GLAM: GRAPH LEARNING BY MODELING AFFINITY TO LABELED NODES FOR GRAPH NEURAL NETWORKS

ABSTRACT

We propose GLAM, a semi-supervised graph learning method for cases when there are no graphs available. This approach learns a graph as a convex combination of the unsupervised *k*-Nearest Neighbor graph and a supervised label-affinity graph. The latter graph directly captures all the nodes' label-affinity with the labeled nodes, i.e., how likely a node has the same label as the labeled nodes. Our experiments show that GLAM gives close to or better performance (up to $\sim$1.5%) while being simpler and faster (up to 70x) to train than state-of-the-art graph learning methods. We also demonstrate the importance of individual components and contrast them with the state-of-the-art methods.

1 INTRODUCTION AND RELATED WORK

Incorporating graph to improve semi-supervised classification is a long and well-studied problem. Early methods used graph to propagate labels (Zhu & Ghahramani, 2002; Zhou et al., 2004). Classifiers later utilized graphs for regularization (Belkin et al., 2006; Weston et al., 2008). Graph Neural Networks (GNNs), which incorporate graphs in their architecture, was introduced recently. Graph Convolutional Network (GCN) is the most prominent among them (Kipf & Welling, 2017; Wu et al., 2019). GNN models have shown performance exceeding approaches that only use the graph to propagate labels or to regularize. However, all these approaches assume the existence of a graph. In this paper, we are mainly interested in scenarios where we do not have access to a graph. In the absence of a graph, utilizing the *k*-Nearest Neighbor (*k*NN) graph has been proven to be quite effective (Belkin et al., 2006; Gidaris & Komodakis, 2019) and gives improvements over models that do not use graph information.

Recent literature, including LDS (Franceschi et al., 2019) and IDGL (Chen et al., 2020), propose graph learning approaches beyond simple *k*NN construction. LDS treats a graph as a hyperparameter and employs a bi-level optimization setup. IDGL learns a graph by iteratively learning latent node representations and construct adjacency structures from these representations. PG-Learn (Wu et al., 2018), TO-GCN (Yang et al., 2019), and GCRN (Yu et al., 2020) explored similar ideas to LDS and IDGL. Tangential to these works, the Graph Agreement Model (GAM) (Stretcu et al., 2019) proposed an agreement model to regularize the classifier. Bojchevski et al. (2018); Trivedi et al. (2020) explored generative models for graphs, but these approaches do not cater to graph learning.

Existing graph learning approaches are iterative and computationally expensive. An alternative way to model edges is by learning attention over edges using models like GAT (Veličković et al., 2018) and SuperGAT (Kim & Oh, 2021). However, learning attention across all the edges with limited labels is difficult. To address these shortcomings, we propose GLAM, a graph learning approach. GLAM combines a supervised graph with an unsupervised kNN graph. We introduce a novel label-affinity model, which uses an explicit loss function to learn the supervised graph. Our experiments show that GLAM gives close to or better performance (up to $\sim$1.5%) while being simpler and faster (up to 70x) to train than state-of-the-art graph learning methods.

2 PROBLEM STATEMENT

In the semi-supervised classification problem, we have labeled examples $D = \{(x_i, y_i)\}_{i=1}^{l}$ and unlabeled examples $U = \{x_i\}_{i=l+1}^{l+u}$, where $x_i \in X$. Let Y be the set of all possible labels and $\mathbf{y}$

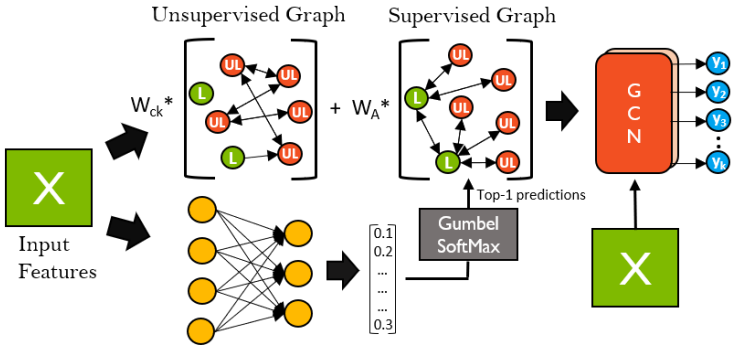


Figure 1: Schematic representation of our GLAM

be the vector $[y_1, y_2, \ldots, y_{l+u}]$. The goal in traditional semi-supervised classification problem is to learn a function $f : X \mapsto Y$ and it is often learnt by solving the following minimization problem,

$$f^* = \arg\min_f \mathcal{L}(f(X), \mathbf{y}) + \alpha\|f\| \tag{1}$$

where $\mathcal{L}$ is the loss function, and α is the regularization coefficient.

In Graph Neural Networks (GNNs), the classifier function additionally assumes access to a graph between the data points in X. Let this graph be G. Then, this classifier function is often estimated by solving,

$$f^* = \arg\min_f \mathcal{L}(f(X, G), \mathbf{y}) + \alpha\|f\| \tag{2}$$

But, often in practice, we may not have access to any graphs. The goal of our work here is to estimate a classifier function by solving,

$$f^*, G^* = \arg\min_{f,G} \mathcal{L}(f(X, G), \mathbf{y}) + \alpha\|f\| + \beta\Omega(G, \mathbf{y}) \tag{3}$$

where Ω is additional loss terms which could depend on G and the known labels $\mathbf{y}$ and β is its coefficient.

3 Proposed Approach

The proposed model consists of three basic components: Label-Affinity Model, Convex Combination of Graphs and GCN. GLAM's architecture is presented in Figure 1

Label-Affinity Model: Attention-based models attempt to learn attention over all the edges of a graph. This could be challenging when working with limited labeled data. The critical insight here is to realise that having a few noisy edges can hurt more than throwing away several good edges (i.e. edges where the source and target nodes have same labels). To illustrate this, we remove all the noisy edges from the *k*NN graph and perform two experiments where we vary the percentage of noisy/good edges added/removed, and report GCN performance on these graphs. We observe that adding noisy edges hurts performance more than removing several good edges. A detailed analysis of this experiment is presented in the Appendix A.4. Based our analysis, we make an assumption that more confident predictions can be made on edges from any node to labeled node, instead of any arbitrary edge, because of the partial information available. Conventional attention-based models attempt to learn attention as a function $X \times X \mapsto \mathbb{R}$. However, we decided to change the model to $X \mapsto \Delta^{|D|}$ where X is the set of instances, Δ is the probability simplex: $\{\sum_{i=1}^{|D|} \theta_i = 1$ and $\forall_i \theta_i \geq 0\}$. D is the set of labeled data as indicated in Section 2. This form of modeling was a straight-forward way of enforcing our restriction. This model predicts a distribution over the labeled set. The probability value indicates how likely it is for the target labeled node to have the same label as the input node. We model this with a simple two-layer neural network as follows:

$$Z^A = \textsc{SoftMax}(\sigma_1(XW_1)W_2) \tag{4}$$

where $\mathbf{X} \in \mathbb{R}^{(l+u)\times d}$ is the feature matrix, d is the number of features, σ_1 is a non-linear activation function and W_1, W_2 are model weights. There are two advantages to this model - 1] it is a simple way of enforcing restriction on edges that we desire, 2] it also allows us to write an explicit loss function using the labeled data (we will discuss this in more detail below).

The label-affinity model can be used to construct the affinity graph. To construct a graph, for every node, we compute its predicted distribution over the labeled set and add the edge with the highest probability. This edge is added in both directions. We can write this mathematically as,

$$G_A = \textsc{One-Hot}(\textsc{Arg-Max}(Z^A)) \tag{5}$$

$$G_A = [G_A : \mathbf{0}^{(l+u)\times u}] \tag{6}$$

$$G_A = G_A + G_A^\top \tag{7}$$

In Equation 5, we get the highest affinity labeled node for all the nodes. In Equation 6, we augment the remaining unlabeled columns which are simply zeroes to complete the full graph. In Equation 7, we make the graph symmetric. However, since ARG-MAX is not differentiable, we use the Gumbel-Softmax trick Jang et al. (2017) and generate samples from Gumbel-Softmax distribution instead to create the graph. Relying on the classifier loss to learn the parameters would be difficult as the gradients that flow into this model are minuscule. Thereby, we add explicit loss for this model by constructing labeled data from D. For every $x_i \in D$, we create a new label vector y_i^A such that,

$$y_i^A[j] = \begin{cases} 1 & i \neq j, \;\; y_i = y_j \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

where $y_i^A[j]$ is the j^{th} element of the vector. Further, it is normalized as $\widehat{y_i^A} = y_i^A / \mathbf{1}^\top y_i^A$. We use this to define the loss for the affinity model as follows:

$$\mathcal{L}_A = \sum_{i=1}^{l} \textsc{Cross-Entropy}(Z_i^A, \widehat{y_i^A}) \tag{9}$$

where Z_i^A is the row corresponding to the i^{th} point in X.

Combining Graphs and GCN: The constructed affinity graph, G_A, contains only a few high confident edges and restricts feature propagation, particularly between unlabeled nodes. Thereby, we utilize the *k*NN graph to assist in this regard. The *k*NN graph is constructed from input features. We use cosine distance metric in our experiments. Number of neighbors, *k*, is a hyper-parameter. We pre-process the *k*NN graph by cropping all the incoming edges to the labeled nodes to reduce noise among labeled nodes. We recognize that there is noise in other edges of the graph, but it is difficult to filter this noise. However, removing incoming noise to training nodes gives improvements as we show in Table 3 (a). We refer to this graph as G_{ck}, where ck stands for cropped *k*NN graph. This graph is then combined with the affinity graph as follows:

$$G = w_A G_A + w_{ck} G_{ck} \text{ s.t. } w_A + w_{ck} = 1 \tag{10}$$

This combined graph is finally fed to a two-layer GCN model. However, because of the cropped *k*NN graph, this combined graph is asymmetric. For computing the Laplacian for GCN, we can use either the indegree or the outdegree matrix. In our experiments, we have utilized the indegree matrix. The combined graph is finally fed to a two-layer GCN model and the final objective we minimize is,

$$\mathcal{L}_{final} = \mathcal{L}_C + \beta \mathcal{L}_A + \alpha_A \sum_i \|W_a^i\| + \alpha_C \sum_i \|W_{gcn}^i\| \tag{11}$$

where α_A, α_C are regularization coefficients, W_a^i and W_{gcn}^i are affinity model and GCN model weights.

The first term in Equation 11 refers to the classifier loss, which takes the combination of unsupervised *k*NN and supervised affinity graph (Equation 10). The second term is the affinity loss (Equation 9) which models label affinity from all nodes to labeled nodes. The affinity loss together with classifier loss improve affinity predictions. Improvement in affinity predictions improves classifier performance.

	Features					Boosted Features				
Models\Datasets	**Cora**	**CiteSeer**	**Pubmed**	**ACM**	**DBLP**	**Cora**	**CiteSeer**	**Pubmed**	**ACM**	**DBLP**
LogReg	56.20	61.90	73.60	78.80	66.90	70.20	69.90	76.20	88.30	80.20
MLP	59.18 (1.39)	62.02 (1.80)	72.92 (0.55)	79.92 (0.55)	67.00 (0.53)	70.34 (0.56)	68.50 (1.52)	74.78 (0.89)	87.66 (0.69)	78.96 (0.66)
LP	54.20	57.40	63.40	75.20	66.30	59.20	54.20	67.70	89.00	65.20
ManiReg	58.80	61.10	69.20	80.90	72.60	53.50	55.30	68.10	90.00	73.40
SemiReg	66.24 (1.13)	66.44 (0.92)	73.78 (0.69)	88.76 (1.21)	74.26 (1.21)	69.13 (0.55)	70.09 (0.10)	74.04 (0.35)	88.88 (0.33)	79.54 (0.34)
GCN	68.32 (1.37)	68.76 (1.04)	70.12 (1.05)	86.38 (0.53)	77.62 (0.95)	70.20 (0.81)	69.36 (0.67)	75.24 (0.39)	90.84 (0.39)	80.48 (0.34)
GAT	68.50 (0.99)	70.00 (0.76)	70.06 (1.83)	86.36 (0.26)	77.58 (0.81)	70.86 (0.63)	69.10 (1.35)	75.18 (0.39)	90.92 (0.34)	80.28 (0.49)
SuperGAT	69.23 (0.31)	69.36 (0.58)	70.88 (0.68)	86.27 (0.31)	77.44 (0.44)	69.90 (1.67)	69.70 (1.29)	75.58 (0.25)	90.58 (0.93)	78.38 (0.95)
LDS	**70.76 (0.78)**	72.16 (0.61)	OOM	86.98 (0.78)	76.72 (0.57)	**72.87 (0.45)**	71.44 (0.40)	OOM	**92.93 (0.61)**	79.42 (0.84)
IDGL	70.32 (0.54)	67.65 (1.71)	**77.20 (0.76)**	88.94 (0.52)	74.26 (0.84)	71.90 (0.69)	68.88 (0.44)	**79.04 (0.26)**	91.09 (0.38)	79.66 (0.47)
GLAM	70.58 (0.18)	**72.22 (0.45)**	74.03 (0.32)	**89.34 (0.15)**	**79.70 (0.32)**	72.64 (0.35)	**71.86 (0.44)**	76.06 (0.35)	92.38 (0.20)	**81.52 (0.30)**

Table 1: Mean test accuracy over 5 random seeds for benchmark datasets. Standard deviation is reported in brackets where ever applicable.

Note on Input Features: Note that, for the construction of the *k*NN graphs in previous section the input to the affinity model in Equation 4, we have used $\mathbf{X}$ (the feature matrix). However, in a recent work on text classification, the authors of HeteGCN Ragesh et al. (2021) have proposed an architecture which utilizes a normalized version of $\mathbf{X}^\top\mathbf{X}$, call it $\widetilde{\mathbf{X}^\top\mathbf{X}}$ as a first layer of GCN. Essentially they are utilizing feature correlation as the second layer graph. The simplified version of this model is simply equivalent to $\widetilde{\mathbf{X}\mathbf{X}^\top\mathbf{X}}$. We believe that the HeteGCN model benefits from this feature correlation matrix and that utilizing $\widetilde{\mathbf{X}\mathbf{X}^\top\mathbf{X}}$ instead of $\mathbf{X}$ should benefit our graph learning problem as well. We simply refer to them as boosted features. We use these boosted features in *k*NN construction and as input to affinity model. We show experimental results with both the normal and the boosted features and not for just our model but for all the baselines, to illustrate how these simple modified features can give significant performance improvements.

4 EXPERIMENTS

We conducted several experiments to highlight the effectiveness of GLAM against several baselines and state-of-the-art graph learning methods (refer to appendix A.2 for details). We discovered a label-leakage bug in GAM's source code, which post fixing shows marginal improvements over baselines. A comparison against bug-fixed version of GAM is reported in Appendix A.3. We restrict our setting to semi-supervised node classification problems where a graph structure is not available. Our experiments also present empirical analysis to illustrate why our model performs better. Additional experiments are reported in Appendix A.3.

Performance Improvement and Speed up. We report the mean test accuracy and standard deviation over five random seeds for benchmark datasets (refer to appendix A.1 for details). GLAM shows competitive performance across all benchmark datasets, except PubMed, with accuracy gains of up to 3%. We further improve all the baselines by feeding them with boosted features either as input features or by using them for constructing *k*NN graphs. Our results in Table 1 show that these new features, in most cases, considerably improve baseline models with accuracy gains of up to 14%. We perform a timing comparison by computing the average end-to-end training time over 2000 runs. Table 3 (b) shows timing analysis for different benchmark datasets. We observe that we get up to $\sim 42\times$ speedup compared to IDGL and up to $\sim 70\times$ speed up compared to LDS.

Analysis. What do Attention models capture? We can view GLAM as an instance of an attention model that places sparse attention on edges between all nodes and labeled nodes. SuperGAT (Kim & Oh, 2021) works under the assumption that if two nodes are linked, they are more relevant and suggests that if the homophily of the graph is > 0.2, SuperGAT$_{\text{MX}}$ performs well. However, we observe that GAT performs as well as or better than SuperGAT$_{\text{MX}}$ with *k*NN graphs. We report homophily percentages for *k*NN graphs in Table 4, indicating that there might be more underlying reasons for when attention works or does not. Towards this end, and to quantify the quality of learned attention coefficients, we compute few metrics.

(1) Bad Neighbor Ratio (BNR): We define bad neighbors for a center node as all the neighboring nodes having different labels than the center node. For Attention-based models, we extract attention matrices (A) from the hidden layers for all heads. For GLAM and GCN, we treat Laplacian matrices as attention matrices. We compute BNR as follows:

	Weighted Homophily Scores				Bad Neighbour Ratio			
	Cora	**CiteSeer**	**ACM**	**DBLP**	**Cora**	**CiteSeer**	**ACM**	**DBLP**
GCN	61.29	62.00	86.23	73.61	35.95	36.32	13.32	25.98
GAT	58.72	60.90	**87.21**	**74.52**	50.79 (70.00)	41.33 (67.20)	14.11 (91.2)	26.00 (79.60)
SuperGAT	55.96	54.14	84.97	73.08	38.98 (71.40)	39.16 (69.80)	14.34 (91.1)	27.21 (78.4)
GLAM	**67.01**	**69.22**	85.25	73.80	**26.05**	**26.35**	**07.91**	**21.43**

Table 2: We report Weighted Homophily Scores and Bad Neighbour Ratio (BNR) for different models on several datasets

(a) Ablation Study

	Cora	**CiteSeer**	**ACM**	**DBLP**
GLAM	**72.64 (0.35)**	**71.86 (0.44)**	**92.38 (0.20)**	**81.52 (0.30)**
w/o affinity graph	70.54 (0.76)	70.54 (0.76)	91.46 (0.42)	80.63 (0.85)
w/o affinity loss	70.90 (0.83)	69.06 (1.66)	91.46 (0.48)	80.40 (1.17)

(b) Timing Analysis

Average Time	**Cora**	**CiteSeer**	**PubMed**	**ACM**	**DBLP**
IDGL	152.68s	394.94s	546.81s	329.64s	88.89s
LDS	327.75s	683.43s	NA	338.25s	348.75s
GLAM	**5.32s**	**22.22s**	**23.36s**	**7.76s**	**5.04s**

Table 3: We study the effect of affinity graph and affinity loss in (a). End-to-End training time averaged over multiple runs are shown in (b).

$$\text{BNR} = \frac{1}{|h|} \sum_{h} \frac{\sum_{i=1}^{|V|} (\mathbb{1}_{\text{BW}^{(i)} > \text{GW}^{(i)}})}{\sum_{i=1}^{|V|} (\mathbb{1}_{\text{BW}^{(i)} > 0})} \tag{12}$$

where, $\text{BW}^{(i)} = \sum_{j \in N_{(i)}} A_{ij}^{(h)} \mathbb{1}_{\ell(i) \neq \ell(j)}$, $\text{GW}^{(i)} = \sum_{j \in N_{(i)}} A_{ij}^{(h)} \mathbb{1}_{\ell(i) = \ell(j)}$. In the above equation, the number of attention heads is denoted by h, attention matrix by A, and the total number of nodes by $|V|$. This metric gives a direct insight into how the underlying attention mechanism performs.

(2) Weighted homophily: this is a simple extension of homophily. It is equal to the ratio of attention placed on all good edges to the attention placed on all edges. Table 2 shows these metrics for models in comparison. We find that the Bad Neighbor Ratio correlates well with model performance giving us a perspective on why GLAM performs better.

(3) To study the importance of Label Affinity Graph, we computed the average weight placed on edges in the affinity graph and report it in Table 4. We notice that, in most datasets, $\text{W}_\text{A} > 0.3$ is assigned to affinity edges, implying the importance of them. In the PubMed dataset, GLAM places significant weight on the *k*NN graph, which indicates that when *k*NN graph quality is good, they are preferred over affinity graphs.

Dataset	**Cora**	**CiteSeer**	**PubMed**	**ACM**	**DBLP**
Homophily	58.20	59.46	75.08	87.32	85.12
$\mathbf{W_A}$	0.33	0.49	0.04	0.33	0.30

Table 4: Homophily and average of chosen affinity weights

Ablation Study. Table 3 (a) shows that removing the affinity graph and affinity loss significantly affects the performance, thus indicating their importance. GLAM's performance with only cropped *k*NN (w/o affinity graph in Table 3 (a)) is equivalent to that of GCN with *k*NN (Table 1). Therefore, removing all incoming edges to the training nodes does not affect and sometimes improves the performance. Another observation is that unless there is an explicit loss for the affinity term, we see minuscule improvements over plain *k*NN-GCN model (Table 1).

5 Discussion

In this paper, we proposed a model to jointly learn the graph and classifier for semi-supervised classification tasks where no graphs are available. Our experimental results suggest that our model is fast to train and particularly effective when the unsupervised *k*NN graph is noisy. We analysed and compared with baselines along several dimensions, highlighting their limitations and how our model addresses them. For future work, we want to explore combining the merits of IDGL with our work and see if a single model can work across all possible graphs with different noise levels. Also of great interest would be to combine the proposed idea with some of the graph generative models like NetGAN (Bojchevski et al., 2018) and GraphOpt (Trivedi et al., 2020).

REFERENCES

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. *ArXiv*, abs/1907.10902, 2019.

Mikhail Belkin, Partha Niyogi, and Vikas Sindhwani. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *Journal of Machine Learning Research*, pp. 2399–2434, 2006.

Aleksandar Bojchevski, Oleksandr Shchur, Daniel Zügner, and Stephan Günnemannm. Netgan: Generating graphs via random walks. In *ICML*, 2018.

Paola Cascante-Bonilla, Fuwen Tan, Yanjun Qi, and Vicente Ordonez. Curriculum labeling: Revisiting pseudo-labeling for semi-supervised learning, 2020.

Yu Chen, Lingfei Wu, and Mohammed Zaki. Iterative deep graph learning for graph neural networks: Better and robust node embeddings. *Advances in Neural Information Processing Systems*, 33, 2020.

Luca Franceschi, Mathias Niepert, Massimiliano Pontil, and Xiao He. Learning discrete structures for graph neural networks. In *Proceedings of the 36th International Conference on Machine Learning*, 2019.

Spyros Gidaris and Nikos Komodakis. Generating classification weights with GNN denoising autoencoders for few-shot learning. In *IEEE Conference of Computer Vision and Pattern Recognition*, pp. 21–30, 2019.

Omar Jaafor and Babiga Birregah. Collective classification in social networks. In *Proceedings of the 2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, 2017.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *International Conference on Learning Representations*, 2017.

Dongkwan Kim and Alice Oh. How to find your friendly neighborhood: Graph attention design with self-supervision. In *International Conference on Learning Representations*, 2021.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

Jiaqi Ma, Weijing Tang, Ji Zhu, and Qiaozhu Mei. A flexible generative framework for graph-based semi-supervised learning. In *Advances in Neural Information Processing Systems*, pp. 3276–3285, 2019.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake VanderPlas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Edouard Duchesnay. Scikit-learn: Machine learning in python. *CoRR*, abs/1201.0490, 2012. URL `http://arxiv.org/abs/1201.0490`.

Rahul Ragesh, Sundararajan Sellamanickam, Arun Iyer, Ram Bairi, and Vijay Lingam. Hetegcn: Heterogeneous graph convolutional networks for text classification. In *WSDM*, 2021.

Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *CoRR*, abs/1811.05868, 2018. URL http://arxiv.org/abs/1811.05868.

Otilia Stretcu, Krishnamurthy Viswanathan, Dana Movshovitz-Attias, Emmanouil Platanios, Sujith Ravi, and Andrew Tomkins. Graph agreement models for semi-supervised learning. In *Advances in Neural Information Processing Systems 32*, 2019.

Rakshit Trivedi, Jiachen Yang, and Hongyuan Zha. Graphopt: Learning optimization models of graph formation. In *ICML*, 2020.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. *International Conference on Learning Representations*, 2018.

Vikas Verma, Alex Lamb, Juho Kannala, Yoshua Bengio, and David Lopez-Paz. Interpolation consistency training for semi-supervised learning. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pp. 3635–3641. International Joint Conferences on Artificial Intelligence Organization, 7 2019. doi: 10.24963/ijcai.2019/504. URL https://doi.org/10.24963/ijcai.2019/504.

Xiao Wang, Houye Ji, Chuan Shi, Bai Wang, Yanfang Ye, Peng Cui, and Philip S Yu. Heterogeneous graph attention network. In *The World Wide Web Conference*, 2019.

Jason Weston, Frédéric Ratle, and Ronan Collobert. Deep learning via semi-supervised embedding. In *Proceedings of the 25th International Conference on Machine Learning*, 2008.

Xuan Wu, Lingxiao Zhao, and Leman Akoglu. A quest for structure: Jointly learning the graph structure and semi-supervised classification. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, CIKM '18, pp. 87–96, 2018.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *ArXiv*, abs/1901.00596, 2019.

Liang Yang, Zesheng Kang, Xiaochun Cao, Di Jin, Bo Yang, and Yuanfang Guo. Topology optimization based graph convolutional network. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pp. 4054–4061. International Joint Conferences on Artificial Intelligence Organization, 2019.

Zhilin Yang, William Cohen, and Ruslan Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. In *Proceedings of The 33rd International Conference on Machine Learning*, 2016.

Liang Yao, Chengsheng Mao, and Yuan Luo. Graph convolutional networks for text classification. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019*, pp. 7370–7377. AAAI Press, 2019.

Donghan Yu, Ruohong Zhang, Zhengbao Jiang, Yuexin Wu, and Yiming Yang. Graph-revised convolutional network. In *ECML-PKDD*, 2020.

Yingxue Zhang, Soumyasundar Pal, Mark Coates, and Deniz Ustebay. Bayesian graph convolutional neural networks for semi-supervised classification. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):5829–5836, Jul. 2019.

Dengyong Zhou, Olivier Bousquet, Thomas Lal, Jason Weston, and Bernhard Olkopf. Learning with local and global consistency. *Advances in Neural Information Processing Systems 16*, 2004.

Xiaojin Zhu and Zoubin Ghahramani. Learning from labeled and unlabeled data with label propagation. Technical report, Carnegie Mellon University, 2002.

A APPENDIX

A.1 DATASETS

We evaluate on five citation network datasets: Cora, CiteSeer, PubMed taken from Jaafor & Birregah (2017), and ACM, DBLP taken from Wang et al. (2019). In the DBLP dataset, each node represents an author. In rest of the datasets, nodes correspond to documents (scientific papers). The node features correspond to sparse bag-of-words features with either binary or TF-IDF values. These datasets are evaluated for node classification task in a transductive setting closely following the experimental setup of Yang et al. (2016). For Cora, CiteSeer, and PubMed datasets, we use the standard split from previous work (Kipf & Welling, 2017). For ACM and DBLP datasets, we fix the validation and test set to 500 and 1000 nodes and create a training set by sampling 20 nodes per class from the remaining nodes to be in line with the standard split. Additionally, we also evaluate on three text classification datasets where a graph is not available with the given dataset. 20NG consists of long text documents, categorized into 20 newsgroups. MR consists of movie reviews classified into positive and negative sentiments. Ohsumed consists of medical abstracts, which are categorized into 23 cardiovascular diseases (Yao et al., 2019). Detailed statistics of the datasets used are available in the Table 5.

	Cora	**Citeseer**	**Pubmed**	**ACM**	**DBLP**
Nodes	2,708	3,327	19,717	3,025	4,057
Features	1,433	3,703	500	1,830	334
Classes	7	6	3	3	4
No. Training nodes	140	120	60	60	80

Table 5: Citation Datasets Statistics

Dataset	**Words**	**Docs**	**Train Docs**	**Test Docs**	**Classes**	**Avg. Length**
20NG	42,757	18,846	11,314	7,532	20	221.26
MR	18,764	10,662	7,108	3,554	2	20.39
Ohsumed	14,157	7,400	3,357	4,043	23	135.82

Table 6: Text Datasets Statistics

A.2 BASELINES AND HYPER-PARAMETERS RANGES

We compare GLAM against several baselines covering simple non-graph based approaches, semi-supervised classification methods, graph neural networks, and state-of-the-art graph learning approaches. Below are the hyper-parameters ranges we followed for tuning the baselines. For all the methods that rely on graphs, we construct *k*NN graphs from input features using cosine metric. Number of neighbors, *k*, is a hyper-parameter and swept over $\{5, 10, 15, 20\}$.

LogReg: Logistic Regression's weight-decay hyper-parameter, C, is tuned over [1e-4, 1e4] in powers of 10 on the validation set.

MLP: We employ a Multi Layer Perceptron with 1 hidden layer. For tuning, hidden layer dimensions were swept over $\{32, 64, 128\}$, weight decay over [1e-4, 1e4] in logarithmic steps, learning rate over [1e-3, 1e-2, 1e-1], and dropout from (0, 1).

LP: Zhu & Ghahramani (2002) In Label Propagation, we tune the hyper-parameter α (clamping factor) over the range (0, 1) in steps of 0.01. We observe much better results for LP than reported in LDS because of this extensive tuning.

ManiReg: Belkin et al. (2006) Manifold regularization's hyper-parameters γ_{A} and γ_{I} are tuned from the range (1e-5, 1e2) in logarithmic steps. We observe a discrepancy in ManiReg's numbers reported in LDS. Kipf & Welling (2017) reported 59.5 and 60.1 as test performance on Cora and Citeseer datasets using the original graphs that are part of the datasets. LDS reports 62.3 and 67.7 as mean test accuracy for these datasets using *k*NN graphs. However, *k*NN graphs are of poor quality in terms of homophily and GNN performance on these datasets.

SemiEmb: Weston et al. (2008) Semi-Supervised embedding's hyper-parameters λ is tuned over (1e-5, 1e2) in logarithmic steps, hidden layer dims over $\{32, 64, 128\}$, weight decay over [1e-4, 1e4] in steps of 10, learning rate from $\{$1e-3, 1e-2, 1e-1$\}$, and dropout from (0, 1).

GCN: Kipf & Welling (2017) We used 2 layered Graph Convolutional Networks and follow the hyper-parameter ranges mentioned in Shchur et al. (2018) for tuning.

GAT: Veličković et al. (2018) For tuning Graph Attention Networks, we consulted Shchur et al. (2018) for hyper-parameters ranges.

SuperGAT: Kim & Oh (2021) we rely on authors[1] code for experiments and follow Kim & Oh (2021) for tuning.

LDS: Franceschi et al. (2019) We rely on authors[2] code to perform LDS experiments on our benchmark datasets. We follow the hyper-parameters ranges mentioned by the author.

IDGL: Chen et al. (2020) We rely on authors[3] code to perform IDGL experiments on our datasets. For PubMed dataset, we ran IDGL-Anchor variant to report numbers and for the rest datasets, IDGL base variant was used for conducting experiments. The hyper-parameters mentioned by the author are used for tuning the model.

CL-MLP: Cascante-Bonilla et al. (2020) We implemented Curriculum Labeling algorithm, a very recent SSL method, for an MLP as described by the authors. For tuning the MLP, we used the hyper-parameters described above.

ICT-MLP: Verma et al. (2019) Interpolation Consistency Training is one of the latest SSL method proposed. We implemented ICT algorithm for an MLP followed Verma et al. (2019) for hyper-parameter tuning.

G3NN: Ma et al. (2019) We rely on authors[4] code for experiments and sweep the hyper-parameters ranges suggested by the authors.

BGCN: Zhang et al. (2019) We rely on authors[5] code for our experiments and sweep the hyper-parameters ranges suggested by the authors.

GLAM: Hyper-parameters α_A and α_C are tuned from (1e-5, 1e4) in logarithmic steps, learning rate from (1e-3, 1e0), dropouts from (0, 1), W_{ck} from (0, 1), affinity classifer's hidden layer dims from $\{32, 64, 128, 256\}$, GNN's hidden layer dims from $\{16, 32, 64, 128\}$, *k*NN's *k* from $\{5, 10, 15, 20\}$, and gumbel softmax's temperature is set to 1e-10. We use Adam optimizer to minimize our combined loss term. GLAM is trained for 500 epochs with an early stopping criterion of no improvement in validation accuracy for 25 epochs. Hyper-parameter tuning was done using *optuna* Akiba et al. (2019). We swept through 2000 configurations using TPE sampler in *optuna* for each dataset.

All the models we implemented except LP and LogReg were written in *Tensorflow* Abadi et al. (2015). LP and LogReg were implemented using scikit-learn python package Pedregosa et al. (2012). For all models, test accuracy is reported for the configuration that achieves the highest validation accuracy.

A.3 Additional Results

We compare GLAM against additional baselines including recent SSL methods, bug-fixed version of GAM, Bayesian graph methods and additional results in Table A.3. We also assess GLAM on datasets where a graph is genuinely not available along with the datasets. Table A.3 shows that GLAM consistently outperforms previous state-of-the-art methods on several of these datasets.

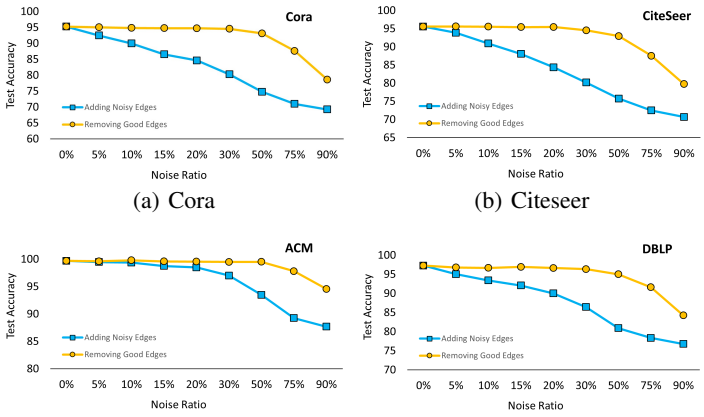
A.4 Additional Analysis

Noise Analysis. We perform the following analysis to show that having a few noisy edges can hurt performance more than throwing away several good edges (i.e. edges where source and target nodes have same labels). We construct *k*NN graphs for several benchmark datasets. We throw away all the noisy edges from this graph and call this as Perfect-kNN graph. We conduct two experiments

[1] https://github.com/dongkwan-kim/SuperGAT
[2] https://github.com/lucfra/LDS-GNN
[3] https://github.com/hugochan/IDGL
[4] https://github.com/jiaqima/G3NN
[5] https://github.com/huawei-noah/BGCN

	Features					Boosted Features				
Models \Datasets	**20NG**	**MR**	**Ohsumed**	**ACM**	**CiteSeer**	**20NG**	**MR**	**Ohsumed**	**ACM**	**CiteSeer**
MLP	64.89 (1.24)	62.22 (1.21)	40.02 (1.30)	79.92 (0.55)	62.02 (1.80)	77.95 (1.12)	66.95 (0.80)	46.95 (1.11)	87.66 (0.69)	68.50 (1.52)
SemiEmb	74.10 (1.65)	66.08 (1.09)	42.78 (1.93)	88.76 (1.21)	66.44 (0.92)	77.17 (0.90)	66.77 (1.22)	46.87 (1.63)	88.88 (0.33)	70.09 (0.10)
CL - MLP	65.26 (1.24)	64.45 (1.42)	45.20 (2.49)	**89.77 (0.76)**	69.40 (1.01)	80.87 (0.41)	68.03 (0.71)	48.11 (0.71)	85.83 (1.59)	69.57 (2.41)
ICT - MLP	61.66 (1.34)	62.44 (0.84)	39.36 (1.43)	79.42 (0.80)	58.80 (0.76)	77.14 (0.89)	66.29 (0.71)	45.30 (1.25)	91.28 (0.35)	68.80 (0.61)
G3NN	70.45 (0.85)	65.48 (1.57)	41.50 (0.42)	87.12 (0.48)	70.12 (0.27)	72.08 (0.80)	63.11 (0.27)	42.73 (1.18)	90.62 (0.73)	70.82 (0.32)
BGCN	71.59 (0.58)	66.07 (0.67)	44.26 (1.29)	87.14 (0.63)	68.52 (1.37)	74.23 (0.63)	64.52 (1.32)	44.38 (0.67)	91.26 (0.25)	69.28 (0.79)
LDS	OOM	69.33 (0.90)	47.82 (1.33)	86.98 (0.78)	72.16 (0.61)	OOM	67.16 (1.61)	47.07 (1.61)	**92.93 (0.61)**	71.44 (0.40)
IDGL	68.21 (1.45)	68.58 (0.90)	44.82 (1.34)	88.94 (0.52)	67.65 (1.71)	69.23 (1.15)	66.03 (1.42)	43.65 (1.30)	91.09 (0.38)	68.88 (0.44)
GCN + GAM	64.74 (0.83)	66.82 (1.33)	46.32 (0.78)	88.33 (1.67)	70.43 (1.93)	64.93 (0.47)	66.89 (1.05)	44.51 (1.21)	91.11 (0.96)	68.50 (0.99)
GCN + GAM*	54.73 (0.76)	56.46 (0.64)	37.09 (2.07)	66.32 (1.41)	57.22 (1.27)	73.52 (1.31)	59.56 (0.78)	40.03 (0.58)	68.33 (2.31)	59.17 (1.44)
GLAM	**77.01 (1.81)**	**69.43 (0.97)**	**49.40 (1.01)**	89.34 (0.15)	**72.22 (0.45)**	**81.32 (0.76)**	**68.26 (0.68)**	**48.25 (0.84)**	92.38 (0.20)	**71.86 (0.44)**

Table 7: Subset of results on additional datasets and baselines.



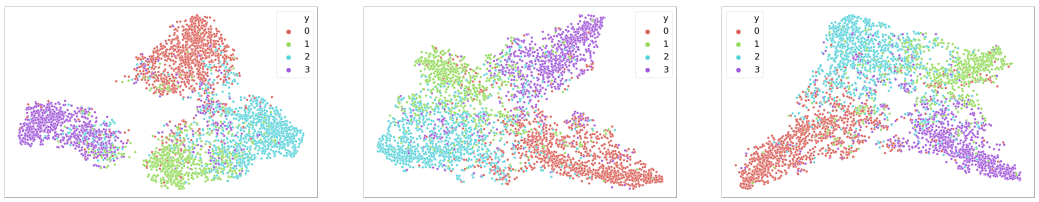
(a) Cora (b) Citeseer

(c) ACM (d) DBLP

Figure 2: Noise Analysis Plots

- 1] We randomly add noisy edges to this Perfect-kNN graph, and 2] We randomly remove good edges from the Perfect-kNN graph. We report test accuracies (using GCN) for varying percentages of added noisy edges and removed good edges. Figure 2 shows these plots. We observe that adding noisy edges deteriorates the performance of the model. However, even after removing 50%-75% of good edges, the GCN can still perform well on the test set. This observation suggests that it is important to reduce the amount of noise added rather than saving good edges.

GLAM v/s IDGL: Latent Node Representations. IDGL is an iterative approach, thereby, low-quality graph leads to learning poor representations, and this effect cascades over multiple iterations leading to minor or no accuracy gains. We note that IDGL performs better on PubMed compared to GLAM as the *k*nn graph has the homophily score close to the original link graph. GLAM works better in the presence of noisy graphs. Figures 3(b), 3(c) shows IDGL's first and last iteration's embeddings and Figure 3(a) shows GLAM's embeddings TSNE plots on the DBLP dataset. We observe that the final iteration's plot is similar to the first iteration plot whereas GLAM's plot shows discernible clusters.

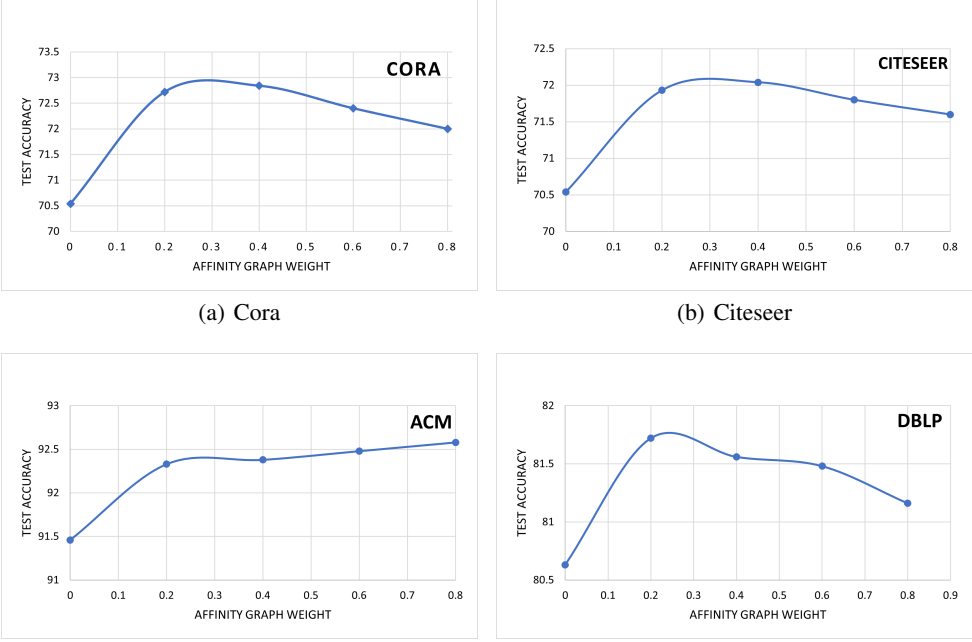


(a) GLAM (b) IDGL - First Iteration (c) IDGL - Last Iteration

Figure 3: TSNE Plots on DBLP Dataset

	Cora	CiteSeer	ACM	DBLP
Affinity Graph Weight = 1.0	63.88	62.8	85.52	74.43

Table 8: GCN's performance on Affinity Graph



(a) Cora (b) Citeseer

(c) ACM (d) DBLP

Figure 4: Effect of Weight on Affinity Graph Plots

Is Affinity Graph alone enough?. Table 8 shows GCN's performance on using only affinity graph. We see that using affinity graph alone is not sufficient. Affinity graph only contains edges from all nodes to labeled nodes restricting feature propagation, thus limiting the performance of GNNs. We study how placing different weights on the affinity graph during graph combination affects GCN's Kipf & Welling (2017) performance. We observe that in all the datasets, adding affinity graphs improves the performance of GNNs. Figure 4 illustrates the effect of the affinity graph on four different benchmark datasets. A bell shape sort of behavior is observed in most cases, where the performance of GNN starts to dwindle as more weight is given to the affinity graph. *K*NN graphs are responsible for feature propagation. Placing more weight on the affinity graph affects feature propagation and we start losing performance after a certain point.