characteristics of the data used in traditional neural networks[5]. As a solution, graph neural networks were first introduced in [6] and a lot of successful variants of these neural models have been proposed in recent years [7, 8, 9, 10].

Graph convolutional networks (GCN), as a generalization of CNNs to graph-structured data, extract useful feature embeddings of each node's signal or feature in an end-to-end manner by considering local graph topology of each node. In general, there are two approaches for designing graph convolutional networks: spectral methods and spatial methods [11]. In spectral methods, graph convolution operation is defined as multiplication of a parametric graph filter with spectral representation of graph signal in the Fourier domain [12]. In spatial approaches, graph convolution is defined directly in the vertex-domain as the message passing between each node's local neighborhood by means of propagating node information along graph edges. The outputs of these networks are feature embeddings for each node and contain both signal and structural information of each node and are used in a variety of node-level graph related machine learning tasks such as node classification, link prediction, community detection and etc.[13, 14, 15, 16, 17, 18, 19, 20]

However, there are problems such as graph classification, requiring to make a decision about the entire graph, and not merely any specific node(s)[21, 22, 23, 24, 25]. In these settings, a representation containing information of the whole graph need to be inferred. For this reason, there are approaches that perform summation over feature embeddings of every node in the graph as a readout phase or using different neural network architectures to get a single representation of the graph by considering nodes features. However, these approaches do not consider the hierarchical community structure of graphs and produce the graph representation in a flat manner [26].
Inspired by pooling operation in convolutional neural networks, there have been multiple attempts to extend this idea to graph convolutional networks by down-sampling the nodes of the graph in order to extract hierarchical information of graph structure [26, 27, 28, 29, 30].
In this paper we propose a novel Topology-Aware Graph Signal Sampling (TAGSS) method for pooling in graph convolutional networks which considers both the graph signal information and the topological structure of graph and nodes connectivity. In particular, the graph signal information is extracted according to the frequency content and the local variation of each node in the graph. Intuitively, we suggest that the nodes with higher local variations which show a lower resemblance with their neighbors, carry the most significant information about the graph signal, and by sampling such nodes we can preserve the more informative portion of the graph. On the other hand, by introducing a topology-preserving operator, we ensure that the sampled nodes are distributed in all the communities of the graph. Finally, since various methods propose their own architectures to evaluate the power of their pooling layers, we utilize a simple architecture to assess the effectiveness of such pooling layers in a unified setting. Furthermore, we show that our results outperform the existing methods in several datasets. In summary, our contribution in this paper is as follows:

- A novel method for graph pooling based on the local variations of the nodes signals and global topology of the graph.
- An interpretation of the sampled nodes of the graph in terms of their frequency content and location in the graph.
- A unified setting to evaluate and compare different graph pooling methods.

2 Related Works

In this section, we briefly review the related works on graph representation learning. Initially, we present a short review on graph convolutional networks, but our main focus is on pooling layers adopted in such networks. In order to be relevant to the main problem that is addressed in this paper, we avoid the details of graph neural networks.

Graph convolutional networks: Graph convolutional networks can be designed by defining the graph convolution operation in two different domains of graph information: spectral domain and spatial domain. In spectral domain, graph convolution operation is defined in the graph Fourier domain as transforming the graph signal by the matrix of the eigenvectors of graph Laplacian operator [12]. [31] proposed the spectral network in which the transformed graph signal is multiplied by a parametric filter, in the graph Fourier domain. [32] proposed ChebNet where the parametric filter was approximated by a truncated expansion of Chebyshev polynomial up to K-th order and [7] proposed a simple layer-wise propagation rule by further limiting the Chebyshev polynomial approximation of parametric filter to the order $K = 1$. Spatial approaches define the convolution operation directly in the vertex-domain by aggregating features of each node's local neighborhood with different permutation invariant functions [8, 9, 13].

Pooling methods: In tasks such as graph classification where the decision should be made about the entire graph, a single feature embedding of the graph is needed. In order to obtain the graph representation from each node's feature embedding, some approaches perform a summation over each node's feature to construct a single representation [31, 33, 34]. Several other approaches use neural networks to obtain a representation of the graph from the nodes features [13, 35]. Some introduce a virtual node to the graph which is connected to every other node and its feature representation is used as graph's representation [6]. However, these approaches do not take the hierarchical community structure of the graph topology into account. To address this issue, inspired by pooling layers in CNNs, there has been an interest in defining a pooling operation in graph domain data. Here we review the attempts of defining pooling operation in graphs from two perspectives: the approaches that exploit learnable parameters in their methods and the approaches which are static, i.e., they do not include any learnable parameter.
In earlier works, by considering graph topology, graph coarsening algorithms used spectral clustering to cluster the graph into more coarse versions of it. However due to high computational complexity of eigendecompostion, other alternatives were introduced. Graclus algorithm clusters the graph without performing eigendecomposition and some recent approaches used this algorithm to devise the pooling layer in their corresponding graph convolutional architectures [36]. Other approaches design the pooling layer of the graph convolutional architecture as a learnable module which is trained in an end-to-end manner as well as the graph convolutional layers. In fact, these approaches tune the model's parameters by using a classification loss and its gradient back-propagation into the pooling layer and

selecting the down-sampled nodes such that the classification loss decreases. Therefore, in contrast to the pooling layers in convolutional neural networks, there is no interpretable rule to specify which nodes are pooled together or have more importance in the graph. DiffPool uses learnable clustering approach to down-sample the nodes of the graph into a more coarsened one. It learns a cluster assignment matrix for softly assigning nodes to each community by using a separate graph neural network. It also uses another graph neural network to transform nodes features embeddings and finally uses the cluster assignment matrix to construct the coarsened graph's adjacency matrix and its corresponding graph signal [26]. GPool projects graph signal onto a learnable projection vector and selects the nodes that have the largest projection magnitude on this vector as the sampled nodes [30]. As the model is being trained, the projection vector is tuned such that the nodes which are more important and result in less classification error, have larger projection magnitude. It uses the indices of sampled nodes to reconstruct the adjacency matrix of coarsened graph and its residing graph signal. SAGPool follows the DiffPool's approach as it uses a graph neural network to learn an attention score for each node of the graph and it down-samples the nodes that have the highest attention scores [27]. After specifying the sampled nodes it follows the same procedure as GPool in order to reconstruct the adjacency matrix of the coarsened graph and its residing graph signal.

3 Proposed Method

In this section we present the Topology-Aware Graph Signal Sampling (TAGSS) algorithm. Initially, we introduce the concepts of graph neural networks and then propose the the TAGSS algorithm and its properties. Here we introduce the building blocks of our algorithm which makes it consider the smoothness of the nodes signals as well as the topology of the graph. Finally, we present the architecture we used for graph classification task and how to incorporate TAGSS in this architecture.

3.1 Graph Convolutional Unit

The basic graph convolutional network used in our method is the GraphSAGE layer proposed by [8]. The feature embedding $\boldsymbol{h}$ of the node v in the k-th layer is computed by

$$\boldsymbol{h}_v^{(k)} \leftarrow \sigma(\boldsymbol{W}.\text{MEAN}(\{\boldsymbol{h}_v^{(k-1)}\} \cup \{\boldsymbol{h}_u^{(k-1)}, \forall u \in \mathcal{N}(v)\})) \tag{1}$$

$$\boldsymbol{h}_v^{(k)} \leftarrow \boldsymbol{h}_v^{(k)} / \|\boldsymbol{h}_v^{(k)}\|_2, \forall v \in \mathcal{V} \tag{2}$$

where in Eq.(2), $\mathcal{V}$ denotes the set of the graph vertices, and σ denotes the ReLU activation function.

3.2 Nodes Signals Variations

Given a set of graphs $\{\mathcal{G}_i = (\mathcal{V}_i, \mathcal{E}_i, \boldsymbol{X}_i, l_i)\}_{i=1}^m$, where $\mathcal{V}_i$ is the set of the vertices of the graph i with $|\mathcal{V}_i| = n_i$, $\mathcal{E}_i$ is the set of the edges of the graph i with $\mathcal{E}_i \subset \mathcal{V}_i \times \mathcal{V}_i$, $\boldsymbol{X}_i$ is the graph signal with $\boldsymbol{X}_i \in \mathbb{R}^{n_i \times d}$, and l_i is the label assigned to $\mathcal{G}_i$, the combinatorial *Laplacian* operator of the graph i is defined as

$$\boldsymbol{L}_i = \boldsymbol{D}_i - \boldsymbol{A}_i \tag{3}$$

where $\boldsymbol{D}_i$ is the diagonal degree matrix of the graph i where $d_{jj} = \sum_{u=1}^{n_i} \boldsymbol{A}_{ju}$ denotes the degree of the node j, and $\boldsymbol{A}_i$ is the weighted adjacency matrix of the graph i. Given the Laplacian operator, the *variation* on the nodes of the graph can be defined as

$$\boldsymbol{\Delta} = \boldsymbol{L}\boldsymbol{X} \tag{4}$$

Note that by the definition of the Laplacian matrix, the i-th row of $\boldsymbol{\Delta}$ is the sum of the differences of the signal of the node i and the signals of its neighboring nodes. Using $\boldsymbol{\Delta}$ we can assign a *smoothness* score to each node by

$$\boldsymbol{z} = \phi(\boldsymbol{\Delta}) \tag{5}$$

where $\phi(.)$ is a function that calculates the norm of each row of the matrix $\boldsymbol{\Delta}$. $\boldsymbol{z}$ is a vector of size n in which the j-th element is a scalar value encoding the frequency content of the signal of the node i such that the higher values of $\mathbf{z}_i$ indicates a higher dissimilarity of the signal of the node i and the signals of its neighbors [37](Figure(1)). If we use the k-th order of the Laplacian matrix and define $\boldsymbol{\Delta}^{(k)} = \boldsymbol{L}^k\mathbf{X}$ we can define the k-th order score for each node:

$$\boldsymbol{z}^{(k)} = \phi(\boldsymbol{\Delta}^{(k)}) \tag{6}$$

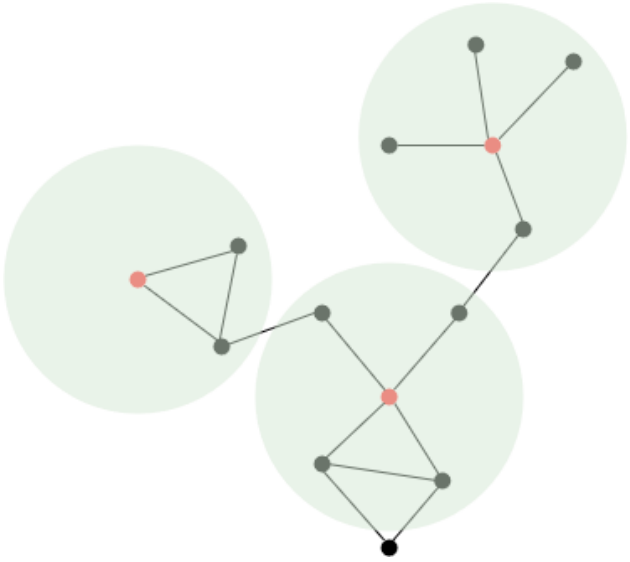

Figure 1: The connectivity of node up to one hop; the red nodes are selected such that they are not in the radius of one hop distance with each other in the graph

Again, the i-th element of the vector $\boldsymbol{z}^{(k)}$ is a scalar value encoding the frequency content of the signal of the node i up to k hops. Intuitively, the graph signal information lies on the nodes with the highest variations with respect to their neighbors, i.e., they have the lowest similarity with their neighbors. This intuition leads us to select the nodes with highest variations as the sampling nodes. However, it is highly possible that the selected nodes are placed close to each other and consequently by selecting them we lose a significant portion of the graph. In section 3.3 we propose a method to circumvent this issue and select the nodes which contain high variations and are distant from each other up to k hops as well.

3.3 Topology-Aware Operator

In the last section, we introduced the $\boldsymbol{z}^{(k)}$ vector to select the nodes with high variations. In order to force our algorithm to chose the nodes such that they are selected from various communities and hence cover a bigger portion of the graph, we introduce an operator,

$$\boldsymbol{A}_c = \mathbf{1}_{n\times n} - \boldsymbol{A} \tag{7}$$

where $\mathbf{1}_{n\times n}$ is a $n \times n$ matrix whose entries are 1. The matrix $\boldsymbol{A}_c$ represents the nodes that are not connected to each other, i.e., the (i, j)-th element of $\boldsymbol{A}_c$ is 1 if the node i and the node j are not connected, and 0 otherwise. Similarly, we define the higher order $\boldsymbol{A}_c$ as

$$\boldsymbol{A}_c^{(k)} = \mathbf{1}_{n\times n} - \psi(\sum_{1}^{k} \boldsymbol{A}^k) \tag{8}$$

where $\psi(.)$ is a function that clips the values greater than 1 to 1. The term $\psi(\sum_1^k \boldsymbol{A}^k)$ indicates a matrix whose (i, j)-th element is 1 if there is a path with the length up to k between the nodes i, and j and 0 otherwise. Similarly, $\boldsymbol{A}_c^{(k)}$ is a matrix whose (i, j)-th element is 1 if i and j are not connected to each other up to k hops. Using the operator $\boldsymbol{A}_c^{(k)}$ and the vector $\boldsymbol{z}^{(k)}$ we introduce our criteria for sampling the nodes of a given graph. Initially we define the sampling matrix $\boldsymbol{S}$ as

$$\boldsymbol{S} = diag(\boldsymbol{z}^{(k)})\boldsymbol{A}_c^{(k)} diag(\boldsymbol{z}^{(k)}) \tag{9}$$

where $diag(\boldsymbol{z}^{(k)})$ is a diagonal matrix whose (i, i)-th element is $\boldsymbol{z}_i^{(k)}$. Therefore, $\boldsymbol{S}_{i,j} = \boldsymbol{z}_i^{(k)}\boldsymbol{z}_j^{(k)}$ if the nodes i, and j are not connected up to k hops, and $\boldsymbol{S}_{i,j} = 0$ otherwise. The matrix $\boldsymbol{S}$ encodes both the topological relation of the nodes and their variation. Hence by choosing the highest values of the matrix $\boldsymbol{S}$ we can detect the nodes which have high variation and they are relatively far from each other as well. After the selection of the nodes, we aggregate the

information content of the neighbors of the selected nodes. To this end, we use the adjacency matrix as a *shift operator* and shift the signal by $\hat{\boldsymbol{A}}X$, and choose the shifted signals of the sampled nodes as the graph signal of the pooled graph. In order to do this, we use the *coarsening matrix* $\boldsymbol{C}$,

$$\boldsymbol{C}_{i,j} = \begin{cases} 1 & j = idx(i) \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

Using the coarsening matrix, the signal of the sampled nodes would be

$$\boldsymbol{X}_{sampled} = \boldsymbol{C}\hat{\boldsymbol{A}}\boldsymbol{X} \tag{11}$$

where $\hat{\boldsymbol{A}}$ is $\boldsymbol{A}+\boldsymbol{I}_n$ which contains self loops as well. The details of the algorithm are presreferencesented in **Algorithm 1**. The procedure of the algorithm gives it an adaptive characteristic. Particularly, based on the restriction imposed by the topology-aware operator $\boldsymbol{A}_c^{(k)}$, the TAGSS algorithm always samples sufficient number of nodes to satisfy its constraints, namely the required distance between the sampled nodes and their local variations.

3.4 Graph Construction

Similar to the proposed methods such as in [26], we use the coarsening matrix to construct the new adjacency matrix. In order to build the adjacency matrix of the new graph we set

$$\boldsymbol{A}_{new} = \boldsymbol{C}\tilde{\boldsymbol{A}}\boldsymbol{C}^T \tag{12}$$

where,

$$\tilde{\boldsymbol{A}} = \psi(\boldsymbol{A}^s - diag(\boldsymbol{A}^s)) \tag{13}$$

where as before, the $\psi(.)$ function clips the values greater than 1 to 1. The parameter s is a hyper-parameter and specifies the connectivity of nodes with a path of length s.

Algorithm 1 TAGSS Algorithm

input Adjacency Matrix $\boldsymbol{A} \in \mathbb{R}^{n\times n}$, Feature Matrix $\mathbf{X} \in \mathbb{R}^{n\times d}$, Sample Size s, Hop Order k
output Sampled Nodes Set ζ

Initialization Calculate the matrix $\mathbf{S}$ as in Eq.(9), an empty sampling set ζ, operator $\mathbf{A}_c^{(k)}$ as in Eq.(8).
for $i \in \{1, \dots n\}$ **do**
  $idx = \text{argmax}(\mathbf{S})$
  if $\zeta = \emptyset$ **then**
    $\zeta = \zeta \cup \{idx\}$
    $\mathbf{S}[idx] = -1$
  else
    if $0 \notin \mathbf{A}_c^{(k)}[\zeta, idx]$ **then**
      $\zeta = \zeta \cup \{idx\}$
    end if
    if $|\zeta| == s$ **then**
      break
    end if
    $\mathbf{S}[idx] = -1$
  end if
end for
return ζ

4 Experiments

In this section we evaluate the proposed TAGSS algorithm in graph classification tasks and compare the results with various pooling methods. Most of the existing methods come with their own architecture and present their results on graph or node classification tasks. One might wonder how to evaluate the effectiveness of the pooling layers. Therefore, our main goal in this set of experiments is to assess the power of pooling layers in a fixed architecture. However, similar to other methods, we propose our customized architecture as well, and evaluate the results with the existing state-of-the-art methods.
In addition, to show the interpretability of our method, we apply the TAGSS algorithm on numerous random graphs generated by the stochastic block model, and illustrate the power of TAGSS in detecting the communities of the graph.

Table 1: Statistics of the datasets

DATASET	# OF GRAPHS	# OF CLASSES	AVG. NUMBER OF NODES	AVG. NUMBER OF EDGES	NODE ATTR. DIM.
ENZYMES	600	6	32.63	62.14	18
PROTEINS	1113	2	39.06	72.82	1
MUTAG	188	2	17.93	19.79	0
FRANKENSTEIN	4337	2	16.90	17.88	780

4.1 Datasets

In these experiments we use four benchmark datasets commonly used in graph classification tasks. We use protein datasets such as **ENZYMES** and **PROTEINS**, molecular graph dataset **FRANKENSTEIN** and **MUTAG** [38, 39, 40, 41]. Table(1) shows the statistics of these datasets. In each of these datasets we use 10-fold cross-validation for model selection and model evaluation.

4.2 Experimental Settings

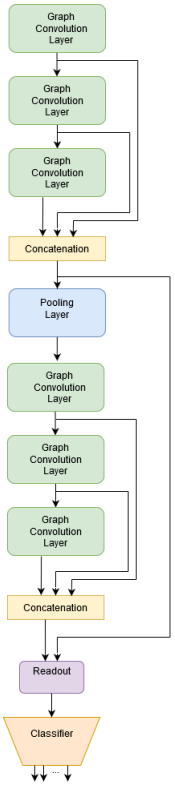

Figure 2: The hierarchical architecture used by TAGSS algorithm

Our evaluation is based on deep graph convolution networks. Initially, we propose a simple baseline without any pooling layers, and evaluate the performance of the baseline on different datasets. Second, we incorporate the DiffPool, SAGPool, iPool, Graph U Net Pooling, and TAGSS pooling layers into the baseline and compare the results. Furthermore, we compare our results with the hierarchical architecture (Figure(2)) with the results of other models based on their corresponding architectures.
The baseline model consists of three GraphSAGE layers, followed by a Readout layer which simply performs a summation, and finally a 2-layer multi-layer perceptron (MLP) with ReLU activation functions. The activation function of the GraphSAGE layers are also ReLU. We use the Adam optimizer for optimizing the models. In all the experiments we set the hop order k to 1 and the pooling ratio to 0.1. The architecture of the baseline and the baseline plus the

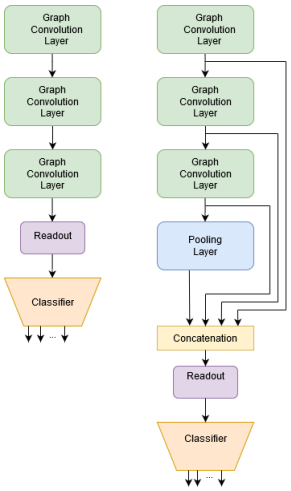

Figure 3: The baseline architecture (left), and the baseline plus the pooling layer (right)

Table 2: Comparison of performance in the fixed architecture

MODEL	ENZYMES	PROTEINS	MUTAG	FRANKENSTEIN
BASELINE	40.01	53.36	59.88	31.41
DIFFPOOL	45.01	64.28	70.50	55.63
SAGPOOL	43.23	66.78	72.08	58.33
GRAPH U NET	41.45	60.51	70.47	54.19
TAGSS	**60.13**	**70.03**	**74.84**	**64.34**

pooling layer is shown in Figure(3). For consistency, we use the GraphSAGE, DiffPool, SAGPool, and TopKPooling (Graph U Net) layers in **PyTorch Geometric** library [42, 43].

Baseline Model. For the baseline model we use three layers of GraphSAGE as shown in Figure(3). Each GraphSAGE unit embeds its input to dimension 30 for small graphs or 64 for larger graphs. After the last GraphSAGE unit we place a Readout layer which simply sums its input in each of its dimensions. Followed by the Readout layer we place an MLP whose input size is 30 or 64 (the output size of the last GraphSAGE unit). Its hidden layers contain 32 and 16 neurons respectively, and the output layer contains as many neurons as the number of classes. Finally, for the loss measure we use the *Cross Entropy* loss.

Model Comparisons. First, we compare the models based on the fixed architecture which is built upon the combination of the baseline model followed by a pooling layer. However, for each pooling method, we compare the results of the proposed architecture in corresponding papers with our specific hierarchical architecture shown in Figure(2). In the hierarchical architecture, we use three layers of GraphSAGE units followed by TAGSS pooling layer followed by three GraphSAGE layers, a Readout layer and an MLP. The output of the first three GraphSAGE layers are concatenated to represent the local information of the graph nodes. This local information is passed through the TAGSS pooling layer to summarize the graph information and structure. The output of the latter three GraphSAGE layers which operate on the pooled graph are also concatenated to form the global information of the graph. The local and global information of the graph are then aggregated in the Readout layer and form a representation for the entire graph. This representation is fed to the MLP classifier for further decision making. We show that either in the fixed architecture or in our hierarchical architecture we outperform the existing results in graph classification tasks on the aforementioned datasets. The experimental results of the fixed architecture with different pooling layers and the results of the hierarchical TAGSS-based architecture compared to proposed architectures of other pooling layers are given in Table(2) and Table(3), respectively.

Illustration on Stochastic Block Model. To show the interpretability of the nodes sampled by TAGSS, we execute the TAGSS algorithm on various random graphs generated by stochastic block model (SBM). We generate three sets of random graphs, each of which contains 10 graphs with three, four, and five communities. Intra- and inter-community

Table 3: Comparison of performance in the hierarchical architecture

MODEL	ENZYMES	PROTEINS	MUTAG	FRANKENSTEIN
DIFFPOOL	59.13	76.25	88.87	62.57
SAGPOOL	-	72.05	-	61.73
GRAPH U NET	-	77.68	-	61.46
TAGSS	**65.23**	77.45	**93.35**	**70.28**

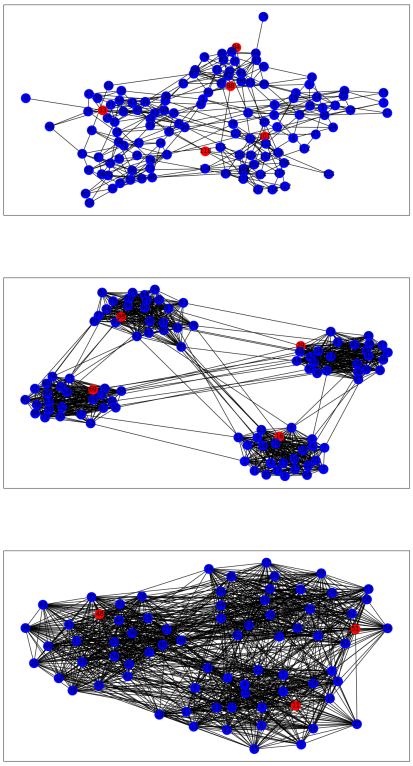

Figure 4: The result of TAGSS algorithm on a 75 node graph with three communities (down, dense-dense), on a 100 node graph with four communities (middle, dense-spare), on a 125 node graph with five communities (up, sparse-sparse)

connectivity probabilities in each set are initialized such that each set contains dense-dense, dense-sparse, and sparse-sparse connectivity distributions for intra- and inter-communitiy connectivities. Further, we use Gaussian distribution to generate random signals for the graph nodes. For simplicity, we suppose that each community of a graph has a different mean value but their variances are equal. Note that, this configuration is completely static and has no learnable parameters. Thus, we can qualitatively observe the performance of the TAGSS algorithm. Since the TAGSS algorithm samples the distant nodes with higher variations, our intuition suggests that the algorithm detects various communities and selects a node from each of them. In fact, as shown in the Figure(4), our results verify this intuition. The different parameters used by SBM are given in Table(4).

5 Conclusion

In this paper we introduced a novel method for graph signal sampling which also considers the topology of the graph, and further used this method as a graph pooling layer in graph neural networks. In addition to the interpretability of the nodes sampled by our method, we showed that it achieves the state-of-the-art performance on several graph

Table 4: Parameters used by SBM to generate random graphs in order to illustrate the stability an interpretability of TAGSS algorithm

PARAMETER	VALUE
INTRA-COMMUNITY PROBABILITY	$\{0.2, 0.5, 0.9\}$
INTER-COMMUNITY PROBABILITY	$\{0.01, 0.1, 0.2\}$
COMMUNITY MEAN	$\{0, 2, 4, 6\}$
COMMUNITY VARIANCE	1

classification benchmark datasets. Our future works include reducing the time complexity of the TAGSS method to make it more suitable for larger graphs.

References

[1] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[2] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine*, 29(6):82–97, 2012.

[3] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

[4] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[5] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

[6] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2008.

[7] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[8] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in neural information processing systems*, pages 1024–1034, 2017.

[9] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[10] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

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

[12] David I Shuman, Sunil K Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE signal processing magazine*, 30(3):83–98, 2013.

[13] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1263–1272. JMLR. org, 2017.

[14] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864, 2016.

[15] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710, 2014.

[16] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th international conference on world wide web*, pages 1067–1077, 2015.

[17] Lars Backstrom and Jure Leskovec. Supervised random walks: predicting and recommending links in social networks. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 635–644, 2011.

[18] Rianne van den Berg, Thomas N Kipf, and Max Welling. Graph convolutional matrix completion. *arXiv preprint arXiv:1706.02263*, 2017.

[19] Qing Lu and Lise Getoor. Link-based classification. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 496–503, 2003.

[20] Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE*, 104(1):11–33, 2015.

[21] S Vichy N Vishwanathan, Nicol N Schraudolph, Risi Kondor, and Karsten M Borgwardt. Graph kernels. *Journal of Machine Learning Research*, 11(Apr):1201–1242, 2010.

[22] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(Sep):2539–2561, 2011.

[23] David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems*, pages 2224–2232, 2015.

[24] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *International conference on machine learning*, pages 2014–2023, 2016.

[25] Steven Kearnes, Kevin McCloskey, Marc Berndl, Vijay Pande, and Patrick Riley. Molecular graph convolutions: moving beyond fingerprints. *Journal of computer-aided molecular design*, 30(8):595–608, 2016.

[26] 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*, pages 4800–4810, 2018.

[27] Junhyun Lee, Inyeop Lee, and Jaewoo Kang. Self-attention graph pooling. *arXiv preprint arXiv:1904.08082*, 2019.

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

[29] Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. Order matters: Sequence to sequence for sets. *arXiv preprint arXiv:1511.06391*, 2015.

[30] Hongyang Gao and Shuiwang Ji. Graph u-nets. *arXiv preprint arXiv:1905.05178*, 2019.

[31] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

[32] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*, pages 3844–3852, 2016.

[33] Ruoyu Li, Sheng Wang, Feiyun Zhu, and Junzhou Huang. Adaptive graph convolutional neural networks. In *Thirty-second AAAI conference on artificial intelligence*, 2018.

[34] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *International conference on machine learning*, pages 2014–2023, 2016.

[35] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

[36] Inderjit S Dhillon, Yuqiang Guan, and Brian Kulis. Weighted graph cuts without eigenvectors a multilevel approach. *IEEE transactions on pattern analysis and machine intelligence*, 29(11):1944–1957, 2007.

[37] Aamir Anis, Akshay Gadde, and Antonio Ortega. Efficient sampling set selection for bandlimited graph signals using graph spectral proxies. *IEEE Transactions on Signal Processing*, 64(14):3775–3789, 2016.

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

[39] Karsten M Borgwardt, Cheng Soon Ong, Stefan Schönauer, SVN Vishwanathan, Alex J Smola, and Hans-Peter Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 21(suppl_1):i47–i56, 2005.

[40] Francesco Orsini, Paolo Frasconi, and Luc De Raedt. Graph invariant kernels. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[41] Asim Kumar Debnath, Rosa L Lopez de Compadre, Gargi Debnath, Alan J Shusterman, and Corwin Hansch. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *Journal of medicinal chemistry*, 34(2):786–797, 1991.

[42] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[43] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.