Robust Graph Representation Learning via Neural Sparsification

Anonymous authors
Paper under double-blind review

Abstract
Graph representation learning serves as the core of many important prediction tasks, ranging from product recommendation in online marketing to fraud detection in financial domain. Real-life graphs are usually large with complex local neighborhood, where each node is described by a rich set of features and easily connects to dozens or even hundreds of neighbors. Most existing graph learning techniques rely on neighborhood aggregation, however, the complexity on real-life graphs is usually high, posing non-trivial overfitting risk during model training. In this paper, we present Neural Sparsification (NeuralSparse), a supervised graph sparsification technique that mitigates the overfitting risk by reducing the complexity of input graphs. Our method takes both structural and non-structural information as input, utilizes deep neural networks to parameterize the sparsification process, and optimizes the parameters by feedback signals from downstream tasks. Under the NeuralSparse framework, supervised graph sparsification could seamlessly connect with existing graph neural networks for more robust performance on testing data. Experimental results on both benchmark and private datasets show that, NeuralSparse can effectively improve testing accuracy and bring up to 7.4% improvement when working with existing graph neural networks on node classification tasks.

1 Introduction
Representation learning has been in the center of many machine learning tasks on graphs, such as name disambiguation in citation networks (Zhang et al., 2018c), spam detection in social networks (Akoklu et al., 2015), recommendations in online marketing (Ying et al., 2018a), and many others (Hamilton et al., 2017; Li et al., 2018). As a class of models that can simultaneously utilize non-structural (e.g., node and edge features) and structural information in graphs, Graph Neural Networks (GNNs) (Kipf & Welling, 2017; Hamilton et al., 2017; Li et al., 2016) construct effective representations for downstream tasks by iteratively aggregating neighborhood information (Kipf & Welling, 2017; Hamilton et al., 2017). Such methods have demonstrated state-of-the-art performance in classification and prediction tasks on graph data (Veličković et al., 2018; Chen et al., 2018; Xu et al., 2019; Veličković et al., 2019).

Meanwhile, graphs from real-life applications are usually large with complex local neighborhood, where each node has rich features and dozens or even hundreds of neighbors. As shown in Figure 1(a), this subgraph from Transaction dataset (detailed in Section 5.1) consists of 38 nodes (i.e., promising organizations and other organizations) with average node degree 15 and node feature dimension 120. The GNNs are expected to grasp useful patterns from neighboring nodes; however, as representative patterns are diluted by overwhelming information in local neighborhood, graph learning algorithms could be misled by neighborhood aggregation. Such complexity in input graphs poses non-trivial overfitting risk to existing GNN based learning techniques.

While it is straightforward yet expensive (sometimes even impractical) to address this overfitting problem by increasing the number of labeled samples, we investigate a cheaper alternative of reducing input graph complexity by graph sparsification in this work. Graph sparsification (Liu et al., 2018; Zhang & Patone, 2017) aims to find smaller subgraphs from input large graphs that best preserve desired properties. Existing sparsification methods could lead to suboptimal performance for downstream prediction tasks: (1) these methods are unsupervised such that the resulting sparsified
graphs may not favor downstream tasks; and (2) they only consider structural information for sparsi-
ification decision, while non-structural information in graphs, such as node/edge features, could have
non-trivial impact to the quality of sparsification. Recently, there have been GNN models attempt-
ing to sample subgraphs from predefined distributions (Leskovec & Faloutsos, 2006; Adhikari et al.,
2018; Hamilton et al., 2017; Chen et al., 2018). As the predefined distributions could be irrelevant
to subsequent tasks, the sparsified graphs may miss important information for downstream tasks,
leading to suboptimal prediction performance.

**Present work.** We propose Neural Sparsification (NeuralSparse), a general framework that simulta-
neously learns graph sparsification and graph representation by feedback signals from downstream
tasks. The NeuralSparse consists of two major components: sparsification network and GNN. For
the sparsification network, we utilize a deep neural network to parameterize the sparsification pro-
cess: how to select edges from one-hop neighborhood given a fixed budget. In the training phase,
the network learns to optimize a sparsification strategy that favors downstream tasks. In the testing
phase, the network sparsifies input graphs following the learned strategy, instead of sampling sub-
graphs from a predefined distribution. Unlike conventional sparsification techniques, our technique
takes both structural and non-structural information as input and optimizes the sparsification strategy
by feedback from downstream tasks, instead of using (possibly irrelevant) heuristics. For the GNN
component, the NeuralSparse feeds the sparsified graphs to a GNN and learns a graph representation
for subsequent prediction tasks.

Under the framework of NeuralSparse, we are able to leverage the standard stochastic gradient
descent and backpropagation techniques to simultaneously optimize graph sparsification and repre-
sentation. As shown in Figure 1(b), the graph sparsified by the NeuralSparse has lower complexity
with average node degree around 5. As a result (illustrated in Figure 1(c)), the testing classification
accuracy on the sparsified graph is improved by 15%, compared with its counterpart in the orig-
inal input graph, while conventional techniques could not offer competitive sparsification for the
classification task.

Experimental results on both public and private datasets show that the NeuralSparse is able to con-
sistently provide improved performance for existing GNNs on node classification tasks, bringing up
to 7% improvement.

## 2 Related Work

Our work is related to two lines of research: graph sparsification and graph representation learning.

**Graph sparsification.** The goal of graph sparsification is to find small subgraphs from input large
dgraphs that best preserve desired properties. Existing techniques are mainly unsupervised and deal
with simple graphs without node/edge features for preserving predefined graph metrics (Hübler
et al., 2008), information propagation traces (Mathioudakis et al., 2011), graph spectrum (Calan-
Our work contributes from a unique angle: by mitigating the overfitting risk from input graphs with reduced complexity, our technique can further boost testing performance of existing GNNs.

3 Proposed Method: NeuralSparse

In this section, we introduce the core idea of our method. We start with the notations that are frequently used in this paper. We then describe the theoretical justification behind NeuralSparse and our architecture to tackle the supervised node classification problem.

Notations. In this paper, we represent an input graph of \( n \) nodes as \( G = (V, E, A) \): (1) \( V \in \mathbb{R}^{n \times d_n} \) includes node features with dimensionality \( d_n \); (2) \( E \in \mathbb{R}^{n \times n} \) is a binary matrix where \( E(u, v) = 1 \) if there is an edge between node \( u \) and node \( v \); (3) \( A \in \mathbb{R}^{n \times n \times d_e} \) encodes input edge features of dimensionality \( d_e \). In addition, we use \( Y \) to denote the prediction target in downstream tasks (e.g., \( Y \in \mathbb{R}^{n \times d_l} \) if we are dealing with a node classification problem with \( d_l \) classes).

Theoretical justification. From the perspective of statistical learning, the key of a defined prediction task is to learn \( P(Y \mid G) \), where \( Y \) is the prediction target and \( G \) is an input graph. Instead of directly working with original graphs, we would like to leverage sparsified subgraphs to mitigate overfitting risks. In other words, we are interested in the following variant,

\[
P(Y \mid G) \approx \sum_{g \in S_G} P(Y \mid g)P(g \mid G),
\]

where \( g \) is a sparsified subgraph, and \( S_G \) is a class of sparsified subgraphs of \( G \).
In general, because of the combinatorial complexity in graphs, it is intractable to enumerate all possible \( g \) as well as estimate the exact values of \( P(Y \mid g) \) and \( P(g \mid G) \). Therefore, we approximate the distributions by tractable functions,

\[
\sum_{g \in S_G} P(Y \mid g)P(g \mid G) \approx \sum_{g \in S_G} Q_{\theta}(Y \mid g)Q_{\phi}(g \mid G) \tag{2}
\]

where \( \theta \) and \( \phi \) are parameters of \( Q_{\theta} \) and \( Q_{\phi} \), respectively.

Moreover, to make the above graph sparsification process differentiable, we employ reparameterization tricks (Jang et al., 2017) to make \( Q_{\phi}(g \mid G) \) directly generate differentiable samples, such that

\[
\sum_{g \in S_G} Q_{\theta}(Y \mid g)Q_{\phi}(g \mid G) \propto \sum_{g' \sim Q_{\phi}(G)} Q_{\theta}(Y \mid g') \tag{3}
\]

where \( g' \sim Q_{\phi}(g \mid G) \) means \( g' \) is a random sample drawn from \( Q_{\phi}(g \mid G) \).

To this end, the key is how to find appropriate approximation functions \( Q_{\phi}(g \mid G) \) and \( Q_{\theta}(Y \mid g) \).

**Architecture.** In this paper, we propose Neural Sparsification (NeuralSparse) to implement the theoretical framework discussed in Equation 3. As shown in Figure 2, NeuralSparse consists of two major components: sparsification network and GNNs.

- The sparsification network is a multi-layer neural network that implements \( Q_{\phi}(g \mid G) \): Taking \( G \) as input, it generates a random sparsified subgraph of \( G \) drawn from a learned distribution.
- GNNs implement \( Q_{\theta}(Y \mid g) \) that takes a sparsified subgraph as input, extracts node representations, and makes predictions for downstream tasks.

**Algorithm 1** Training algorithm for NeuralSparse

**Input:** graph \( G = (V, E, A) \), integer \( l \), and training labels \( Y \).

1: while stop criterion is not met do
2: Generate sparsified subgraphs \( \{g_1, g_2, \ldots, g_l\} \) by sparsification network (Section 4);
3: Produce prediction \( \{\hat{Y}_1, \hat{Y}_2, \ldots, \hat{Y}_l\} \) by feeding \( \{g_1, g_2, \ldots, g_l\} \) into GNNs;
4: Calculate loss function \( J \);
5: Update \( \phi \) and \( \theta \) by descending \( J \)
6: end while

As the sparsified subgraph samples are differentiable, the two components can be jointly trained using gradient descent based backpropagation techniques from a supervised loss function, as illustrated in Algorithm 1. While the GNNs have been widely investigated in recent works (Kipf & Welling, 2017; Hamilton et al., 2017; Veličković et al., 2018), we focus on the practical implementation for sparsification network in the remaining of this paper.

4 SPARSIFICATION NETWORK

Following the theory discussed above, the goal of sparsification network is to generate sparsified subgraphs for input graphs, serving as the approximation function \( Q_{\phi}(g \mid G) \). Therefore, we need to answer the following three questions in sparsification network. i) What is \( S_G \) in Equation 1, the class of subgraphs we focus on? ii) How to sample sparsified subgraphs? iii) How to make sparsified subgraph sampling process differentiable for the end-to-end training? In the following, we address the questions one by one.

**k-neighbor subgraphs.** We focus on \( k \)-neighbor subgraphs for \( S_G \) (Sadhanala et al., 2016): Given an input graph, each node of a \( k \)-neighbor subgraph can select no more than \( k \) edges from its one-hop neighborhood. Although the concept of sparsification network is not limited to a specific class of subgraphs, we choose \( k \)-neighbor subgraphs for the following reasons.

- We are able to make trade-off between model accuracy and graph complexity by tuning the hyperparameter \( k \). Intuitively, when \( k \) is smaller, aggregation complexity in GNNs becomes smaller with less overfitting risk, but meanwhile the model prediction capability could be also lower as
less structural information is preserved. It could be difficult to set a golden hyper-parameter that works all time, but one has the freedom to choose the $k$ that is the best fit for a specific task.

- $k$-neighbor subgraphs are friendly to parallel computation. As each node selects its edges independently from its neighborhood, we can utilize tensor operations in existing deep learning frameworks, such as tensorflow (Abadi et al., 2016), to speed up the sparsification process.

**Sampling $k$-neighbor subgraphs.** Given $k$ and an input graph $G = (V, E, A)$, we obtain a $k$-neighbor subgraph by repeatedly sampling edges for each node in the original graph. Without loss of generality, we sketch this sampling process by focusing on a specific node $u$ in graph $G$. Let $N_u$ be the set of one-hop neighbors of node $u$.

1. $v \sim f_\phi(V(u), V(N_u), A(u))$, where $f_\phi(\cdot)$ is a function that generates a one-hop neighbor $v$ from the learned distribution based on node $u$’s attributes, node attributes of $u$’s neighbors $V(N_u)$, and their edge attributes $A(u)$. In particular, the learned distribution is encoded by parameters $\phi$.
2. Edge $E(u, v)$ is selected for node $u$.
3. The above two steps are repeated $k$ times.

Note that the above process performs sampling without replacement. Given a node $u$, each of its adjacent edges is selected at most once. Moreover, the sampling function $f_\phi(\cdot)$ is shared among nodes; therefore, the number of parameters $\phi$ is independent of the input graph size.

**Making samples differentiable.** While conventional methods are able to generate discrete samples (Sadhanala et al., 2016), these samples are not differentiable such that it is difficult to utilize them to optimize sample generation. To make samples differentiable, we propose a Gumbel-Softmax based multi-layer neural network to implement the sampling function $f_\phi(\cdot)$ discussed in above.

To make the discussion self-contained, we briefly discuss the idea of Gumbel-Softmax. Gumbel-Softmax is a reparameterization trick used to generate differentiable discrete samples (Jang et al., 2017; Maddison et al., 2017). Under appropriate hyper-parameter settings, Gumbel-Softmax is able to generate continuous vectors that are as “sharp” as one-hot vectors widely used to encode discrete data.

Without loss of generality, we focus on a specific node $u$ in a graph $G = (V, E, A)$. Let $N_u$ be the set of one-hop neighbors of node $u$. We implement $f_\phi(\cdot)$ as follows.

1. $\forall v \in N_u,$
   
   $z_{u,v} = \text{MLP}_\phi(V(u), V(v), A(u,v)),$

   where $\text{MLP}_\phi$ is a multi-layer neural network with parameters $\phi$.
2. $\forall v \in N_u,$ we employ a softmax function to compute the probability to sample the edge,
   
   $\pi_{u,v} = \frac{\exp(z_{u,v})}{\sum_{w \in N_u} \exp(z_{u,w})}$

3. Using Gumbel-Softmax, we generate differentiable samples
   
   $x_{u,v} = \frac{\exp((\log(\pi_{u,v}) + \epsilon_v)/\tau)}{\sum_{w \in N_u} \exp((\log(\pi_{u,w}) + \epsilon_w)/\tau)}$

   where $x_{u,v}$ is a scalar, $\epsilon_v = -\log(-\log(s))$ with $s$ randomly drawn from Uniform(0, 1), and $\tau$ is a hyper-parameter called temperature which controls the interpolation between discrete distribution and continuous categorical densities.

Note that when we sample $k$ edges, the computation for $z_{u,v}$ and $\pi_{u,v}$ only needs to be performed once. For the hyper-parameter $\tau$, we discuss how to tune it as follows.

**Discussion on temperature tuning.** The behavior of Gumbel-Softmax is governed by a hyper-parameter $\tau$ called temperature. In general, when $\tau$ is small, the Gumbel-Softmax distribution resembles the discrete distribution, which induces strong sparsity; however, small $\tau$ also introduces high variance gradient that blocks effective backpropagation. A high value of $\tau$ cannot produce expected sparsification effect. Following the practice in (Jang et al., 2017), we adopt the strategy by starting the training with a high temperature and anneal to a small value with a guided schedule.
Sparsification algorithm and its complexity. As shown in Algorithm 2, given hyper-parameter $k$, the sparsification network visits each node’s one-hop neighbors $k$ times. Let $m$ be the total number of edges in the graph. The complexity of sampling subgraphs by the sparsification network is $O(km)$. When $k$ is small in practice, the overall complexity is $O(m)$.

Comparison with multiple related methods. Unlike GraphSAGE (Hamilton et al., 2017), FastGCN (Chen et al., 2018), and AS-GCN (Huang et al., 2018) that incorporate layer-wise node samplers to reduce the complexity of GNNs, NeuralSparse samples subgraphs before applying GNNs. In contrast to GraphSAGE and FastGCN, the NeuralSparse is parameterized and jointly trained with downstream GNNs. As for the computation complexity, the sparsification in NeuralSparse is more friendly to parallel computation than the layer-conditioned approach in AS-GCN. Compared with Graph Attention Models (Veličković et al., 2018; Zhang et al., 2018a), the NeuralSparse can produce sparser neighborhood, which effectively mitigates overfitting risks.

5 Experimental Study

In this section, we evaluate our proposed NeuralSparse on node classification task, including inductive and transductive settings. We demonstrate that NeuralSparse achieves superior classification performance over state-of-the-art GNN models. Moreover, we provide a case study to demonstrate how sparsified subgraphs generated by NeuralSparse could improve classification. The supplementary material contains more detailed experimental information.

5.1 Datasets

We employ five datasets from various domains and conduct node classification task following the settings as described in Hamilton et al. (2017); Kipf & Welling (2017). The dataset statistics are summarized in Table 1.

<table>
<thead>
<tr>
<th>Task</th>
<th>Reddit</th>
<th>PPI</th>
<th>Transaction</th>
<th>Cora</th>
<th>Citeseer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nodes</td>
<td>232,965</td>
<td>56,944</td>
<td>95,544</td>
<td>2,708</td>
<td>3,327</td>
</tr>
<tr>
<td>Edges</td>
<td>11,606,919</td>
<td>818,716</td>
<td>963,468</td>
<td>5,429</td>
<td>4,732</td>
</tr>
<tr>
<td>Features</td>
<td>602</td>
<td>50</td>
<td>120</td>
<td>1,433</td>
<td>3,703</td>
</tr>
<tr>
<td>Classes</td>
<td>41</td>
<td>121</td>
<td>2</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>Training Nodes</td>
<td>152,410</td>
<td>44,906</td>
<td>47,772</td>
<td>140</td>
<td>120</td>
</tr>
<tr>
<td>Validation Nodes</td>
<td>23,699</td>
<td>6,514</td>
<td>9,554</td>
<td>500</td>
<td>500</td>
</tr>
<tr>
<td>Testing Nodes</td>
<td>55,334</td>
<td>5,524</td>
<td>38,218</td>
<td>1,000</td>
<td>1,000</td>
</tr>
</tbody>
</table>
Table 2: Node classification performance

<table>
<thead>
<tr>
<th>Sparsifier</th>
<th>Method</th>
<th>Reddit Micro-F1</th>
<th>PPI Micro-F1</th>
<th>Transaction AUC</th>
<th>Cora Accuracy</th>
<th>Citeseer Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/A</td>
<td>GCN</td>
<td>0.922 ± 0.041</td>
<td>0.532 ± 0.024</td>
<td>0.564 ± 0.018</td>
<td>0.810 ± 0.027</td>
<td>0.694 ± 0.020</td>
</tr>
<tr>
<td></td>
<td>GraphSAGE</td>
<td>0.938 ± 0.029</td>
<td>0.600 ± 0.027</td>
<td>0.574 ± 0.029</td>
<td>0.825 ± 0.033</td>
<td>0.710 ± 0.020</td>
</tr>
<tr>
<td></td>
<td>GAT</td>
<td>-</td>
<td>0.917 ± 0.030</td>
<td>0.616 ± 0.022</td>
<td>0.821 ± 0.043</td>
<td>0.721 ± 0.037</td>
</tr>
<tr>
<td></td>
<td>GIN</td>
<td>0.928 ± 0.022</td>
<td>0.703 ± 0.028</td>
<td>0.607 ± 0.031</td>
<td>0.816 ± 0.020</td>
<td>0.709 ± 0.037</td>
</tr>
<tr>
<td>SS/RD*</td>
<td>GCN</td>
<td>0.912 ± 0.022</td>
<td>0.521 ± 0.024</td>
<td>0.562 ± 0.035</td>
<td>0.780 ± 0.045</td>
<td>0.684 ± 0.033</td>
</tr>
<tr>
<td></td>
<td>GraphSAGE</td>
<td>0.907 ± 0.018</td>
<td>0.576 ± 0.022</td>
<td>0.565 ± 0.042</td>
<td>0.806 ± 0.032</td>
<td>0.701 ± 0.027</td>
</tr>
<tr>
<td></td>
<td>GAT</td>
<td>-</td>
<td>0.889 ± 0.034</td>
<td>0.614 ± 0.044</td>
<td>0.807 ± 0.047</td>
<td>0.686 ± 0.034</td>
</tr>
<tr>
<td></td>
<td>GIN</td>
<td>0.901 ± 0.021</td>
<td>0.693 ± 0.019</td>
<td>0.593 ± 0.038</td>
<td>0.785 ± 0.041</td>
<td>0.706 ± 0.043</td>
</tr>
<tr>
<td>Neural Sparse</td>
<td>GCN</td>
<td>0.946 ± 0.020</td>
<td>0.600 ± 0.014</td>
<td>0.610 ± 0.022</td>
<td>0.821 ± 0.014</td>
<td>0.715 ± 0.014</td>
</tr>
<tr>
<td></td>
<td>GraphSAGE</td>
<td>0.951 ± 0.015</td>
<td>0.626 ± 0.023</td>
<td>0.649 ± 0.018</td>
<td>0.832 ± 0.024</td>
<td>0.720 ± 0.013</td>
</tr>
<tr>
<td></td>
<td>GAT</td>
<td>-</td>
<td>0.921 ± 0.015</td>
<td>0.671 ± 0.018</td>
<td>0.834 ± 0.015</td>
<td>0.724 ± 0.026</td>
</tr>
<tr>
<td></td>
<td>GIN</td>
<td>0.937 ± 0.027</td>
<td>0.744 ± 0.015</td>
<td>0.634 ± 0.023</td>
<td>0.824 ± 0.027</td>
<td>0.719 ± 0.015</td>
</tr>
</tbody>
</table>

(* Report the better performance with SS or RD)

Inductive datasets. We utilize the Reddit and PPI datasets and follow the same setting in Hamilton et al. (2017). The Reddit dataset contains post-to-post graph with word vectors as node features. The node labels represent which community Reddit posts belong to. The protein-protein interaction (PPI) dataset contains graphs corresponding to different human tissues. The node features are positional gene sets, motif gene sets and immunological signatures. The nodes are multi-labeled by gene ontology. The graph in the Transaction dataset contains real transactions between organizations in two years, with the first year for training and the second year for validation/testing. Each node represents an organization and each edge indicates a transaction between two organizations. Node attributes are side information about the organizations such as account balance, cash reserve, etc. On this dataset, we aim to classify organizations into two categories: promising or others for investment in near future. The class distribution in the Transaction dataset is highly imbalanced. During the training under inductive setting, algorithms have only access to training nodes’ attributes and edges. In the PPI and Transaction datasets, the models have to generalize to completely unseen graphs.

Transductive datasets. We use two citation benchmark datasets with transductive experimental setting in Yang et al. (2016); Kipf & Welling (2017). The citation graphs contain nodes corresponding to documents and edges as citations. Node features are the sparse bag-of-words representations of documents and node labels indicate the topic class of the documents. In transductive learning, the training methods have access to all node features and edges, with a limited subset of node labels.

5.2 Experimental Setup

Baseline models. We incorporate four state-of-the-art methods as the base GNN components, including GCN (Kipf & Welling, 2017), GraphSAGE (Hamilton et al., 2017), GAT (Veličković et al., 2018), and GIN (Xu et al., 2019). We evaluate our proposed NeuralSparse with sparsification network and each of the four GNNs. Besides, we also implement variants of NeuralSparse by replacing the sparsification network with either the spectral sparsifier (SS, Sadhanala et al., 2016) or the Rank Degree (RD, Voudigari et al., 2016) method.

Temperature tuning. We anneal the temperature with the schedule \( \tau = \max(0.05, \exp(-rp)) \), where \( p \) is the training epoch and \( r \in 10^{(-5, -4, -3, -2, -1)} \). \( \tau \) is updated every \( N \) steps and \( N \in \{50, 100, ..., 500\} \). Compared with MNIST VAE model in Jang et al. (2017), smaller hyper-parameter \( \tau \) fits NeuralSparse better in practice.

Metrics. We evaluate the performance on the transductive datasets with accuracy (Kipf & Welling, 2017). For inductive tasks on the Reddit and PPI datasets, we report micro-averaged F1 scores (Hamilton et al., 2017). Due to the highly imbalanced classes in the Transaction dataset, models are evaluated with AUC value (Huang & Ling, 2005). The results show the average of 10 runs.

5.3 Classification Performance

Table 2 summarizes classification performance of NeuralSparse and the baseline methods on all datasets. For Reddit, PPI, Transaction, Cora and Citeseer, the hyper-parameter \( k \) is set as 30, 15, 10,
5, and 3 respectively. The hyper-parameter \( l \) is set as 1 in this experiment. Note that the result of GAT on Reddit is missing due to the out-of-memory error.

Overall, NeuralSparse outperforms all baseline methods on all datasets. We make the following observations. (1) Compared with basic GNN models, NeuralSparse can enhance the generalization performance on node classification tasks by utilizing the sparsified subgraphs from sparsification network, especially in the inductive setting. Indeed, large neighborhood size in the original graphs significantly increases the complexity of the convolutional operations, which brings higher risk of overfitting. (2) With different GNN models, the NeuralSparse can consistently achieve superior performance, which demonstrates NeuralSparse is general and can be applied to multiple classification models. (3) In comparison with the two NeuralSparse variants, Spectral Sparsifier and Rank Degree, NeuralSparse outperforms because of the automatically learned graph sparsification with both structural and non-structural information as input.

5.4 Sensitivity to Hyper-parameters and Sparsified Subgraphs

Figure 3(c) demonstrates how classification performance responds when \( k \) increases on the Transaction dataset. There exists an optimal \( k \) that delivers the best classification AUC score. When \( k \) is small, NeuralSparse can only make use of little structural information in feature aggregation, which leads to suboptimal performance. When \( k \) increases, the aggregation convolution involves more complex neighborhood aggregation with higher overfitting risk, which negatively impacts the classification performance for unseen testing data. Figure 3(d) shows how hyper-parameter \( l \) impacts classification performance on the Transaction dataset. When \( l \) increases from 1 to 5, we observe a relatively small improvement in classification AUC score. As the parameters in the sparsification network are shared by all edges in the graph, the estimation variance from random sampling could already be mitigated to some extent by a number of sampled edges in a sparsified subgraph. Thus, when we increase the number of sparsified subgraphs, the incremental gain could be small.

In Figure 3(a, b), we present the sparsified graphs output by two baseline methods, SS and RD. By comparing the two plots with Figure 1(b), we make the following observations. First, the NeuralSparse sparsified graph tends to select edges that connect nodes of identical labels, which favors the downstream classification task. The observed clustering effect could further boost the confidence of decision making. Second, instead of exploring all the neighbors, we can focus on selected connections/edges in sparsified graphs, which could make it easier for human experts to perform model interpretation and result visualization.

6 Conclusion

In this paper, we propose Neural Sparsefication (NeuralSparse) to address the overfitting issues brought by the complexity in real-life large graphs. NeuralSparse consists of two major components: (1) The sparsification network sparsifies input graphs by sampling edges following a learned distribution; (2) GNNs take sparsified subgraphs as input and extracts node representations for downstream tasks. The two components in NeuralSparse can be jointly trained with supervised loss, gradient descent, and backpropagation techniques. The experimental study on real-life datasets show that the NeuralSparse consistently renders more robust graph representations, and brings up to 7% improvement in accuracy over the state-of-the-art GNN models.
REFERENCES


Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *WWW*, 2015.


Yutao Zhang, Fanjin Zhang, Peiran Yao, and Jie Tang. Name disambiguation in aminer: Clustering, maintenance, and human in the loop. In KDD, 2018c.

Peixiang Zhao. gsparsify: Graph motif based sparsification for graph clustering. In CIKM, 2015.

S1 Dataset Details

In this section, we provide additional details about the Transaction datasets in our experiments. The Transaction dataset contains attributed graph that records transaction history between organizations in two years: 2014 and 2015. Each node represents an organization and each directed edge indicates a transaction between two organizations. Node attributes include organization information like account balance, cash research, etc. Under the inductive experimental setting, We use the 47,772 organization data of the year 2014 for training and remaining data are hidden from the model. The 9,554 organizations are used for validation and 38,218 for testing. Validation and testing node sets are from year 2015 and are not connected to the nodes in the training set. Like the PPI dataset, models need to generalize to unseen graph when testing on the Transaction dataset.

S2 Experimental Settings

In this section, we provide more details about our implementation and experiments in favor of reproducibility.

S2.1 Hardware

All experiment are run on a Linux machine with 16 Intel(R) Xeon(R) CPU (E5-2637 v4 @ 3.50GHz) and 128GB RAM. Some models (e.g. NeuralSparse() and GCN) are accelerated by 4 NVIDIA GeForce GTX1080Ti GPU with 11GB RAM.

S2.2 Implementations of NeuralSparse

We implement the proposed NeuralSparse in tensorflow framework for efficient GPU computation. In particular, the multi-layer neural network (Equation 4) in the sparsification network is implemented by two-layer feed-forward neural networks in all experiment, where the hyper-parameter $k$ is searched between 2 and 50 for the optimal performance. We employ cross-entropy to formulate the loss function and apply Adam optimizer for training. The learning rate of Adam optimizer is initially set to be $\alpha = 1.0 \times 10^{-3}$. We initial the weight matrices in the proposed NeuralSparse() model with Xavier initialization.

In the following, we detail the network structures of NeuralSparse() used on individual datasets. FC($a$, $b$, $f$) means a fully-connected layer with $a$ input neurons and $b$ output neurons activated by function $f$ (none means no activation function is used). GNN($a$, $b$, $f$) means a Graph Neural Network layer with input dimension $a$, output dimension $b$, and activation function $f$. We implement GNN layer with GCN, GraphSAGE, GAT, GIN in the experiments.

**Reddit** The sparsification network runs with: FC(602, 16, ReLU)-FC(16, 1, Gumbel-Softmax). The structure of GNN is GNN(602, 128, ReLU)-GNN(128, 64, ReLU)-FC(64, 41, softmax).

**PPI** The sparsification network runs with: FC(50, 16, ReLU)-FC(16, 1, Gumbel-Softmax). The structure of GNN is GNN(50, 128, ReLU)-GNN(128, 64, ReLU)-FC(64, 41, softmax).

**Transaction** The sparsification network runs with: FC(121, 16, ReLU)-FC(16, 1, Gumbel-Softmax). The structure of GNN is GNN(121, 128, ReLU)-GNN(128, 32, ReLU)-FC(32, 2, softmax). Note that there is one-dimensional edge attribute indicating the transaction amount in this dataset.

**Cora** The sparsification network runs with: FC(1433, 32, ReLU)-FC(32, 1, Gumbel-Softmax). The structure of GNN is GNN(1433, 128, ReLU)-GNN(128, 64, ReLU)-FC(64, 7, softmax).

**Citeseer** The sparsification network runs with: FC(3703, 64, ReLU)-FC(64, 1, Gumbel-Softmax). The structure of GNN is GNN(3703, 128, ReLU)-GNN(128, 64, ReLU)-FC(64, 6, softmax).

As the spectral sparsification models cannot be jointly trained with subsequent GNN module, the sparsification process is treated as a preprocessing step. For Spectral Sparsifier (SS), $\epsilon$ is set to 0.4 in all datasets. For the Rank Degree algorithm (RD), we select 1% of nodes as the initial seeds and adopt $\rho \in \{0.1, 0.2, \cdots, 0.8\}$ for the best results.
Table S1: Percentage of edges connecting nodes of the same labels

<table>
<thead>
<tr>
<th></th>
<th>Reddit</th>
<th>PPI</th>
<th>Transaction</th>
<th>Cora</th>
<th>Citeseer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>53.1%</td>
<td>55.0%</td>
<td>67.3%</td>
<td>82.2%</td>
<td>73.1%</td>
</tr>
<tr>
<td>SS</td>
<td>50.9%</td>
<td>52.8%</td>
<td>62.8%</td>
<td>79.8%</td>
<td>75.6%</td>
</tr>
<tr>
<td>RD</td>
<td>49.8%</td>
<td>53.5%</td>
<td>63.4%</td>
<td>84.8%</td>
<td>72.3%</td>
</tr>
<tr>
<td>NeuralSparse</td>
<td>59.6%</td>
<td>61.5%</td>
<td>76.8%</td>
<td>93.1%</td>
<td>87.4%</td>
</tr>
</tbody>
</table>

S3 Qualitative Edge Sampling Evaluation

In this subsection, we qualitatively demonstrate the difference by Figure 1(a) original graph, Figure 1(b) NeuralSparse, Figure 3(a) SS, and Figure 3(b) RD. In addition, we provide quantitative analysis in Table S1, where we report the percentage of edges that connect nodes of same class labels in sparsified graphs. Both qualitative and quantitative results suggest a common trend: NeuralSparse prefers to select neighbors with the same labels compared with the baseline methods.