NODE-SELECT: A FLEXIBLE GRAPH NEURAL NETWORK BASED ON REALISTIC PROPAGATION SCHEME

Anonymous authors
Paper under double-blind review

ABSTRACT

While there exists a wide variety of graph neural networks (GNN) for node classification, only a minority of them adopt effective mechanisms to propagate the nodes' information with respect to these nodes' global importance. Additionally, two very important challenges that still significantly affect graph neural networks are the over-fitting and over-smoothing issues. Essentially, both issues cause poor generalization of the model and much poorer node classification performance. In this paper we propose the NODE-SELECT graph neural network (NSGNN): a novel and flexible graph neural network that uses subsetting filters to learn the contribution from the nodes selected to share their information. For the selected nodes, the way their learned information propagates resembles that of actual networks of the real world; where only a subset of nodes simultaneously share information. With the ability to manipulate the message passing operations through the use of numerous ensembled filters, our NODE-SELECT graph neural network is able to address the over-fitting problem and by-pass the over-smoothing challenge for graph neural networks. Furthermore, we also propose an efficient and informative measure named MICS to quantify the over-smoothing problem. Our NODE-SELECT [1]achieved or matched state-of-the art results in a number of transductive experiments over different benchmark datasets.

1 INTRODUCTION AND RELATED WORK

The use of deep learning techniques for graph analysis has become a very popular research topic in recent years (Zhou et al., 2018). Commonly referred to as graph neural networks (GNN), these deep learning techniques now figure amongst the most used methods for learning from relational data (Zhou et al., 2018; Wu et al., 2020). Just as in the functioning of convolutional neural networks (CNN), multiple convolution operations can also be applied to learn from non-Euclidean data (Zhou et al., 2018; Wu et al., 2020; LeCun et al., 2015). Various adaptations of GNNs have been proposed over the years for the purpose of node classification (Zhou et al., 2018; Wu et al., 2020). These GNN adaptations mainly differ in regards to their node embedding techniques, their algorithms' propagation or aggregation methods, and their model's scalability (Zhou et al., 2018; Wu et al., 2020).

Examples of important GNN variants include the works proposed by Kipf et al. (GCN) (Kipf & Welling, 2016), Velickovic et. al. (GAT) (Veličković et al., 2017), Hamilton et. al (GraphSAGE), Li and Zemel et. al. (GGNN) (Cho et al., 2014), and Defferard et. al. (ChebNet) (Defferrard et al., 2016). Viewed as a pioneer of convolutional graph neural networks, GGNN (Li et al., 2015; Cho et al., 2014) uses gated recurrent units (GRU) to sequentially update the feature vectors of the graph nodes. In their work, Deferrard et al. (Defferrard et al., 2016) used Chebyshev polynomials to approximate the localized spectral filters meant to do the convolutions over the nodes and their neighborhoods. To improve the ChebNet model, GCN (Defferrard et al., 2016; Kipf & Welling, 2016) restricts the feature aggregation only to each node's nearest neighbors (1-hop neighbor) while also applying a normalization trick to manage over-fitting. GraphSAGE (Hamilton et al., 2017) applies a random sampling approach over a node's neighorbood to create the node embeddings. And

[1]The codes for this work can be found at `https://github.com/superlouis/NODE-SELECT`

last, GAT (Veličković et al., 2017) introduces the innovative concept of attention to GNN based on the idea that the contributions of a node's neighbors are unequal. In addition to the aforementioned methods, there exist many other GNN variants that have further incrementally advanced the state-of-the-art of graph neural networks. Such architectures include DropEdge (Rong et al., 2019) and DNA-Conv(Xu et al., 2018). DropEdge proposes a regularization mechanism to address over-fitting and over-smoothing by randomly removing connecting edges (Rong et al., 2019). On the other hand, inspired by the concepts of Jumping Knowledge (Xu et al., 2018), Fey proposed a dynamic neighborhood aggregation mechanism to offer to their learning model a bigger range of feature information (Fey, 2019). Nevertheless, there still remains conceptual and technical limitations that still need to be addressed to further advance the field.

In general, the usage of too many layers hinders the performance of GNNs despite having access to more learnable parameters (Zhou et al., 2018; Wu et al., 2020; Kipf & Welling, 2016). Over-smoothing is a direct consequence of deeply stacked graph convolutional (Gconv) layers (Chen et al.). Particularly, Chen et. al (Chen et al.) defined over-smoothing as the consequence of in-distinguishable representations of nodes in different classes. In other words, over-smoothing is the incidence when the model can no longer attribute the right embedding to the corresponding node so much that numerous embeddings are similar (Li et al., 2018; Zhou et al., 2018; Chen et al.). In their work, Chen et. al. (Chen et al.) justified that this over-smoothing issue occurs due to the fact that more noise gets shared than useful information during the convolution operations. Yet, another issue that is still related to the deep stacking problem is over-fitting, which defines the case when a model poorly generalizes to testing data despite fitting the training data very well (Rong et al., 2019). Besides the over-smoothing and over-fitting issues, GNNs also suffer from a noticeable conceptual limitation. Few GNN variants provide an implementation that fully emulate the relational rules observed in the networks of real world. Variant examples with mechanisms that can easily relate to real-world networks concepts include GAT, Gaan, and GATGNN which are all based on the assumption that nodes' contributions are unequal (Veličković et al., 2017; Zhang et al., 2018; Louis et al., 2020). Nonetheless, the number of GNN variants with mechanisms that so easily translate to real-world networks is minimal and remains to be further exploited (Shchur et al., 2018).

With the motivation to address the existing conceptual limitation that affects GNNs, we propose a novel architecture named NODE-SELECT graph neural network (NSGNN), also referred to as NODE-SELECT, which implements a mechanism that reflects the functioning of the brain and the principles of social networks. From a psychological or behavioral aspect, NODE-SELECT is based on the social concept that is often quoted as "too many cooks spoil the broth", meaning that too many people leading a task results in an inferior product. By the same token, we suspect that having all the nodes simultaneously propagate their information in the graph may introduce obstacles to the pattern learning. To address these potential challenges, NODE-SELECT restricts a learnable proportion of the graph nodes from propagating their features and thereby forces the learning process to be dependent only on the selected unrestricted nodes. Examples of the works where the negative impact of a higher proportion of leaders has been studied include research done by Leo et. al (Leo et al., 2019) and Rese et. al. (Rese et al., 2013). Both works support the same fact that having too many leaders in social setting induces conflicts and would therefore result in a poorly carried task (Leo et al., 2019; Rese et al., 2013). On the other hand, from the cerebral perspective, we incorporate into NODE-SELECT the ordering concept of neurons firing. Simply put, the latter describes the studied concept of neuroscience that states that neurons in the brain fire in a sequential manner and not simultaneously. An example with corresponding findings include the research done by Havenith et. al. (Havenith et al., 2011) on cortical neurons. In their work, Havenith et. al. demonstrated that, although differing by a fractional time, not all neurons fired at the exact same time (Havenith et al., 2011). Just as there is a clear ordering for the firing process in cortical neurons, we also implement in our NODE-SELECT a similar mechanism that grants a direct control over the sequential information propagation from the initially selected nodes throughout the rest of the graph. In summary, the concepts that inspired our NODE-SELECT have been well studied and backed by research in the fields of neuroscience and behavioral science. Notably, the adaptation of these two real-world concepts prove to be very useful for addressing the particular issues of over-smoothing and over-fitting in graph representation learning.

There are several key ideas in our model. First, the NODE-SELECT mechanism allows the model to prevent a subset of nodes to influence their neighbors such that a proportion of the noise information that is usually propagated will not be not introduced. This propagating constraint in our network can

be regarded as a form of regularization. Secondly, the restriction causes the embedding space to be a lot more diverse. Such diversity in the embedding can be interpreted as a form of data augmentation that is applied in our network with the generation of a varying number of correlated embeddings for each node. Lastly, the implementation of this propagating constraint allows for using very efficient graph convolutional layers. By stacking our efficient layers in parallel, our model minimizes the number of learnable parameters and thereby promotes a more effective embedding learning. Notably, both regularization, data augmentation, and parameter reduction are all well-known techniques to limit over-fitting (Srivastava et al., 2014; Domingos, 2012). Likewise, regularization has been demonstrated to be a valid technique for alleviating the over-smoothing problem (Rong et al., 2019), for which we propose a new measuring method called MICS. Altogether, NODE-SELECT addresses the over-fitting and over-smoothing issues through its message-passing regularization, node-embedding augmentation, and reduction in number of parameters.

2 PROPOSED METHOD: NODE-SELECT

In this section, we describe the technical adaptation of the sharing constraint that inspired our NSGNN. We discuss our variant by describing its framework and detail the mechanism of the individual NODE-SELECT layer.

2.1 NSGNN FRAMEWORK

We begin by formulating a graph as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ and $\mathcal{E}$ respectively define a set of nodes and a set of edges. Each node in $\mathcal{V}$, corresponds to a C-dimensional feature vector x_i. In order to do classification on $\mathcal{G}$, a given GNN needs to learn a state embedding h which includes the neighborhood information of each node. This resulting F-dimensional embedding can be expressed in terms of a function f such that: $h_i = f(x_i, \{x_j\}_{j \in \mathcal{N}_{(i)}})$, where h_i and $\mathcal{N}(i)$ denote the learned embedding and the neighborhood for a node v_i. In our algorithm, the final embedding is represented as a combination of various embeddings. NODE-SELECT adjusts the previous function f in the following fashion:

$$h_i = \sum_{l=1}^{K} h_i^{(l)} = \sum_{l=1}^{K} f^{(l)}(x_i, \{x_j\}_{j \in \mathcal{N}_{(i)}}) \tag{1}$$

, in which $l = 1, 2, ..., K$ denotes an individual NSGNN layer or filter. Namely, we designate our layers as filters because of their selection mechanism. Figure 1 below provides an illustration for our proposed framework. NSGNN takes a graph as input and then applies K filters to generate K correlated embeddings for each node. The final output is obtained by summing the K learned embeddings, thus creating a much richer state embedding. In the next sub-section, we introduce the individual NODE-SELECT filter.

2.2 NSGNN FILTER

2.2.1 PRELIMINARY UPDATE

As seen in eq. (1), each filter generates an embedding through an $f^{(\cdot)}$ function described hereafter. We first apply a linear transformation, through the use of a learnable weight matrix $\mathbf{W}_0 \in \mathbb{R}^{F \times C}$, on the feature vector x_i of each node. The shared linear transformation results in an updated feature vector y_i:

$$y_i = LeakyReLU(\mathbf{W}_0\, x_i) \tag{2}$$

, in which a negative slope of 0.01 is used for the non-linear LeakyReLU transformation. It is worth noting that this weight matrix $\mathbf{W}_0$, along with all other learnable parameters defined below, pertain to a single layer. For instance, the matrix $\mathbf{W}_0$ refers to $\mathbf{W}_0^{(l)}$.

2.2.2 SELECTION MECHANISM

In order to implement the sharing restriction, a filter requires each node's aggregated neighboring information $\sum_{j \in \mathcal{N}_{(i)}} (y_j)$. The motivation behind the usage of this aggregated information is to influence the filter to make its selection with respect to the entire graph. Namely, the learnable

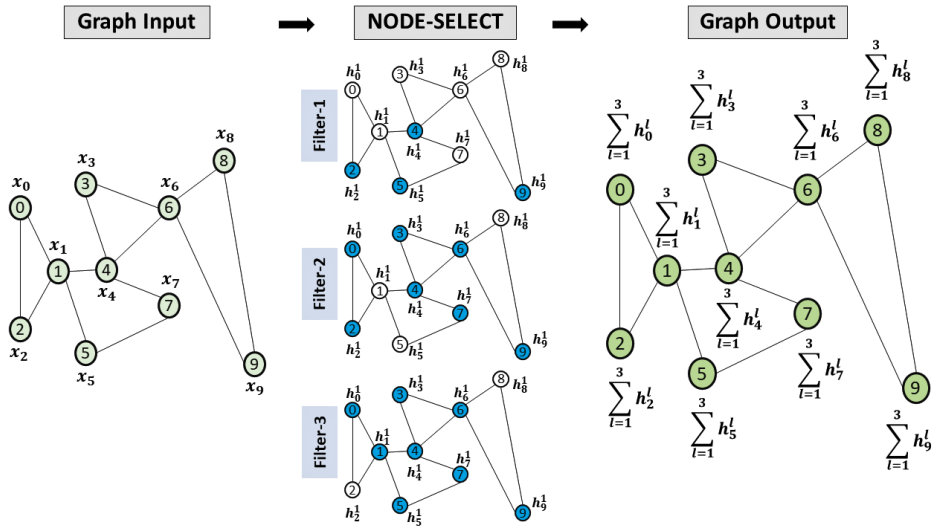


Figure 1: Architecture of the NODE-SELECT graph neural network (NSGNN). Provided a graph, a number of independent NSGNN filters (3 in this figure) are applied, where each provides a different state embedding based on their selection of propagating nodes (in blue). Finally, the output of all the filters are then summed to create a much richer state embedding for the graph nodes.

choice of whether or not a node is to propagate its information should be made on a global scale rather than a neighborhood level. Afterwards, another parametrized matrix $\mathbf{W}_1 \in \mathbb{R}^{1\times F}$ is used to create a probability p_i for each node:

$$p_i = \sigma(\mathbf{W}_1 \sum_{j\in\mathcal{N}_{(i)}} y_j) \tag{3}$$

, where σ refers to the non-linear sigmoid transformation. The probability p_i defines the filter's confidence to include a node v_i in its subset of selected nodes $\mathcal{V}^*$. Henceforth, the filter's selection mechanism $S(\cdot)$, which determines a node's selection s_i, simplifies to a piece-wise function:

$$S(v_i) = s_i = \begin{cases} 1, & p_i \geq T \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

, where a value of 1 for $S(v_i)$ means that v_i is selected and T describes the cut-off hyperparameter for the nodes probabilities. Determining the appropriate value for the selection hyperparameter T is critical. While T provides a great measure of flexibility (any number between 0 and 1), its value significantly influences the behavior of the entire NODE-SELECT network. Specifying a low value, for instance $T = 0.2$, forces the NSGNN filter to use the entire graph, while using a value larger than 0.8 reduces the size of the subset to a proportion that is so small that no nodes are selected (see Figure 11 in the APPENDIX). Corresponding to the effect of T, for very low T values, our network behaves similarly to the GAT variant which depends on the message-passing of all nodes. On the other hand, high T values make our model behave similarly to frameworks that learn from a proportion of nodes such as Node2vec (Grover & Leskovec, 2016) and GraphSAGE.

2.2.3 Message-passing & Attention

Starting with the subset of propagating nodes $\mathcal{V}^*$, a filter sequentially computes self-attention coefficients to perform the message-passing operation across Q-hop neighborhoods. The Q hyperparameter can also be interpreted as the selected nodes' depth of influence. For each $q = 0, 1, .., Q$-1 depths, a corresponding attention coefficient $\alpha_i^{(q)}$ is calculated per node v_i in the graph:

$$\alpha_i^{(q)} = \sigma(\mathbf{W}_2 \, (y_i^{(q)} \parallel q^*)) \tag{5}$$

, in which $\mathbf{W}_2 \in \mathbb{R}^{1\times(F+Q)}$ denotes a learnable matrix, $y_i^{(q)}$ the qth updated feature vector of v_i, and q^* the one-hot encoded vector of the depth q. Note that the same $\mathbf{W}_2$ matrix is shared across

all Q depths and that a depth of $q = 0$ refers to $\mathcal{V}^*$. In order to restrict the propagation of nodes unrelated to the selected subset, we further constraint $\alpha_i^{(q)}$ with:

$$\alpha_i^{(q)} = \begin{cases} \alpha_i^{(q)}, & v_i \in q\text{-hop } \mathcal{V}^* \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

, such that q-hop $\mathcal{V}^*$ refers to the q-hop neighborhood of the initially selected nodes. With this regulation, a node v_i's contribution is cancelled unless either v_i belongs to the subset $\mathcal{V}^*$ or v_i is a q-hop neighbor of an initially selected node (at depth $q \geq 1$). Figure 2 provides an illustration of the message-passing mechanism execution in the filter. Upon learning this selection-depth adapted coefficient, a node's feature updates as:

$$y_i^{(q+1)} = y_i^{(q)} + \sum_{j \in \mathcal{N}_{(i)}} (\alpha_j^{(q)} \cdot y_j^{(q)}) \tag{7}$$

. Lastly, we implement a noise filtering mechanism so as to regulate the amount of noise information that is being learned though the message-passing operations. Our motivation for the latter mechanism comes from the assumption that there exists a minority of nodes that do not need to aggregate their neighbors' information Chen et al.. We adapt our updating operation so that the filter tries to maintain an appropriate balance between learned neighboring information and each node's own feature. Therefore, after Q updates, the filter then calculates a final self-attention attention coefficient c_i though the use of a matrix $\mathbf{W}_3 \in \mathbb{R}^{1\times(2F)}$,

$$c_i = \sigma(\mathbf{W}_3 \, (y_i^{(Q-1)} \parallel y_i^{(q=0)}) \tag{8}$$

, where $Q - 1$ denotes the last depth. The c_i coefficient benefits in adjusting the learning of a given a node such that the layer may filter potential noise acquired during the aggregation process (see Figure 5 of APPENDIX). Hence, the final embedding output by a NSGNN layer l can be formulated as:

$$h_i^{(l)} = (1 - c_i) \cdot y_i^{(q=0)} + c_i \cdot y_i^{(Q-1)} \tag{9}$$

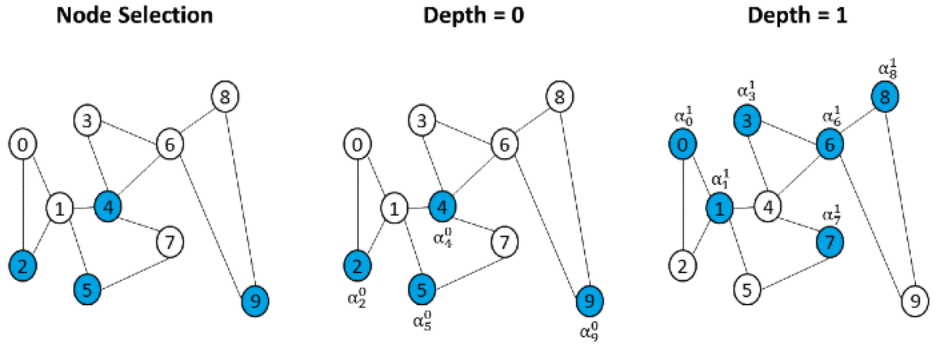


Figure 2: Message Propagation in filter. First, the NSGNN filter selects a subset of propagating nodes based on their global importance. Those selected vertices are represented in blue: $\mathcal{V}^* = \{2, 4, 5, 9\}$. At depth $q = 0$, only the selected nodes $\{2, 4, 5, 9\}$ are allowed to share a proportion $\alpha_{(\cdot)}^{(0)}$ of their embedding. At depth $q = 1$, only the 1-hop neighbors of the initially selected nodes are allowed to share another proportion $\alpha_{(\cdot)}^{(1)}$ of their updated contribute to the message-passing.

2.3 Complexity Analysis

Our proposed framework of NODE-SELECT demonstrates very good efficiency. The time complexity of our variant, which depends on both hyperparameters of Q and K, can be expressed as $\mathcal{O}(QK|\mathcal{E}|)$. In our experiments, we found that Q and K are never simultaneously large. For smaller

graphs, the maximum effective size for K was 3 with Q set at 2. On the other hand, larger graphs needed at most a K of 25 with a smaller Q fixed at 1. Given that the maximum value of the product QK does not exceed 25 and that Q is generally small ($Q = 1$), we can then further estimate the computation complexity to mostly scale in terms of K, hence: $\mathcal{O}(K|\mathcal{E}|)$.

3 EXPERIMENTS

3.1 DATASETS

To assess the performance of our proposed model, we evaluate NODE-SELECT on 8 transductive benchmark datasets: Cora, CiteSeer, PubMed, Cora Full, Coauthor CS, Coauthor Physics, and Amazon Photo. The Cora, CiteSeer, PubMed, and Cora Full datasets contain relational data pertaining to the classification of academic papers Sen et al. (2008), (Bojchevski & Günnemann, 2017). The Coauthor CS (Co-CS) and Coauthor Physics (Co-P) define graphs on co-authorships data on the Microsoft Academic Graph (Shchur et al., 2018). Lastly, Amazon Computers (Amz-C) and Amazon Photo (Amz-P) define segments of the Amazon product categories graphs. For each dataset, we randomly split the graph so that the training, validation, and testing follow a ratio of 20-20-60 percent. This split is repeated on 10 random seeds. Note that those same random seeds are kept and then re-used for each model experiment. Details of the Datasets are provided in Table 3 of the APPENDIX .

3.2 EXPERIMENTAL SETUP

We compare NODE-SELECT to 5 baseline models: GAT, GCN , GraphSAGE or SAGE, Node2vec (Grover & Leskovec, 2016), and a Multilayer-perceptron (MLP) (Nielsen, 2015). Besides the popular variant of GCN, our proposed NODE-SELECT shares at least one similarity with the other baseline models. NODE-SELECT applies self-attention mechanisms just as in GAT, uses subset of nodes and mechanism of neighborhood depth as in GraphSAGE and Node2vec, and the preliminary update in NSGNN contrasts to the linear-transformation done in MLP. For the Cora and Citeseer datasets, we applied a depth of $Q = 2$ and a maximum of 3 filters. On the other hand, for the remaining larger datasets, we evaluated NODE-SELECT with a depth of $Q = 1$ with a maximum of 25 filters. For the baseline models, we perform random hyperparameter (reported in Table 4 in APPENDIX) search for each one because different variants may require distinct training settings on different benchmarks. The same optimizer of Adam (Kingma & Ba, 2014) is used to train all the networks. We implement all the models using Pytorch and the library of Pytorch-Geometric (Paszke et al., 2017), (Fey & Lenssen, 2019).

3.3 RESULTS

Table 1: Average testing accuracy (%) and standard deviation from 10 random splits comparing our NODE-SELECT approach to different baseline variants.

Properties	CiteSeer	Cora	PubMed	Co-P	Co-CS	Cora Full	Amz-C	Amz-P
GAT	**74.2±0.8**	**86.0±0.7**	86.4 ±0.3	95.7±0.1	92.2 ±0.2	64.8 ±0.5	90.0±0.7	93.7±0.6
GCN	74.0 ±0.7	85.0±0.7	87.2±0.3	95.9±0.1	93.1±0.2	**67.3 ±0.5**	89.4±0.5	93.5±0.2
SAGE	73.7 ±0.7	**86.0±0.7**	86.2 ±0.3	95.4 ±0.2	93.4 ±0.2	64.9 ±0.3	**90.2 ±0.5**	**94.4 ±0.5**
MLP	68.5±0.8	69.3±1.2	86.0 ±0.3	94.9 ±0.1	92.7 ±0.1	54.5 ±0.5	83.1±0.2	90.2±0.4
Node2vec	55.3±0.7	78.1±0.8	80.2 ±0.4	93.0±0.1	87.7±0.3	58.8±0.3	87.2±0.4	91.0±0.3
NSGNN	73.9±1.0	**86.0±0.7**	**88.1±0.3**	**96.5±0.1**	**94.8±0.1**	**67.3 ±0.6**	89.6±0.4	**94.4±0.4**

Table 1 displays the average accuracy results over 10 random splits. Our proposed NODE-SELECT method consistently matches or outperforms the performance by well-known and very efficient GNN architectures. On the datasets of Cora, Amazon Products, and Cora-Full, our NODE-SELECT performed as well as the best tuned baseline variants; reaching the same accuracy as best tuned GAT, GCN, and GraphSAGE models. While GAT and GraphSAGE achieved slightly better accuracies on the CiteSeer and Amazon-Computers datasets, our NODE-SELECT outperformed all baseline on the remaining datasets. On the Pubmed dataset, NODE-SELECT is better than the other models by

at least 0.9%, on the Coauthor-Physics by at least 0.6%, and on the Coauthor CS by 1.4%. Out of the 8 tested benchmark datasets, NODE-SELECT achieved new state-of-the-art(SOTA) results on 3 and matched the best performance on 3 other datasets. Simply put, the high accuracy achieved by our proposed framework is due to our calculated richer embedding obtained by combining the learned embeddings from the NODE-SELECT filters.

4 ANALYSIS OF NODE-SELECT

4.1 NSGNN FILTERS

To better understand the functioning of NODE-SELECT, we provide a detailed analysis of our proposed framework with respect to each one of its filters. Compared to the traditional sequential stacking of layers, NODE-SELECT adopts the well-known ensemble method in machine learning (Dietterich, 2000), for which each filter independently learns the nodes embedding based on a primary selection of propagating nodes. In our study, we observed that the average cosine-similarity of these embeddings per node is always high (≥ 0.85). Henceforth, by combining these sets of independent yet correlated embeddings, the model achieves subsequently learns a final embedding that is much richer or precise. Figure 3 provides an illustration of the correlation of the embeddings for the first node from the Coauthor-Physics graph dataset. Noticeably, the majority of the embeddings are very similar. Particularly the embeddings from filters 1, 3, 6, 7, 8, and 10 appear the most alike and together they contribute the most significant information towards the calculation of the final NSGNN embedding. Based on our experiments, we deduce that not all of the filters' embeddings need to match for a good node prediction. As long as there is a general harmony in a sufficient number of filters, the model will depend on these more frequent harmonious embeddings to form its final embedding. Hence, we conclude that the final embedding output by NSGNN comprises information of embeddings but also the confidence level of the prediction.

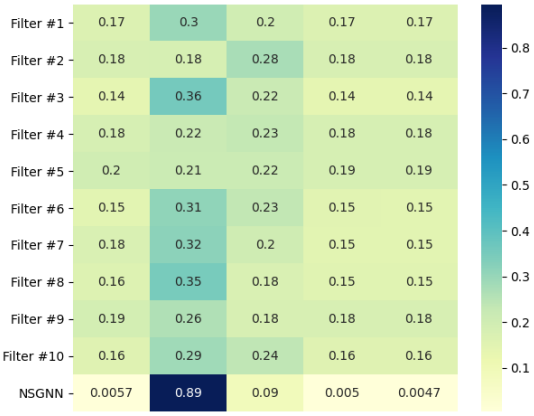


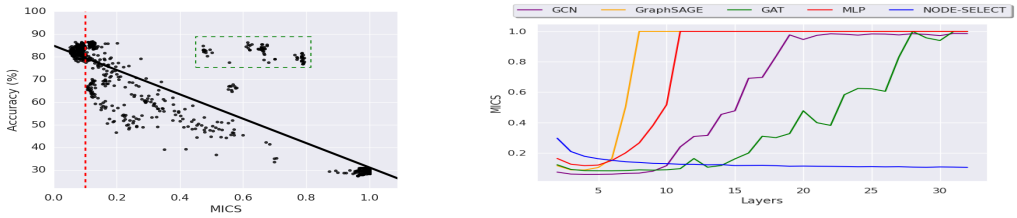
Figure 3: Heat-map of softmax-transformed embeddings for a node in Co-P dataset.

Even though the filters learn the embeddings independently, they tend to capture the influence of the selected nodes in a similar fashion. We confirmed that the filters learned the patterns in a comparable manner by inspecting the filters' self-attention coefficients. Figures 6a and 6b from the APPENDIX provide illustrations of the self-attention weights $\alpha_i^{(q)}$ learned by a trained NODE-SELECT model on the Cora Dataset. A visual inspection validates of those figures can validate the previous claim so much that the coefficients distributions of those filters are similar. The overall higher performance achieved by NSGNN is due to its filters capturing similar message-passing information while selecting different subset $\mathcal{V}^*$. Thus, these filters jointly cover more of the embedding space. Table 2 below lists the accuracy and ratio size $\mathcal{V}^*$ of a trained NSGNN model and its 3 filters. Based on the accuracy score of the filters, we deduce that the performance of a filter depends more on the quality of the selection (which nodes were selected) and less on the size of the subset.

Table 2: Accuracy score and ratio by Filter and NSGNN for model trained on Cora Dataset.

	Accuracy	Size of $\mathcal{V}^*$
Filter #1	85.5	0.69
Filter #2	86.3	0.88
Filter #3	85.7	0.92
NSGNN	**87.3**	—

4.2 OVER-FITTING & OVER-SMOOTHING



(a) Over-smoothing (MICS) x Accuracy

(b) Number of layers x over-smoothing.

Figure 4: (a) Impact of over-smoothing over models' accuracy. (b) Effect of additional layers on over-smoothing.

Mechanisms indicative of regularization generally help against over-fitting. NODE-SELECT demonstrated better resilience to over-fitting than most baseline GNN thanks to its restriction of propagating nodes and its generation of various embeddings (identifiable as data augmentation). In our experiments, GCN, GraphSAGE, MLP, and Node2vec were all a lot more prone to over-fitting (see Figure 13 in APPENDIX contrasting over-fitting in NODE-SELECT agaisnt in GCN on the Cora dataset). On the other hand, over-smoothing does not affect our NODE-SELECT. We assessed the latter claim through the quantitative results of our new proposed measure: MICS, which stands for mean inter-class smoothing and is applicable to node classification problems. Compared to the works by Chen et. al. and Zhao et. al., our MICS is much more simple, efficient, and informative (Chen et al.),(Zhao & Akoglu, 2019). Basically, the calculation resolves to measuring the similarities of embeddings between nodes of different classes (inter-class smoothing). The inter-class smoothing (ICS) for two classes A and B can be calculated as: $ICS_{A,B} = \frac{1}{|A|\times|B|}\sum_{i\in A}\sum_{j\in B}\frac{h_i^*\cdot h_j^*}{|h_i^*|\times|h_j^*|}$; where h_i^* and h_j^* represent softmax-transformed embeddings of nodes belonging to class A and class B. The ICS represents a very informative metric since it allows one to specifically assess the learning of each group of nodes based on how a trained model confuses that group's embedding with other classes of nodes. The final MICS is calculated as: $MICS = \frac{1}{|D|}\sum_{d\in D}(ICS)_d$; where D is the set of combinations of different classes. MICS, the average similarity score between different classes of node, summarizes in one interpretable score the severity of over-smoothing. Figure 4a illustrates the negative correlation between over-smoothing (MICS) and a model's accuracy of multiple models trained on the Cora dataset. Supporting previous findings on over-smoothing, higher MICS values typically correlate with lower accuracy performance (Chen et al.) (Zhao & Akoglu, 2019). However, MICS values may also correspond to high accuracy. These special occurrences of high MICS and high accuracy, in the green rectangle, represent cases of under-fitting, i.e. a trained model with too few learnable parameters (1 layer). On the other hand, figure 4b illustrate the effect of over-smoothing from the addition of layers on various models. Thanks to its parallel stacking, our NODE-SELECT is extremely resistant to over-smoothing. As opposed to the other frameworks which sequentially convolve the embeddings, thus removing key information in the embeddings, our NODE-SELECT relies on various complementary convolutions to enhance those key information. Therefore, the introduction of additional learnable parameters will only impact our NSGNN in over-fitting and not in over-smoothing.

5 CONCLUSION

We introduced NSGNN, a novel graph neural network for node-classification, which learns node embeddings by summing correlated embeddings learned by its filters. Inspired by the functioning of real-world graphs, our NODE-SELECT addresses the conceptual limitation of selective propagation based on the nodes global importance. Our framework also addresses the technical challenges of over-fitting by applying various regularization mechanisms. Besides the SOTA performance, our framework also has complete immunity to over-smoothing, quantifiable by our proposed MICS measure, thanks to the parallel stacking of its layers. We expect that more in-depth exploration of our proposed method will be done to further the advancement of the GNN field.

REFERENCES

Aleksandar Bojchevski and Stephan Günnemann. Deep gaussian embedding of graphs: Unsupervised inductive learning via ranking. *arXiv preprint arXiv:1707.03815*, 2017.

Deli Chen, Yankai Lin, Wei Li, Peng Li, Jie Zhou, and Xu Sun. Measuring and relieving the over-smoothing problem for graph neural networks from the topological view.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

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

Thomas G Dietterich. Ensemble methods in machine learning. In *International workshop on multiple classifier systems*, pp. 1–15. Springer, 2000.

Pedro Domingos. A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78–87, 2012.

Matthias Fey. Just jump: Dynamic neighborhood aggregation in graph neural networks. *arXiv preprint arXiv:1904.04849*, 2019.

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

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*, pp. 855–864, 2016.

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

Martha N Havenith, Shan Yu, Julia Biederlack, Nan-Hui Chen, Wolf Singer, and Danko Nikolić. Synchrony makes neurons fire in sequence, and stimulus properties determine who is ahead. *Journal of neuroscience*, 31(23):8570–8584, 2011.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

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

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

Francisco M Leo, Tomás García-Calvo, Inmaculada González-Ponce, Juan J Pulido, and Katrien Fransen. How many leaders does it take to lead a sports team? the relationship between the number of leaders and the effectiveness of professional sports teams. *PloS one*, 14(6):e0218167, 2019.

Qimai Li, Zhichao Han, and Xiao-Ming Wu. Deeper insights into graph convolutional networks for semi-supervised learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

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

Steph-Yves Louis, Yong Zhao, Alireza Nasiri, Xiran Wong, Yuqi Song, Fei Liu, and Jianjun Hu. Global attention based graph convolutional neural networks for improved materials property prediction. *arXiv preprint arXiv:2003.13379*, 2020.

Michael A Nielsen. *Neural networks and deep learning*, volume 2018. Determination press San Francisco, CA, 2015.

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.

Alexandra Rese, Hans-Georg Gemünden, and Daniel Baier. 'too many cooks spoil the broth': Key persons and their roles in inter-organizational innovations. *Creativity and Innovation Management*, 22(4):390–407, 2013.

Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. Dropedge: Towards deep graph convolutional networks on node classification. In *International Conference on Learning Representations*, 2019.

Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

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

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. Representation learning on graphs with jumping knowledge networks. *arXiv preprint arXiv:1806.03536*, 2018.

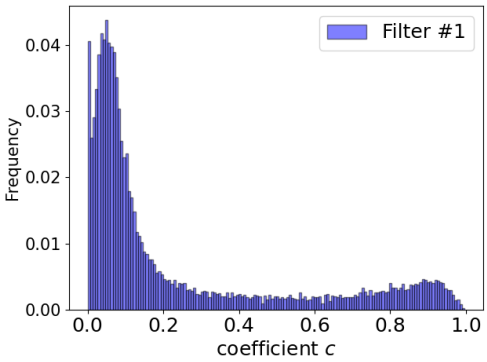
Jiani Zhang, Xingjian Shi, Junyuan Xie, Hao Ma, Irwin King, and Dit-Yan Yeung. Gaan: Gated attention networks for learning on large and spatiotemporal graphs. *arXiv preprint arXiv:1803.07294*, 2018.

Lingxiao Zhao and Leman Akoglu. Pairnorm: Tackling oversmoothing in gnns. In *International Conference on Learning Representations*, 2019.

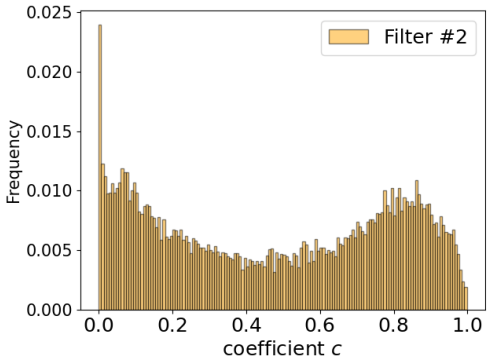
Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434*, 2018.

A APPENDIX

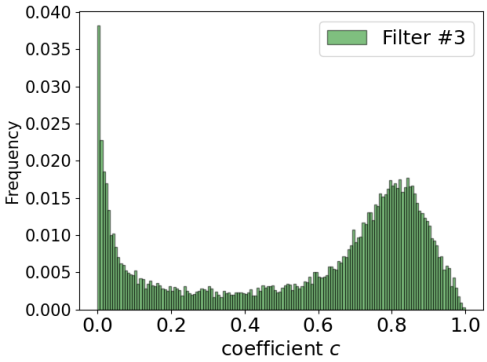
A.1 REGULARIZATION COEFFICIENT C



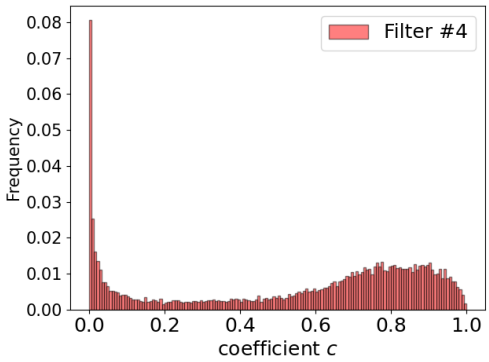
(a) Filter #1 with accuracy 83.0%



(b) Filter #2 with accuracy 84.8%



(c) Filter #3 with accuracy 87.2%



(d) Filter #4 with accuracy 87.1%

Figure 5: Comparisons of distribution of C learned by each filter in a NODE-SELECT model trained on the Pubmed dataset. The overall accuracy reached by the model is 89.2%.

A.2 DATASETS

Table 3: Statistics of transductive Datasets used in this paper.

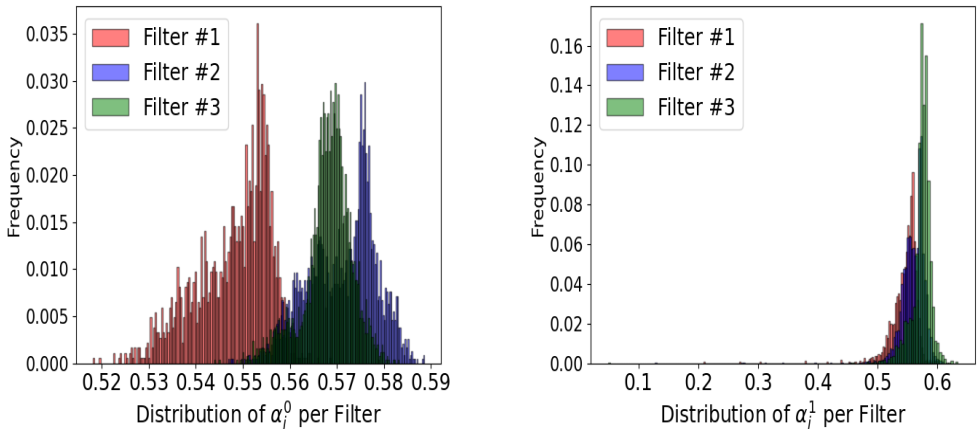
Dataset	Nodes	Edges	Classes	Features
CiteSeer	3,327	4,552	6	3,703
Cora	2,708	5,278	7	1,433
PubMed	19,117	44,324	3	500
Coauthor P (Co-P)	34,493	247,962	5	8,415
Coauthor CS (Co-CS)	18,333	81,894	15	6,805
Cora Full	19,793	63,421	70	8,710
Amazon Photo (Amz-P)	7,650	245,861	8	8,415
Amazon Computers (Amz-C)	13,752	119,081	10	767

A.3 HYPERPARAMETER CONFIGURATIONS

Table 4: The hyperparameters providing best accuracy for each baseline model on all datasets. These parameters are listed as (# of layers / # of neurons used in hidden layers / learning-rate / optimizer's weight-decay/*additional-details). The same parameters were used in Node2vec whose additional details include (walk-length:20, context-size:10,walk-per-node:1,negative-sample=5)

Framework	Dataset	Acc.	Configuration
GAT	CiteSeer	74.0±0.7	2 / 64 / 0.0005 / 0.005 / attention-heads:8
	Cora	86.0±0.7	2 / 128 / 0.0005 / 0.005 / attention-heads:8
	PubMed	86.4 ±0.3	3 / 64 / 0.01 / 0.00005 / attention-heads:8
	Co-P	95.7 ±0.1	3 / 64 / 0.01 / 0.00005 / attention-heads:8
	Co-CS	92.2 ±0.2	3 / 64 / 0.01 / 0.00005 / attention-heads:8
	Cora Fullll	64.8 ±0.5	2 / 128 / 0.005 / 0.00005 / attention-heads:8
	Amz-P	93.7±0.6	2 / 128 / 0.005 / 0.00005 / attention-heads:8
	Amz-C	90.0±0.7	2 / 128 / 0.005 / 0.00005 / attention-heads:8
GCN	CiteSeer	74.0±0.6	2 / 128 / 0.0005 / 0.05 / —
	Cora	85.0±0.7	2/ 128 / 0.01 / 0.0005 / —
	PubMed	87.2 ± 0.2	2 / 128 / 0.01 / 0.0005 / —
	Co-P	95.9 ±0.1	2 / 64 / 0.01 / 0.0005 / —
	Co-CS	93.1±0.2	2 / 128 / 0.01 / 0.0005 / —
	Cora Full	67.3 ±0.5	2 / 128 / 0.01 / 0.0005 / —
	Amz-P	93.5±0.2	2 / 128 / 0.01 / 0.0005 / —
	Amz-C	89.4±0.5	2 / 128 / 0.01 / 0.0005 / —
GraphSAGE	CiteSeer	73.7 ±0.7	2 / 64 / 0.0005 / 0.005 / —
	Cora	86.0±0.7	2 / 64 / 0.0005 / 0.005 / —
	PubMed	86.2 ±0.3	2 / 64 / 0.05 / 0.0005 / —
	Co-P	95.4 ±0.2	2 / 64 / 0.005 / 0.0005 / —
	Co-CS	93.4±0.2	2 / 64 / 0.001 / 0.0005 / —
	Cora Full	64.9 ±0.3	3 / 128 / 0.005 / 0.0005 / —
	Amz-P	94.4 ±0.5	2 / 64 / 0.005 / 0.0005 / —
	Amz-C	90.2 ±0.5	3 / 128 / 0.005 / 0.0005 / —
MLP	CiteSeer	68.5±0.8	3 / 128 / 0.005 / 0.05 / —
	Cora	69.3±1.2	2 / 64 / 0.005 / 0.0005 / —
	PubMed	86.0 ±0.3	3 / 128 / 0.005 / 0.0005 / —
	Co-P	94.9 ±0.1	3 / 64 / 0.005 / 0.0005 / —
	Co-CS	92.7 ±0.1	3 / 128 / 0.005 / 0.0005 / —
	Cora Full	54.5 ±0.5	3 / 128 / 0.005 / 0.0005 / —
	Amz-C	83.1 ±0.2	3 / 128 / 0.005 / 0.0005 / —
	Amz-P	90.2±0.4	3 / 128 / 0.005 / 0.0005 / —
Node2vec	*	*	1 / 64 / 0.005 / — /—
NODE-SELECT	CiteSeer	73.9±1.0	3 / — / 0.01 / 0.05 / depth:2
	Cora	85.8±0.6	3 / — / 0.005 / 0.05 / depth:2
	PubMed	88.1 ±0.3	8 / — / 0.05 / 0.00005 / depth:1
	Co-P	96.5±0.1	10 / — / 0.005 / 0.00005 / depth:1
	Co-CS	94.8±0.1	8 / — / 0.005 / 0.00005 / depth:1
	Cora Full	67.3 ±0.6	8 / — / 0.005 / 0.00005 / depth:1
	Amz-C	89.6±0.4	25 / — / 0.001 / 0.00005 / depth:1
	Amz-P	94.4±0.3	25 / — / 0.001 / 0.05 / depth:1

A.4 WEIGHTS ANALYSIS OF FILTERS



(a) Self-Attention weights per node at depth $q = 0$ (b) Self-Attention weights per node at depth $q = 1$

Figure 6: Distribution of self-attention weights from a trained NSGNN model on the Cora dataset (random-split).

A.5 EFFICIENCY ANALYSIS OF MICS METRIC

Table 5: Number of cosine-similarity involved between other over-smoothing methods and our MICS method.

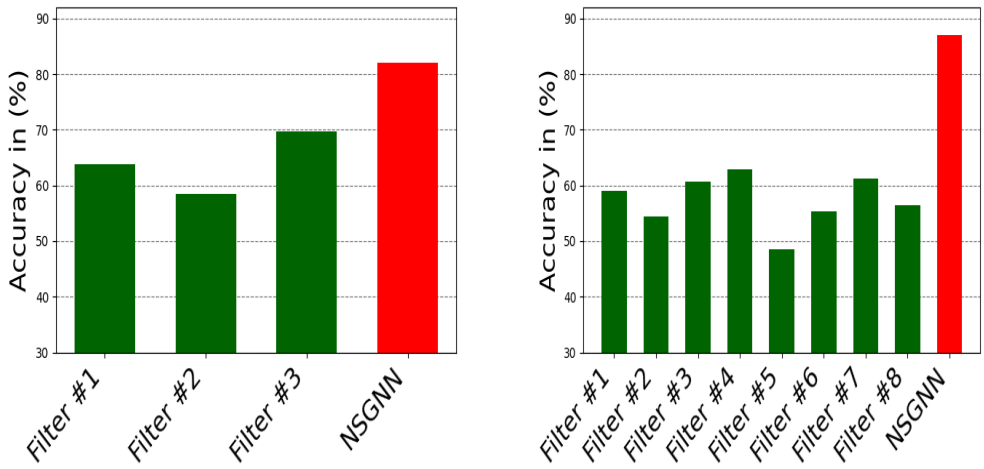
Dataset	**Other proposed Methods**	**MICS**	**% change**
CiteSeer	11,068,929	4,546,229	-59%
Cora	7,333,264	3,008,223	-59%
PubMed	388,760,089	125,008,867	-68%
Coauthor P (Co-P)	1,189,767,049	403,812,339	-66%
Coauthor CS (Co-CS)	336,098,889	149,185,346	-56%
Cora Full	391,762,849	191,631,114	-51%
Amazon Photo (Amz-P)	58,522,500	24,440,007	-58%
Amazon Computers (Amz-C)	189,117,504	74,853,473	-60%

As opposed to the over-smoothing methods proposed in the works of Chen et al. and Zhao & Akoglu (2019) which require the calculations of N^2 similarity calculations, MICS reduces by more than half the number of calculations. As seen in Table 5, MICS is a lot more efficient than the other proposed methods. Contrasting the number of cosine-similarity involved, MICS decreases by at least 51% the required number of computations needed to obtain an over-smoothing score.

A.6 EFFECT OF INCREASING NUMBER OF FILTERS

While NODE-SELECT is robust and performs well under most conditions, like GAT, it is relatively sensitive to its parameters tuning. First of all, there exists an optimal number for the number of filters that comprise a NSGNN model. Exceeding this optimal number of filters increases the effect of over-fitting to the NODE-SELECT model and thus reduces the generalization of the model. Figure 9 of the appendix display the impact of surpassing the optimal number of filters (3) in the Cora dataset. At the optimal number of filters, the model's performance (average accuracy for 10 random splits) is at its peak. However, the increase in the number of filters prior to reaching that optimal number increases the performance of the NSGNN model. As more filters are added, NODE-SELECT captures the relationship patterns more efficiently and thus its predictive performance improves. The latter increase in performance is due to the specialization of the filters, which together complement each

the other's inaccuracies. Figures 7 and 8 illustrate the effects of the addition of layers (in green) on the overall model (in red) on the Amazon-Computers dataset. In Figure 7a, the model's accuracy reaches an accuracy score of about 82% while best filter obtains accuracy at exactly 70%. As 5 more filters are added in the illustration of figure 7b, the highest accuracy reached by any filter drops 62% while the NSGNN accuracy improves to 87%. Last in figure 8, 8 more filters are afterwards added which again cause the best accuracy of the filters to drop to 60% while the overall accuracy of the model further improves to 89.4%. This reduction in performance by individual filters leading to the improved performance of NODE-SELECT can be explained by the specialization of the filters. Notably, the inclusion of additional layers influences the filters to become less generalizable and to mainly focus on learning from a particular subset of nodes. Lastly, the threshold parameter (T) is related to both the overall performance of the model and sizes of the subset of selected nodes $\mathcal{V}^*$. As seen in Figure 10, our experiments consistently reached the best performance when T ranges between 0.38 and 0.48. For very low T values, our model behaves similarly to GAT and persistently performs well. However, for higher T values $T \geq 0.5$, our model's performance quickly drops. Ultimately, training a NSGNN model using sub-optimal parameters will likely also output sub-optimal results.



(a) 3 NSGNN filters

(b) 8 NSGNN filters

Figure 7: 3 layers in (a) and (8) layers in (b)

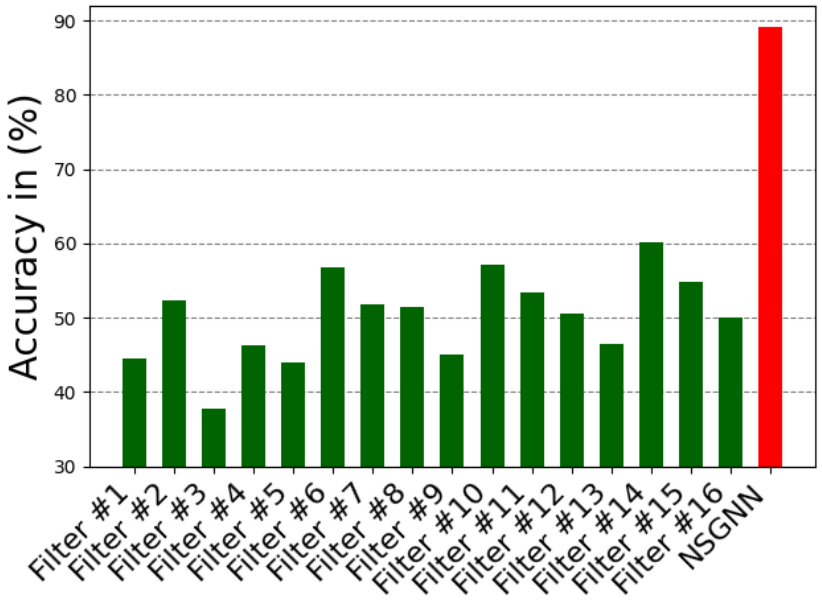


Figure 8: 16 NSGNN filters

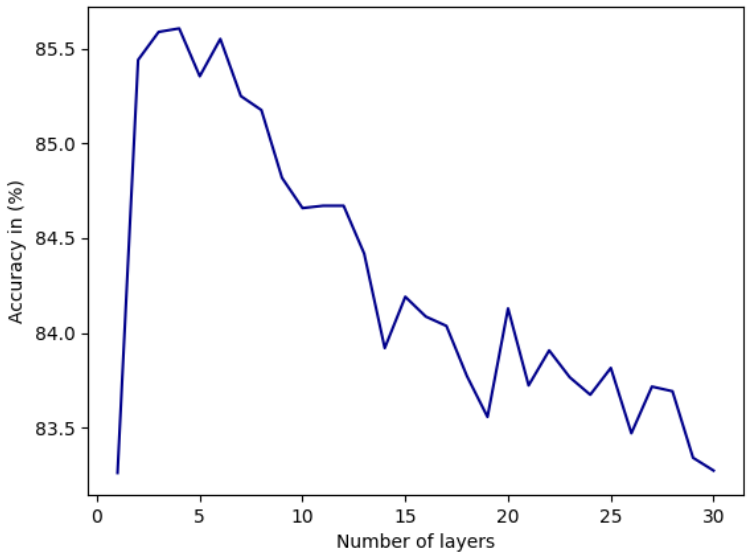


Figure 9: Number of layers (filters) x Avg. Accuracy

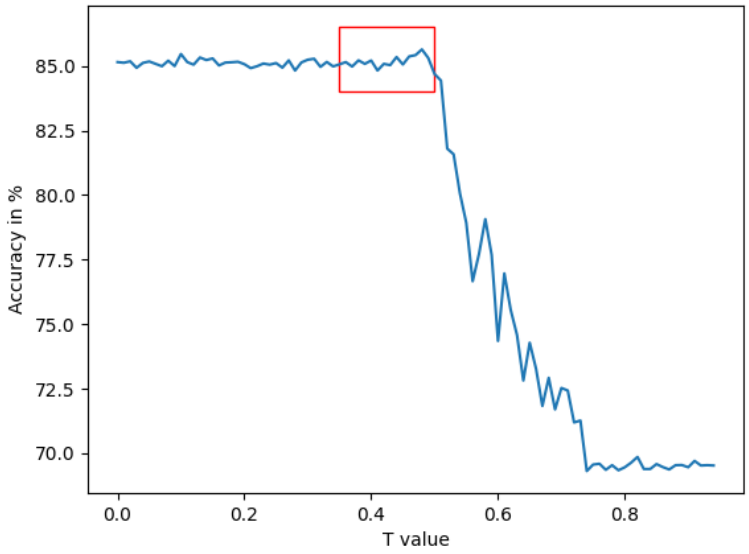


Figure 10: Threshold (T) x Accuracy

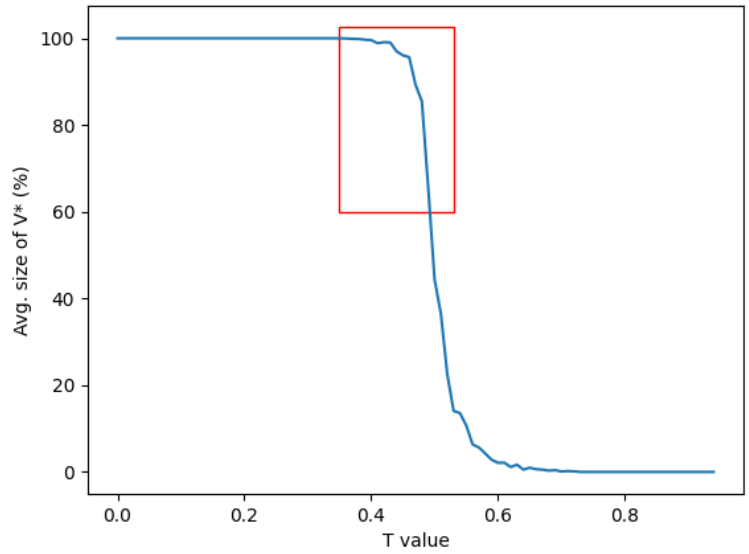


Figure 11: Threshold (T) x Size of $\mathcal{V}^*$

A.7 EFFECT OF THRESHOLD

A.8 EMBEDDINGS SIMILARITY

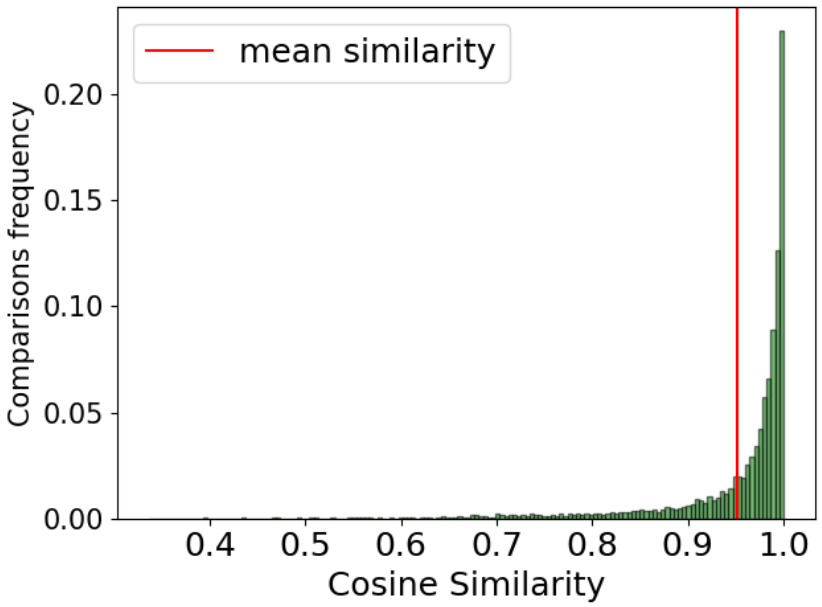
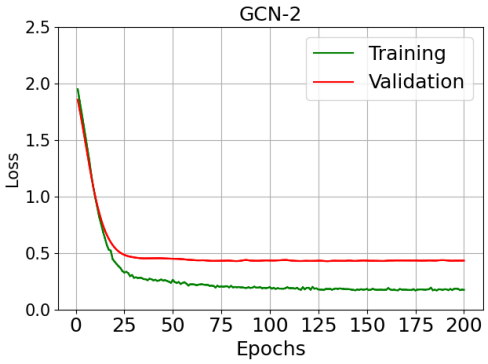
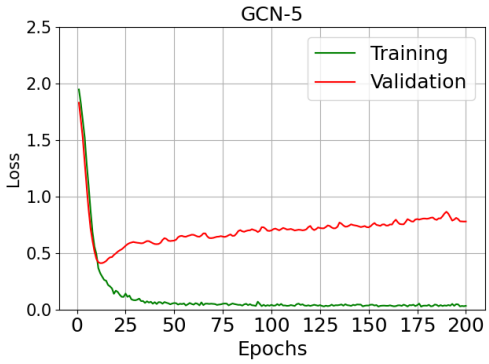


Figure 12: Histogram of embeddings cosine-similarity.

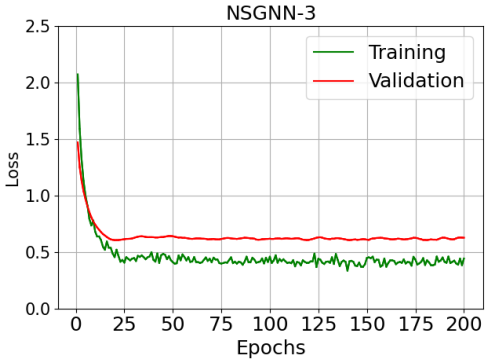
A.9 OVER-FITTING FOR NSGNN VS GCN



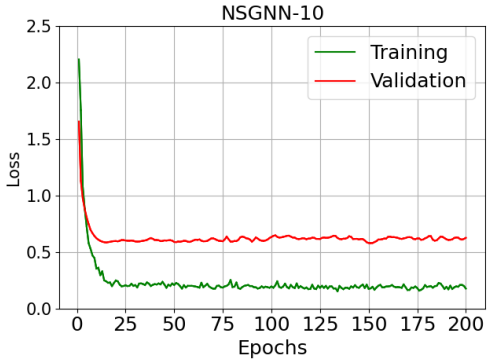
(a) Good-fit in a GCN model with 2 layers



(b) Over-fitting in a GCN model with 5 layers



(c) Good-fit in a NSGNN model with 3 layers



(d) Over-fitting in a NSGNN model with 10 layers

Figure 13: Comparisons of over-fitting vs. good-fitting in trained GCN (a,b) and NSGNN (c,d)

Our variant is less affected by over-smoothing. Particularly, the increase of additional parameters framework will generally impact our NODE-SELECT less severely than other GNN. Figure 13 below contrasts the training of our NODE-SELECT with GCN on the Cora dataset. On the left, figures 13a and 13c display well-fitted GCN (2 layers) and NSGNN (3 layers/ filters). On the right side, figures 13b and 13d illustrate the over-fitting in GCN (5 layers) and NSGNN (10 layers). Over-fitting is far more noticeable (severe) in the GCN model with the addition of 3 more layers than it is in NODE-SELECT with the addition of 7 more filters. The deviation between the training and validation loss-curves is much smaller for NSGNN than it is for GCN. In our experiments, the frameworks of GCN, GraphSAGE, MLP, and Node2vec were all to prone to the over-fitting issue while GAT showed similar resilience as our NODE-SELECT to the challenge. In essence, we deduce that the mechanisms indicative of a regularization techniques generally help against over-fitting. The self-attention in GAT and the combination of restrictive propagation with self-attention and data augmentation represent in our NODE-SELECT are examples of such mechanisms.