A Preprint

Steph-Yves Louis\textsuperscript{1}, Alireza Nasiri\textsuperscript{1}, Fatima J. Rolland\textsuperscript{2}, Cameron Mitro\textsuperscript{3}, and Jianjun Hu\textsuperscript{1}

\textsuperscript{1}Department of Computer Science & Engineering, University of South Carolina
\textsuperscript{2}Department of Applied Behavioral Analysis, Drexel University
\textsuperscript{3}School of Medicine, University of South Carolina

February 18, 2021

Abstract

While there exists a wide variety of graph neural networks (GNN) for node classification, only a minority of them adopt mechanisms that effectively target noise propagation during the message-passing procedure. Additionally, a very important challenge that significantly affects graph neural networks is the issue of scalability which limits their application to larger graphs. In this paper we propose our method named NODE-SELECT: an efficient graph neural network that uses subetting layers which only allow the best sharing-fitting nodes to propagate their information. By having a selection mechanism within each layer which we stack in parallel, our proposed method NODE-SELECT is able to both reduce the amount noise propagated and adapt the restrictive sharing concept observed in real world graphs. Our NODE-SELECT significantly outperformed existing GNN frameworks in noise experiments and matched state-of-the-art results in experiments without noise over different benchmark datasets.

Keywords Node Selection · Graph Neural Networks

1 Introduction

The use of deep learning techniques for graph analysis has become a very popular research topic in recent years \cite{1}. Commonly referred to as graph neural networks (GNN), these deep learning techniques now figure amongst the most used methods for learning from relational data \cite{1,2}. Just as in the functioning of convolutional neural networks (CNN), multiple convolution operations can also be applied to learn from non-Euclidean data \cite{1,2,3}. Various adaptations of GNNs have been proposed over the years for the purpose of node classification \cite{1,2}. These GNN adaptations mainly differ in regards to their node embedding techniques, their algorithms’ propagation or aggregation methods, and their model’s scalability \cite{1,2}. Examples of important GNN variants include GCN \cite{4}, GAT \cite{5}, GraphSAGE \cite{6}, DeepGCN \cite{7}, and Pairnorm as one of the most recent works \cite{8}.

Prior GNNs have made use of regularization, Chebyshev polynomials, feature normalization, attention, residual blocks, random sampling, and many other techniques which have further pushed state-of-the-art for GNN \cite{1,2,4,5,6,7,9,10}. Nevertheless, there still remain many factors that still present challenges to GNN \cite{1,2,3}. Two important factors that mostly get referenced in the literature are over-smoothing and overfitting. Over-smoothing, defined as excessive similarity of node representation, is a direct consequence of deeply stacking graph convolutional layers \cite{1,2,11,12}. Particularly, the over-smoothing issue has been justified to result from the fact that more noise gets shared than useful information during the convolution operations \cite{12}. On the other hand, overfitting is another issue which happens with adding more parameters and increasing the complexity of the model \cite{9}. Besides the over-smoothing and over-fitting issues, GNN also suffer from a noticeable conceptual limitation. Few GNN variants provide an implementation that fully mimic the relational rules observed in the networks of real world \cite{5,13,9,8,14,15}. Nonetheless, the number of GNN variants with mechanisms that easily translate to real-world networks is minimal and remains to be further exploited; particularly, scenarios in which there are consequences to letting any nodes propagate information \cite{16,17,18,19}.

With the motivation to mainly tackle this existing conceptual limitation, we propose a new kind of graph neural network named: NODE-SELECT\textsuperscript{1}. For this conceptual limitation, we introduce an efficient selection mechanism that prevents nodes with representation of poorer quality from propagating information. Beyond the selection process, we also learn a global weight weight coefficient for these propagating nodes and also

\textsuperscript{1}The codes for this work can be found at \url{https://github.com/superlouis/NODE-SELECT}
combine our memory-efficient layers in parallel as in the ensemble concept. To demonstrate the effectiveness and importance of our proposed method, we evaluate it on standard benchmark datasets with and without noise data [20][21][16]. Overall, our proposed NODE-SELECT considerably outperforms popular GNN frameworks on graphs augmented with noise vertices and marginally surpasses them on graph without noise vertices.

We summarize our contribution as four main points. (1) We implement a very important concept of node selection to graph neural networks. (2) We adapt the ensemble concept by stacking our graph convolutional layers in parallel and demonstrate how it benefits our framework. (3) We demonstrate through extensive experiments how current GNN can be considerably affected by the presence of harmful vertices representations during the message-passing but our NODE-SELECT is not affected by them. (4) We demonstrate that our proposed method is extremely scalable with the increase of graph sizes.

2 RELATED WORK

Researchers have used various techniques to do their convolution operation on the graph vertices. Examples of such techniques include the adaptation of gated recurrent units (GRU), Chebyshev polynomials, attention mechanisms, etc... GNN was the first framework to apply gated recurrent units to sequentially update the feature vectors of the graph nodes [22][23]. While ChebConv first used Chebyshev polynomials to do the node convolutions, the adaptation by GCN proved to be more effective thanks to its feature aggregation restriction and normalization trick [24][25].

The technical concept of sampling was introduced by the works of GraphSAGE and FastGCN [6][25]. Another important technical concept: attention, for graph neural networks, was first adapted in the framework of GAT [5].

In addition to the aforementioned methods, there exist many GNN variants that have further incrementally introduced other important techniques into the field of graph neural networks. Such architectures include DropEdge and DNA-Conv [9][26]. DropEdge proposes a regularization mechanism to address overfitting and over-smoothing by randomly removing connecting edges [9]. On the other hand, inspired by the concepts of Jumping Knowledge [26], Fey proposed a dynamic neighborhood aggregation mechanism to offer to their learning model a bigger range of feature information [10]. Nevertheless, there still remains conceptual limitations that still need to be addressed to further advance the field.

The main limitation we aim to address is the need for a straightforward adaptation of the natural message-passing mechanism often found in real-world graphs. In real graphs such as social networks, computer networks, brains, or molecules; we frequently observe an orderly communication between the units. Our goal is to implement this communication of real-world tasks within a social network [17][18][27]. This restriction of propagating vertices in a social setting is often paraphrased as 'Too many cooks spoil the broth'. In Neuroscience there has also been works that studied the topic that only a subset of neurons fire simultaneously in brain networks [28][29]. Just as there is a clear limitation in propagating vertices in some real-world graphs, we also implement in our NODE-SELECT a similar mechanism that grants our model the flexibility to adapt this restriction.

Besides the need of adapting this conceptual selection mechanism, we also assumed that the implementation of such mechanism would also technically benefit the network. This adapted selection mechanism could act as a regularization in the network while preventing the least sharing-fit nodes from propagating their embeddings [30][31]. This restriction would mainly reduce the amount of noise coming from particular nodes. With the cancellation of inappropriate nodes’ propagation, the network would also benefit in efficiency having to arguably do less node convolutions [24][1]. With the selection implemented within each layer, we also presumed that ensembling our layers could be more beneficial than sequentially them. The layers’ sequential stacking could lead to poorer performance if prior selections were sub-optimal. Also, ensembling the layers could result in a diverse generation of embeddings which would likely increase accuracy performance of the model [32].

3 PROPOSED METHOD

We begin by formulating an input graph $G$ as the set $G = (V, E)$, where $V$ and $E$ respectively define a set of nodes $v_i$ and a set of edges $e_{ij}$ connecting 2 nodes: $v_i$ and $v_j$. Also, we denote as $X = \{x_1,...,x_n\}$ or $[h^{(0)}_{i1}, ..., h^{(0)}_{in}] \in \mathbb{R}^{n \times F}$ the node features. For the task of node classification, a graph neural network needs to learn an embedding $h^{(l)}_i$ based on a prior embedding or feature vector $h^{(l-1)}_i$. The operation done by each layer can be expressed by a function $f^{(l)}_i$:

$$h^{(l)}_i = f^{(l)}_i(h^{(l-1)}_i, \Gamma), \Gamma \in \mathbb{R}^{F \times F}$$

where $\theta$ denotes the trainable parameters of layer $l$, $\Gamma$ any function aggregating localized neighborhood information, and $N_{(i)}$ the neighborhood of node $v_i$.

To update node $v_i$’s embedding, these layers generally utilize information from all neighboring nodes and need to be arranged in sequence within the network. However, such layout favours the potential issue of oversmoothing and lacks adequate techniques to prevent the propagation of noise from specific nodes. In contrast, we propose to use a selection mechanism $S(\cdot)$ to limit noise sharing between nodes and the combination of embeddings from independent layers to reduce noise propagation between layers.

3.1 Node Selection

The first step in our method consists of linearly transforming the initial feature vector $x_i$ using a matrix $W \in \mathbb{R}^{F \times F}$. The transformed feature vector $W_{xi}$, with reduced dimensionality, has the same cardinality as the embedding output. Subsequently, we estimate a random sensitivity value $\hat{p}_i$ as a measure to classify nodes with potentially harmful or useless embedding information. The goal is to use this $\hat{p}_i$ value so the network can detect nodes whose omission of information improves the training. We
utilize the sum of the neighbors embedding \( N_{(i)} \) to learn this value and then define our selection technique as:

\[
S(v_i) = \begin{cases} 
1, & \hat{p}_i \geq T \\
0, & \text{otherwise}
\end{cases}
\]  

(2)

where \( \hat{p}_i = \sigma \left( \sum_{j \in N_{(i)}} W_0 \cdot W_j \right) \), \( W_0 \in \mathbb{R}^{1 \times F'} \) is a weight matrix, \( T \) a determined threshold, and \( \sigma \) a non-linear transformation. Thus, \( \hat{p}_i \) can be interpreted as a normalized signal, ranging between 0 and 1, allowing the model to make its selection with respect to the entire graph. Namely, the learnable choice of whether or not a node needs to propagate its information is made on a global scale such that an un-selected node can no longer affect any other nodes.

3.2 Selective Aggregation and Feature Update

Once a node’s selection \( S(v_i) \) is learned, we allow the layer to aggregate the embedding information only from its selected neighbors. In other words, we want the model to keep learning embeddings even with the cancellation of some nodes’ propagation. The simplified form of this selectively aggregated information is simplified below:

\[
A_i = \sum_{j \in N_{(i)}} \alpha_j \cdot S(v_j) \cdot W x_j
\]  

(3)

where \( \alpha_i \) defines the propagation weight for each selected node. This propagation weight is calculated by linearly transforming the concatenated summed and selectively aggregated neighborhood embeddings; such that \( \alpha_i = \sigma \left( W_1 \left( \sum_{j \in N_{(i)}} S(v_j) \cdot W x_j \right) \right) \) and \( W_1 \in \mathbb{R}^{1 \times F'} \) is a weight matrix. Since our model learns the selection \( S(v_j) \) with respect to the entire graph (global selection), \( \alpha_i \) thereby corresponds to the hard attention for a selected node \( v_i \). Also in our experiments, applying this global weight \( \alpha_i \) resulted in higher ( \( \geq 3\% \) ) accuracy performance than adapting a weight relative to each node’s neighbor (i.e. \( \alpha_{ij} \) as in GAT).

Therefore, for layer \( l \) in our method, the simplified form of its function \( f^{(l)}_0 \) takes the form of:

\[
f^{(l)}_0(x_i) = \sigma \left( W x_i \right) + \sigma(A_i)
\]  

(4)

where \( W x_i \) is the originally transformed feature vector and \( A_i \) the pooled embedding information from just the selected nodes.

3.3 Parallel Stacking

The last step consists of simply summing the embedding information from all layers. Given \( L \) layers, the final embedding output for a given is the summation of all \( L \) independently learned embeddings \( f^{(l)}_0(x_i) \). This summation is described below.

\[
\text{output}_i = \sum_{l=1}^{L} f^{(l)}_0(x_i)
\]  

(5)

4 EXPERIMENTS

4.1 Datasets

To assess the performance of our proposed model, we conduct two sets of experiments using a total of 8 benchmark datasets.

In the first experiment, we utilize all 8 datasets: Cora, CiteSeer, PubMed, Cora Full, Coauthor CS, Coauthor Physics, Amazon Computers, and Amazon Photo. Cora, CiteSeer, and PubMed contain relational data on academic papers [20, 21]. Datasets Coauthor CS (Co-CS) and Coauthor Physics (Co-P) are co-authorship datasets from the Microsoft Academic Graph [16]. Lastly, Amazon Computers (Amz-C) and Amazon Photo (Amz-P) are graph datasets defining segments of the Amazon product categories graphs. For each dataset, we randomly split the nodes so that the training, validation, and testing sets follow a ratio of 20-20-60 percent. This split is repeated for 10 randomly chosen seeds which are used in each model experiment.

In the second experiment, we only use Cora, CiteSeer, and the PubMed datasets which we modify by adding noise data into their graphs. We increase the size of each graph by 10 and 25% using pseudo vertices. We attribute each pseudo vertex a feature vector from a standard normal distribution, a random label, and random neighbors from the original graph. We follow the same 20-20-60 splitting ratio as in the first experiment but afterwards remove the pseudo vertices from the testing set. This split is repeated for 5 randomly chosen seeds. Details of the Datasets are provided in Table 1.

4.2 Experimental Setup

We compare our proposed method to 6 GNN variants selected for either their robust performance, contrasting node sampling method, or both. These baselines include DropEdge, FastGCN, GAT, GCN, GraphSAGE, and Node2vec [9, 33, 5, 4, 6, 34]. Given that each framework performs differently under various training dynamics, we perform random hyper-parameter and only report results from the best performing models with respect to the validation set. We apply a fixed dropout rate of 0.5 after the GNN layers and use Adam as optimizer [35, 36]. We implement all the models using Pytorch and the library of Pytorch-Geometric [37, 25]. The hyper-parameters from the best performing models are provided in Table 1 of the Appendix.

4.3 Results

Table 1 displays the average accuracies over the 10 random splits from the first experiment. As seen, NODE-SELECT consistently
matches or outperforms the performance by the baselines by up to 1.4 percentage points. Table 3 lists the average classification accuracy of 5 random splits for the second experiment with noise introduced to the training. Once a GNN model is introduced to a considerably amount of noise information, the results demonstrate that the accuracy significantly drops [31]. Nevertheless, our NODE-SELECT is only marginally affected by the presence of noise information whereas the baselines considerably are. As shown in the results, our proposed method particularly stands out in these noise experiments by outperforming the other baselines by up to 20 percentage points. Simply put, the resilient ability of our network to be unaffected by noise information is due to the used selection mechanism which allows the network a direct control of blocking nodes propagating them.

5 Discussion

5.1 Parallel vs Sequential Stacking

Compared to the traditional sequential stacking of GNN layers, NODE-SELECT adopts the approach of stacking its layers in parallel. Based on our experiments, we have found that the parallel stacking of these selective layers yielded better and more stable results. Figure 2 illustrates a comparative study of these two stacking options. As demonstrated, the parallel setting proves to be more beneficial with its results reaching higher accuracy and lower variance. The sequential layout forces a layer \( l \) to depend on the set of selected nodes \( V_{l-1} \) of a previous layer \( l-1 \). In the rare cases that the composition of \( V_{l-1} \) is not completely suitable to the weights of layer \( l \), the model may result in a much lower performance; thus observing a higher variance and reduced accuracy. The parallel layout removes this dependence issue by stacking its layers side-by-side and having them learn independently as in the ensemble method [38].

5.2 Layers Operation

Because of the parallel configuration, any given NODE-SELECT layer operates independently. Each layer separately makes a node selection that yields to a node embedding. Figures 3 and 4 contrast these layers’ differences in terms of their predicted sensitivity value, accuracy score, and proportion of selected nodes for experiments on the Cora and Pubmed Dataset. In Figure 3, the \( \hat{p}_i \) values of 3 nodes from the training set are predicted by 3 parallel layers on a Cora experiment. Because of the layers’ independence, a node’s \( \hat{p}_i \) values from distinct layers are also independent. The independence is illustrated with Layer-1 learning to output high values above the threshold for...
Table 2: Results from the first experiment with the 8 standard benchmarks. Average testing accuracy (%) and standard deviation from 10 random splits are listed.

| Variant          | CiteSeer | Cora   | PubMed | Co-P   | Co-CS  | Cora Full | Amz-C | Amz-P |
|------------------|----------|--------|--------|--------|--------|-----------|-------|-------|
| DropEdge-GCN     | 56.2±1.6 | 83.5±2.3 | 87.1±0.5 | 95.9±0.1 | 92.8±0.6 | 57.4±1.8 | 84.9±3.4 | 89.2±4.1 |
| FastGCN          | 74.0±1.0 | 82.1±2.6 | 87.6±0.5 | 95.5±0.3 | 92.2±0.4 | 60.8±1.0 | 83.5±2.2 | 91.0±0.9 |
| 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 |
| GraphSAGE        | 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 |
| 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 |
| NODE-SELECT      | 74.1±1.1 | 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 3: Results from the second experiment with the Cora, Citeseer, and Pubmed benchmarks. Graphs sizes from the datasets are increased by 10 and 25% from the addition of noise data. Average testing accuracy (%) and standard deviation from 5 random splits are listed.

| Variant          | +10% | +25% | +10% | +25% | +10% | +25% |
|------------------|------|------|------|------|------|------|
| CiteSeer         |      |      |      |      |      |      |
| Cora             |      |      |      |      |      |      |
| Pubmed           |      |      |      |      |      |      |
| DropEdge-GCN     | 42.0±2.3 | 35.6±1.4 | 38.1±3.4 | 34.4±2.9 | 74.4±12.0 | 46.2±12.3 |
| FastGCN          | 33.7±0.9 | 29.9±1.5 | 40.8±2.1 | 30.3±1.2 | 57.3±0.9 | 47.9±1.3 |
| GAT              | 34.1±1.2 | 34.1±1.8 | 61.7±2.0 | 58.0±1.4 | 55.0±4.8 | 52.5±7.5 |
| GCN              | 56.0±0.8 | 49.3±1.1 | 74.3±1.7 | 65.2±1.3 | 58.4±0.6 | 54.8±0.6 |
| GraphSAGE        | 35.4±1.7 | 33.9±0.7 | 52.8±1.3 | 51.9±1.7 | 45.4±0.5 | 42.7±1.1 |
| Node2vec         | 38.7±0.8 | 35.4±0.8 | 58.4±1.4 | 51.1±1.3 | 62.9±0.4 | 58.5±0.3 |
| NODE-SELECT      | 68.6±0.4 | 64.8±2.0 | 80.9±3.8 | 78.4±1.8 | 83.6±0.9 | 79.7±0.8 |

Figure 3: Learning plot of $p_i$ during a NODE-SELECT model training. Red dashed line - - - represents the threshold parameter $T$. For each node (65, 1601, 1728), an independent layer learns its selection (allowing it to share its embedding information). A given node $v_i$ is selected when its $p_i$ signal is at least as high as the threshold $T$. 

5.3 Scalability

A direct benefit of stacking our layers in parallel is that our method is very scalable. Because our method is composed of ensembled 1-layer GNN, its memory usage is very effective. For example, the number of trainable parameters can be estimated with $L(F'(3 + F))$; with $L$ describing the number of layers, $F'$ the output dimension, and $F$ the input dimension. Figure 4 contrasts how NODE-SELECT scales with larger graphs compared to some baseline frameworks. As demonstrated, NODE-SELECT scales to larger graphs comparably to GCN and much better than GAT, GraphSAGE, and DropEdge. As the graph gets larger, the amount of memory required for the learning to take place also increases and the challenge of be-
Table 4: Results of using variable number of layers. Rows in blue are for under-fit models and rows in red for over-fit models.

| # of Layers | Layers mean accuracy | Model accuracy | Layers avg. size of selection |
|-------------|----------------------|----------------|-------------------------------|
| 1           | 94.0                 | 94.0           | 78%                           |
| 5           | 93.8                 | 95.0           | 81%                           |
| 10          | 93.0                 | 94.8           | 52%                           |
| 20          | 85.5                 | 94.8           | 64%                           |
| 100         | 62.3                 | 93.5           | 66%                           |

Figure 4: Accuracy score by layer and output for model trained on Pubmed. Blue color displays the percentage of selected nodes $V^*(l)$. The green color displays the reported accuracy of the final output, obtained by summing the embeddings from the 10 layers.

Figure 5: Scalability of NODE-SELECT compared to other methods when graph size increases.

5.4 Effect of Parameters $L$ and $T$

In contrast to other GNNs, NODE-SELECT only has two configurable parameters that affect the model’s performance. The parameter $L$ guides the model’s fitting behavior while $T$ guides its selection mechanism. In our study, we found that any arbitrary number of layer leads to the a good performance. However, depending on the size and properties of the graph, too few layers may cause the model to under-fit while too many to over-fit. Table 4 provides a simple illustration of the effect of increasing the parameter $L$. Using only 1 layer causes the model to under-fit with an accuracy of 94%. Using 20 or more (100) layers causes the model to over-fit with accuracy scores that are below the well-fit models trained with 5 or 10 layers. Particularly, as a NODE-SELECT model uses more layers (past the optimal number), each individual layer becomes weaker. This decrease in performance results from the fact that each layer learns the patterns pertaining to a specific region in a graph and thus generalizes poorly.

A NODE-SELECT model will adjust its weights so as to retain as much embedding information as possible during training. Therefore, a NODE-SELECT only begins to cancel nodes when the threshold parameter is not too small; for instance above $T$ value of 0.3. However, using low values for the threshold results in performance that is only marginally lower than with moderate threshold. Also, the selection mechanism is a lot more effective when the graph is not small. In our experiments with smaller graphs (Cora and Citeseer), an effective layer only cancelled a minimal number of nodes (i.e. between 0 and 10%). Figure 6 displays the effect of changing $T$ on the model accuracy. Using the parameter $T$ allows the model to conservatively remove subset of nodes that are not needed to lower its loss. Using a $T$ value in a range of $0.3 \leq T \leq 0.49$ gives the best results in terms of both accuracies and node cancellations. However the application of a large $T$ value leads the model to cancel too many nodes, thereby losing crucial information from potentially important nodes.

6 CONCLUSION

We introduced NSGNN, a novel graph neural network for node-classification, which learns node embeddings by summing correlated embeddings learned by its layers. Inspired by the functioning of real-world graphs, our NODE-SELECT addresses...
the conceptual limitation of selective propagation based on the nodes global importance. As opposed to 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. Besides the reaching state-of-the-art performance in experiments which introduced noise propagation, its scalability to larger graphs is much more effective than other baselines. With a simple selection mechanism that allows our model to effectively adapt to the problem of noise-propagating nodes, we expect that our proposed method can adapt to real world problems where such mechanism can be very important such as in Botnet detection or cancellation of particular instances within a graph. Further research may also be done to additionally improve our method’s performance by testing other ways to combine the independent layers’ embedding or how other ways to ensemble the separate layers (i.e. boosting).

References

[1] 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.

[2] Zhonghan 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.

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

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

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

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

[7] Guohao Li, Matthias Muller, Ali Thabet, and Bernard Ghanem. Deepgcn: Can gcls go as deep as gcnns? In Proceedings of the IEEE International Conference on Computer Vision, pages 9267–9276, 2019.

[8] Lingxiao Zhao and Leman Akoglu. Pairnorm: Tackling oversmoothing in gnnns. In International Conference on Learning Representations, 2019.

[9] 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.

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

[11] 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.

[12] 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.

[13] 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.

[14] Steph-Yves Louis, Yong Zhao, Alireza Nasiri, Xiran Wang, Yuqi Song, Fei Liu, and Jianjun Hu. Graph convolutional neural networks with global attention for improved materials property prediction. Physical Chemistry Chemical Physics, 22(32):18141–18148, 2020.

[15] Meng Liu, Hongyang Gao, and Shuiwang Ji. Towards deeper graph neural networks. In Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pages 338–348, 2020.

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

[17] 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.

[18] Alexandre 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.

[19] Maryam Feily, Alireza Shahrestani, and Sureswaran Radmardass. A survey of botnet and botnet detection. In 2009 Third International Conference on Emerging Security Information, Systems and Technologies, pages 268–273. IEEE, 2009.
Preprint – NODE-SELECT: A Graph Neural Network Based On A Selective Propagation Technique

[20] 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.

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

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

[23] 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.

[24] 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.

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

[26] 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:1806.05536*, 2018.

[27] Boris Groysberg, Jeffrey T Polzer, and Hillary Anger Elfenbein. Too many cooks spoil the broth: How high-status individuals decrease group effectiveness. *Organization Science*, 22(3):722–737, 2011.

[28] 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.

[29] Takuya Sasaki, Norio Matsuki, and Yuji Ikegaya. Interneuron firing precedes sequential activation of neuronal ensembles in hippocampal slices. *European Journal of Neuroscience*, 39(12):2027–2036, 2014.

[30] Samet Oymak. Learning compact neural networks with regularization. In *International Conference on Machine Learning*, pages 3966–3975. PMLR, 2018.

[31] James Fox and Sivasankaran Rajamanickam. How robust are graph neural networks to structural noise? *arXiv preprint arXiv:1912.10206*, 2019.

[32] Zohaib M Jan, Brijesh Verma, and Sam Fletcher. Optimizing clustering to promote data diversity when generating an ensemble classifier. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 1402–1409, 2018.

[33] Jie Chen, Tengfei Ma, and Cao Xiao. Fastgcn: fast learning with graph convolutional networks via importance sampling. *arXiv preprint arXiv:1801.10247*, 2018.

[34] 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.

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

[36] 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.

[37] 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.

[38] Thomas G Dietterich. Ensemble methods in machine learning. In *International workshop on multiple classifier systems*, pages 1–15. Springer, 2000.
7 Appendix

7.1 Effect of Number of layers

There is an optimal number of layers that should be included in a NODE-SELECT model. Exceeding this number of layers increases the likelihood the model will over-fit while using less causes it to underfit. Figure 7 displays the impact of surpassing the optimal number of layers (3-5) in the Cora dataset. At the optimal number of layers, the model’s performance (average accuracy for 10 random splits) is at its peak. However, the increase in the number of layers prior to reaching that optimal number increases the performance of the NSGNN model. As more layers 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 layers, which together complement each the other’s inaccuracies. Figure 9 illustrates the effects of the addition of layers (in green) on the overall model (in red) on the Amazon-Computers dataset. In Figure 8, the model’s accuracy reaches an accuracy score of about 82% while best layers obtains accuracy at exactly 70%. As 5 more layers are added in the illustration of figure 9, the highest accuracy reached by any layers drops 62% while the NSGNN accuracy improves to 87%.

7.2 Comparison of Layers’ embeddings

Compared to the traditional sequential stacking of layers, NODE-SELECT adopts the well-known ensemble method for which each layer 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 ($\geq 0.85$). Henceforth, by combining these sets of independent yet correlated embeddings, the model subsequently learns a final embedding that is more precise. Figure 10 resulting embeddings from 10 independent layers for the first node on the Co-P experiment. Noticeably, the majority of the embeddings are very similar. Based on our experiments, we deduced that not all of the layers’ embeddings need to match for a good node prediction. As long as there is a general harmony in a sufficient number of layers, the model will depend on these more frequent harmonious embeddings to form its final embedding.
7.3 Complex NODE-SELECT Layer

Beyond the simple selection mechanism that we presented in our paper, we also implemented a sequence mechanism that is more appropriate for smaller graphs. Mainly, this additional sequence mechanism allows each selected node to propagate information up to its $q$-hop neighbours. Figure 11 provides an illustration of this sequence mechanism.

Starting with the subset of propagating nodes, a layer sequentially learns a global weight coefficient to perform the message-passing operation across $Q$-hop neighborhoods. For each $q = 0, 1, ..., Q-1$ depths, a corresponding weight coefficient $\alpha_i^{(q)}$ is calculated:

$$\alpha_i^{(q)} = \sigma(W_2 (y_i^{(q)} \parallel q^*)) \tag{6}$$

, in which $W_2 \in \mathbb{R}^{1 \times (F+Q)}$ denotes a learnable matrix, $y_i^{(q)}$ the $q$th updated feature vector of $v_i$, and $q^*$ the one-hot encoded vector of the depth $q$. Upon learning this selection-depth adapted coefficient, a node’s feature updates as:

$$y_i^{(q+1)} = y_i^{(q)} + \sum_{j \in N_0(i)} (\alpha_j^{(q)} \cdot y_j^{(q)}) \tag{7}$$

Lastly, we implement a noise layering mechanism so as to regulate the amount of noise information that is being learned through 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. We adapt our updating operation so that the layer tries to maintain an appropriate balance between learned neighboring information and each node’s own feature. Therefore, after $Q$ updates, the layer then calculates a final global coefficient $c_i$ though the use of a matrix $W_3 \in \mathbb{R}^{1 \times (2F)}$,

$$c_i = \sigma(W_3 (y_i^{(Q-1)} \parallel y_i^{(q=0)})) \tag{8}$$

, 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 layer potential noise acquired during the aggregation process. 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 11: Sequential message propagation in layer. First, the layer selects a subset of propagating nodes based on their global importance. Those selected vertices are represented in blue: $V_{S_l} = \{2, 4, 5, 9\}$. At depth $q = 0$, only the selected nodes $\{2, 4, 5, 9\}$ are allowed to share a proportion $\alpha_{i,j}^{(0)}$ of their embedding. At depth $q = 1$, only the 1-hop neighbors of the initially selected nodes are allowed to share another proportion $\alpha_{i,j}^{(1)}$ of their updated contribute to the message-passing.
Table 5: 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 |
|-----------|---------|------|---------------|
| FastGCN   | CiteSeer| 74.0±1.0 | 2/16/0.001/0.0005/— |
|           | Cora    | 82.1±2.6 | 2/64/0.01/0.0005/— |
|           | PubMed  | 87.6±0.5 | 2/16/0.005/0.0005/— |
|           | Co-P    | 95.5±0.3 | 3/64/0.005/0.0005/— |
|           | Co-CS   | 92.2±0.4 | 3/128/0.005/0.0005/— |
|           | Cora Full| 60.8±1.0 | 3/128/0.005/0.0005/— |
|           | Amz-C   | 83.5±2.2 | 3/128/0.005/0.0005/— |
|           | Amz-P   | 91.0±0.9 | 3/128/0.005/0.0005/— |

| 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 Full| 64.8±0.5 | 2/128/0.0005/0.00005/attention-heads:8 |
|           | Amz-P   | 93.7±0.6 | 2/128/0.0005/0.00005/attention-heads:8 |
|           | Amz-C   | 90.0±0.7 | 2/128/0.0005/0.00005/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/— |

| Node2vec  | *       | *      | 1/64/0.005/—/— |

| NODE-SELECT | CiteSeer| 74.1±1.1 | 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 |