S1. More Details on Transaction Dataset

In this section, we provide additional details about the Transaction datasets in our experiments. The Transaction dataset contains an 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 the year 2015 and are not connected to the nodes in the training set. Like the PPI dataset, models need to generalize to the 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 experiments 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 the GNN layer with GCN, GraphSAGE, GAT, GIN in the experiments.

Reddit.[1] The sparsification network runs with: FC(1204, 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.[1] The sparsification network runs with: FC(100, 16, ReLU)-FC(16, 1, Gumbel-Softmax). The structure of GNN is GNN(50, 128, ReLU)-GNN(128, 128, ReLU)-FC(128, 121, softmax).

Transaction. The sparsification network runs with: FC(243, 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.[2] The sparsification network runs with: FC(2866, 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.[2] The sparsification network runs with: FC(7406, 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), ϵ 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.

S3. Experiment with Similar Numbers of Trainable Parameters

In this section, we evaluate the impact brought by reducing the number of parameters in a GNN with NeuralSparse so that the numbers of trainable parameters in a NeuralSparse GNN and an original GNN are similar. In particular, we focus on GCN in this set of experiments. Using the same notation in S2.2, NeuralSparse-GCN-Compact is implemented as follows.

Reddit. NeuralSparse-GCN-Compact runs with: FC(1204, 8, ReLU)-FC(8, 1, Gumbel-Softmax) and GCN(602, 112, ReLU)-GCN(112, 64, ReLU)-FC(64, 41, softmax). The total number of trainable parameters is 86,856 in the NeuralSparse-GCN-Compact, while it is 87,872 in the original GCN.

PPI. NeuralSparse-GCN-Compact runs with: FC(100, 16, ReLU)-FC(16, 1, Gumbel-Softmax) and GCN(50, 118,

[1] http://snap.stanford.edu/graphsage/
[2] https://github.com/tkipf/gcn

Table S1. Node classification performance with similar numbers of trainable parameters

Dataset	**Reddit**	**PPI**	**Transaction**	**Cora**	**Citeseer**
Metrics	Micro-F1	Micro-F1	AUC	Accuracy	Accuracy
GCN	0.922 ± 0.041	0.532 ± 0.024	0.564 ± 0.018	0.810 ± 0.027	0.694 ± 0.020
NeuralSparse-GCN	0.946 ± 0.020	0.600 ± 0.014	0.610 ± 0.022	0.821 ± 0.014	0.715 ± 0.014
NeuralSparse-GCN-Compact	0.943 ± 0.018	0.601 ± 0.021	0.605 ± 0.013	0.820 ± 0.012	0.713 ± 0.009

ReLU)-GCN(118, 128, ReLU)-FC(128, 121, softmax). The total number of trainable parameters is 38,108 in the NeuralSparse-GCN-Compact, while it is 38,272 in the original GCN.

Transaction. NeuralSparse-GCN-Compact runs with: FC(243, 16, ReLU)-FC(16, 1, Gumbel-Softmax) and GCN(121, 100, ReLU)-GCN(100, 32, ReLU)-FC(32, 2, softmax). The total number of trainable parameters is 19,268 in the NeuralSparse-GCN-Compact, while it is 19,648 in the original GCN.

Cora. NeuralSparse-GCN-Compact runs with: FC(2866, 8, ReLU)-FC(8, 1, Gumbel-Softmax) and GCN(1433, 115, ReLU)-GCN(115, 32, ReLU)-FC(32, 7, softmax). The total number of trainable parameters is 191,635 in the NeuralSparse-GCN-Compact, while it is 192,064 in the original GCN.

Citeseer. NeuralSparse-GCN-Compact runs with: FC(7406, 32, ReLU)-FC(32, 1, Gumbel-Softmax) and GCN(3703, 64, ReLU)-GCN(64, 32, ReLU)-FC(32, 6, softmax). The total number of trainable parameters is 476,256 in the NeuralSparse-GCN-Compact, while it is 482,560 in the original GCN.

From the evaluation results shown in Table S1, we draw the following observations. First, both NeuralSparse-GCN and NeuralSparse-GCN-Compact consistently outperform GCN on all the datasets. Second, compared with NeuralSparse-GCN, NeuralSparse-GCN-Compact achieves comparable prediction accuracy with smaller variance in most cases.

S4. Convergence Analysis

We analyze the convergence properties of NeuralSparse and DropEdge on Citeseer. The results, as shown in Figure S1, demonstrate that NeuralSparse converges faster and achieves better performance than DropEdge.

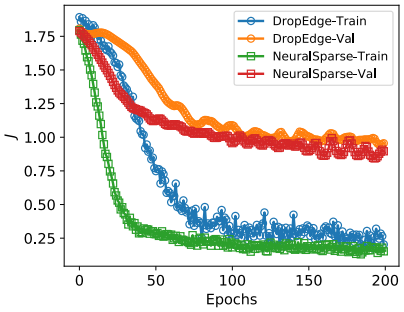
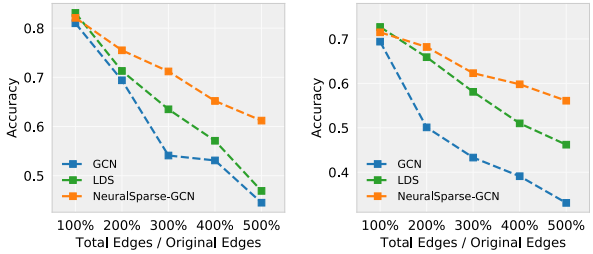


Figure S1. Convergence analysis



(a) Cora (b) Citeseer

Figure S2. Node classification performance when adding noise to graph structure.

S5. More Empirical Comparison between NeuralSparse and LDS

S5.1. Random Edge Addition to Cora and Citeseer

We further compare NeuralSparse and LDS (Franceschi et al., 2019) on the node classification tasks where original graph structure is available but more random edges are introduced as noise. Starting from the original graphs, we add edges by randomly sampling two nodes u, v from node set $\mathbb{V}$ and connecting them.

The results are shown in Figure S2. In both datasets, NeuralSparse achieves better performance compared with LDS as the noise level goes beyond 200%. When the amount of noise increases, the classification accuracy of LDS drops

Table S2. Percentage of edges connecting nodes of the same labels

	Reddit	**PPI**	**Transaction**	**Cora**	**Citeseer**
Original	53.1%	55.0%	67.3%	82.2%	73.1%
SS	50.9%	52.8%	62.8%	79.8%	75.6%
RD	49.8%	53.5%	63.4%	84.8%	72.3%
NeuralSparse	59.6%	61.5%	76.8%	93.1%	87.4%

significantly. This result confirms our conjecture that NeuralSparse is more robust to random edges, compared to LDS.

S5.2. Complete Graphs

We compare the NeuralSparse and LDS in the case of complete graphs suggested in (Franceschi et al., 2019). As shown in Table S3, we observe that NeuralSparse consistently performs better.

Table S3. Node classification performance with complete graphs

	Cora	Citeseer
GCN	0.580 ± 0.037	0.493 ± 0.026
LDS	0.684 ± 0.029	0.656 ± 0.039
NeuralSparse-GCN	0.691 ± 0.016	0.679 ± 0.033

S6. Validation Performance as hyper-parameter k Changes

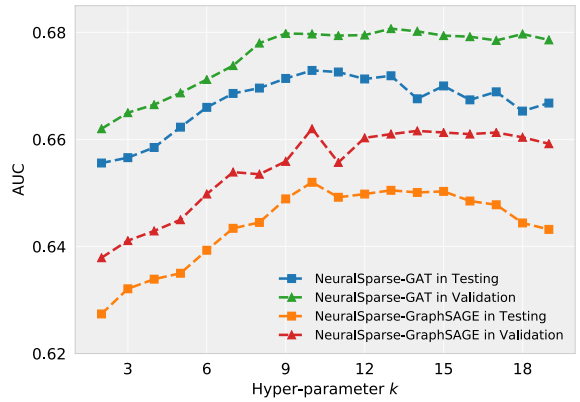


Figure S3. Impact from hyper-parameter k on validation and testing on the Transaction dataset

In this section, we demonstrate how the hyper-parameter k impacts the performance of NeuralSparse-GAT and NeuralSparse-GraphSAGE in both validation and testing on the Transaction dataset. In terms of validation, as shown in Figure S3, the validation performance increases when k ranges from 2 to 10 with more available graph data. After k exceeds 10, the increase in validation performance slows down and turns to be saturated. In terms of testing performance, it shares a similar trend when k ranges from 2 to 10. Meanwhile, the testing performance drops more after k exceeds 10.

S7. Quantitative Edge Sampling Evaluation

In Section 5.4, we qualitatively demonstrate the difference by Figure 3(a) original graph, (b) NeuralSparse, (c) SS, and (d) RD. In addition, we provide quantitative analysis in Table S2, 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.