Using Node Similarity to Achieve Local Differential Privacy in Graph Neural Networks

Saqib Ameen, Thirupathi Reddy Emireddy, Justin Stevens
Department of Computing Science
University of Alberta
Edmonton, AB, T6G 2R3
saqib1@ualberta.ca, emireddy@ualberta.ca, jdsteven@ualberta.ca

Abstract

Protecting the privacy of node data in Graph Neural Networks (GNN) has become an area of paramount importance due to the widespread application of GNNs in various domains that involve sensitive data. Local Differential Privacy (LDP) has proved to be an effective tool in achieving node data privacy. However, there is always a tradeoff between privacy and utility. Prior works have tried to reduce this gap by leveraging the correlation between aggregation benefits offered by LDP and the initial aggregation layer of GNN. They did so by expanding the node neighborhood in approximating the initial aggregation layer. However, they failed to mitigate the impact of dissimilar nodes on accuracy that arises while performing neighborhood expansion with nodes far away from the node of interest. We propose a new algorithm, SIMGAT, that addresses this limitation by first considering the neighborhoods of nodes at a closer distance, then performing aggregation with the neighborhoods of similar nodes. We show that our method improves the utility under strict privacy guarantees and performs better than the state-of-the-art work.

1 Introduction

GNNs have recently gained popularity due to their ability to learn a rich representation of relational data and solve complex problems. They are being widely adopted to achieve state-of-the-art results in various domains like social sciences, biological sciences, recommendation systems, and program synthesis [2, 4, 9, 17]. Recently, GNNs have made breakthroughs possible like AlphaFold, which solves a long-standing problem of predicting protein structure for the medical community [19]. In general, GNNs can be useful in tasks related to link prediction, node classification, and graph classification.

GNNs are also being used by social networking sites which involve and operate on human data. For instance, Facebook treats each person as a node and the edges between different nodes represent how the people are connected. This information can be utilized for friend recommendations. To make such recommendations, Facebook needs to utilize the users' data which may include sensitive information. Multiple recent studies have tried to demonstrate that through sophisticated attacks on GNNs, such information can be inferred even from the black-box model [6, 12, 20]. This puts individuals at risk and renders privacy in GNNs to be a crucial problem.

To mitigate this, various techniques have been developed in the past. Differential privacy (DP) is one of them and has become very popular among the entities collecting and utilizing users' data to make sure that individual's privacy is protected [7]. Intuitively, differential privacy guarantees that a third-party entity can't significantly distinguish whether or not an individual is a part of a dataset. This technique has become the de facto standard in intelligent systems to protect user privacy. It does not only provide anonymization, but also helps in preventing the information linkage in case of a data

Project Code Available at https://github.com/saqibameen/LPGNN

leak. Furthermore, preserving user's privacy helps entities developing these systems to gain users' trust and abide by the privacy laws. User data privacy laws such as the European Union's General Data Protection Regulation (GDPR) strictly bind the entities developing intelligent systems dealing with user data to collect and manage data fairly. Implementing DP in such cases can be helpful.

While DP is widely used in intelligent systems to achieve user data privacy, it is still an active topic of research for different settings and learning methods. In particular, the work of privacy in the case of GNNs is still in the early stages. One of the key challenges in this niche is the trade-off between privacy and accuracy. In our work, we consider the issue of protecting user data privacy in node classification tasks using GNNs without significantly compromising the accuracy of the GNN model.

2 Background

2.1 Graph Neural Networks (GNNs)

We use the standard definition of GNNs in a fashion similar to [16]. GNNs learn a powerful representation of the underlying graph structure to try to predict a label using a set of graph convolution layers. Consider the input to a graph neural network as $G = (\mathcal{V}, \mathcal{E}, \mathcal{X}, \mathcal{Y})$, where $\mathcal{V}$ is the set of nodes, $\mathcal{E}$ is the set of edges/links between the nodes, $\mathcal{X}$ is the set of features for all the graph nodes, and $\mathcal{Y}$ is each nodes' label. For an M-layer convolutional graph neural network, at any given layer m, the model updates the embedding vector h_v^m for every node $v \in \mathcal{V}$ by first aggregating the embeddings of the neighboring nodes from the previous $(m-1)$ layer. The aggregation step is defined as:

$$h_{\mathcal{N}(v)}^m = \text{Aggregate}(h_u^{m-1} \forall u \in \mathcal{N}(v)) \tag{1}$$

$$h_v^m = \text{Update}(h_{\mathcal{N}(v)}^m). \tag{2}$$

The $\mathcal{N}(v)$ in equation (1) represents the neighbors of a given node v. An aggregate function is usually a permutation invariant function such as sum, max, or mean. The next step after aggregation is update, which is a trainable non-linear function. The specific update function we use in our work is known as GRAPHSAGE, which applies a weight matrix to the concatenation of h_v^{m-1} and $h_{\mathcal{N}(v)}^m$ at every layer, and learns the weight matrix using gradient descent [11].

For the initial layer, we consider the feature vector of node x_v as its embedding, i.e., $h_v^0 = x_v$. The last layer gives a c-dimensional output, where c is the total number of classes in our task. A softmax layer is applied to predict the labels.

2.2 Local Differential Privacy

LDP has become a ubiquitous approach in collecting private data by data aggregators in approximating the aggregate queries without compromising the privacy of the data owners. Many major technology companies like Google [8], Apple[1], and Microsoft[5] have adopted LDP in collecting user feedback for improving their products while keeping individual users' data private. In LDP, a data holder does not send their private data to an untrusted aggregator as is, but rather sends a perturbed version of their data. The individual perturbed data is not meaningful in itself but can help the aggregator approximate the target query with better accuracy.

LDP consists of two steps. In the first step, users first perturb their data using a randomized mechanism $\mathcal{M}$ and send it to the data aggregator. In the second step, the aggregator approximates the target query by combining the perturbed data received from all the data holders. To achieve LDP, mechanism $\mathcal{M}$ must satisfy the below definition of differential privacy [13]:

Definition 2.1 *Given "privacy budget" $\epsilon > 0$, a randomized mechanism $\mathcal{M}$ satisfies ϵ-local differential privacy, if for all possible pairs of private data x and x', and for all possible outputs $y \in$ Range($\mathcal{M}$), we have: $Pr[\mathcal{M}(x) = y] \leq e^{\epsilon} Pr[\mathcal{M}(x') = y]$*

The "privacy budget" ϵ is used in tuning the tradeoff between privacy and utility. Lower values of ϵ translate to stronger privacy guarantees with lower utility and vice versa. LDP benefits the aggregators because the noise added to individual data gets canceled when aggregation is performed over a large set of samples thereby providing a good estimate of the target aggregation query.

3 Related Work

In recent years, an influx in the adoption of GNNs is observed due to their remarkable ability to solve real-world tasks. As a result, research studies around GNNs have also seen rapid growth. One area of focus in such studies is differential privacy in GNN models. This section lists a few of the latest and closely related contributions to our work.

The first set of techniques used to preserve privacy in GNNs is based on utilizing classic privacy-preserving algorithms such as perturbing the loss function [22], or privately perturbed gradient descent [23] to achieve differentially private embeddings. While they extend the classic embedding algorithms to achieve privacy, they do not fully utilize the graphical features. Another set of approaches is pivoted on federated and split learning of graph neural networks. For instance, [21] computes the differentially private gradients locally and securely transmits them to the server for aggregation. However, this suggested approach is limited to recommendation systems and assumes that models have access to the features. It also assumes that a users' device is powerful enough to compute the gradients.

Another work introduces SAPGNN [18] to preserve privacy during node classification. However, their approach only targets horizontally partitioned graph datasets and is only applicable in split-learning settings. Further, it does not generalize well to the other graph datasets. Our proposed approach is more general than these approaches since we don't use federated or split learning, making no such assumptions on the dataset or model. Our work is closely related to the Locally Private Graph Neural Networks [16] work. We extend their work to improve accuracy on a strict privacy budget.

3.1 Locally Private Graph Neural Networks (LPGNN)

LPGNN [16] applies the idea of LDP to GNN for achieving end-user data privacy. They assume that there is no third-party trusted aggregator and noise is added to node features before transmitting it to the server. The way that noise is added to the features is through a multi-bit encoder, which runs at the user side by randomly perturbing the feature vector. They prove that this method satisfies Definition 2.1 of differential privacy. This is followed by a multi-bit rectifier, which runs at the server-side for debiasing the perturbed data.

The first layer in the GNN is an aggregation layer over the neighbors of nodes. Ideally, this would closely approximate the true initial aggregation layer if the nodes have a high degree, but in reality, graphs follow power-law degree distribution and most nodes have few neighbors. As a result, the noise is not completely canceled out. To solve this problem, LPGNN introduced a neighborhood expansion mechanism called K-Prop, which considers not only the immediate neighbors but also neighbors up to K hops away from the node in initial layer aggregation.

Although they have achieved good accuracy when aggregation is extended to a large number of neighbors, using far-off neighbors' aggregated node data as each node's initial feature set can lead to misrepresentation of the node feature data. This is because nodes become highly unrelated to each other as we go further away from our original node. We aim to address this problem while maintaining good accuracy by exploiting the node similarity measures in the graph.

4 Methodology

We start with the LPGNN architecture as the base and extend their work by introducing a new denoising layer based on a novel algorithm called SimGat to achieve better accuracy. On top of K-Prop, SimGat extends the neighborhood to similar nodes as well. This idea of considering similar nodes helps us to better cancel the noise during the aggregation of features in the first layer. As a result, better accuracy can be achieved during the classification task.

Algorithm 1 shows the pseudocode for SIMGAT. In line 2, we run the K-PROP algorithm from LPGNN [16] to get node embedding h'_v for each node v in our graph. Then in line 3, we use one of our similarity method S (GRAREP, DEEPWALK, NODE2VEC), to get the set of similar nodes $\mathrm{S}(v)$ for a given node v. Finally in line 4, we compute the node embedding h_v by aggregating all the embeddings of the similar nodes returned by $\mathrm{S}(v)$.

By considering this node similarity, we also try to limit the depth of the neighborhood that must be considered (K'). The similarity methods that we use are explained in the section below.

Algorithm 1: SimGat Algorithm

Input : Graph $G = (\mathcal{V}, \mathcal{E})$, $x_v \, \forall v \in \mathcal{V}$, Number of Hops K', Aggregation Function AGGREGATE, Similarity Method S (GRAREP, DEEPWALK, NODE2VEC)

Output : Node embeddings $h_v \, \forall v \in \mathcal{V}$

foreach $v \in \mathcal{V}$ **do**
  $h'_v \leftarrow$ K-PROP$(G, x_v,$ AGGREGATE$, K' - 1)$
  Compute similar nodes $\mathrm{S}(v)$
  $h_v \leftarrow$ AGGREGATE$(h'_u \, \forall u \in \mathrm{S}(v))$
return $\{h_v \, \forall v \in V\}$

4.1 Node Similarity Methods

We considered three methods of node similarity inspired by the works of [3, 10, 15]. The objective of these works is to map high dimensional feature space into a low dimensional feature embedding. They do this by finding similarities among the nodes. The idea is similar nodes will have similar embeddings in the embedding space. We leverage these methods to find similar nodes for us to use in aggregation. In the following sections, each of the similarity methods is explained in detail, and the pseudocode is given in the Appendix in Section A.3.

4.1.1 GRAREP Similarity

This node similarity method is based on GRAREP (Graph Representation) [3]. Algorithm 2 represents the pseudocode for this method. In this method, we first find the transition probability among the nodes after a number of transition steps m (also known as walk length). This is done by normalizing the summation performed over the power adjacency matrices up to the degree same as the number of transition steps. It is known that the k^{th}-power of an adjacency matrix gives the number of possible paths between nodes that are a distance of exactly k hops. Once the transition probability matrix is calculated, we consider the similarity among nodes based on transition probability being greater than a similarity threshold δ. We take the number of transition steps m and similarity threshold δ as a set of hyperparameters to tune.

4.1.2 DEEPWALK Similarity

DEEPWALK is a classical technique for computing the graph node embeddings [15]. Algorithm 3 represents the pseudocode for this similarity method based on DEEPWALK. The idea of similarity in DEEPWALK is inspired by the popular WORD2VEC[14] technique from NLP literature. DEEPWALK begins by first computing unbiased random walks for each node v in the graph. After that, it treats the random walks as the equivalent of sentences in WORD2VEC and computes the co-occurrence probability for each pair of nodes in a random walk using the skip-gram model. The result of DEEPWALK is an embedding matrix E, where each pair of similar nodes have similar embedding.

We extend these embeddings to get a similarity matrix $\mathcal{S}$ of dimensions $n \times n$. Once we get the embeddings matrix $E_{n \times d}$, we compute the l_2 norm of the difference of each pair of nodes embeddings (represented as rows), to obtain a difference matrix $\mathcal{D}_{n \times n}$. Then this matrix is normalized. Finally, we calculate the similarity matrix $\mathcal{S}$ based on a threshold δ, such that the pairs with a value less than δ are assigned a value of 1 (similar nodes) and 0 (dissimilar nodes) otherwise.

4.1.3 NODE2VEC Similarity

NODE2VEC is a method for generating embeddings based on biased random walks [10]. The pseudocode for node2vec similarity method is given in Algorithm 4. The random walks are generated by parameters p and q. After a random walk moves from node n_a to node n_b, the probability distribution for the next move n_c , in un-normalized form, is given by the following probability

distribution:

$$\Pr(\text{Move to node } n_c | \text{Previous move was } n_a \to n_b) \propto \begin{cases} \frac{1}{p} \text{ if } c = a \\ 1 \text{ if } n_c \text{ and } n_a \text{ are neighbors} \\ \frac{1}{q} \text{ otherwise.} \end{cases}$$

NODE2VEC uses the above search policy to generate m random walks of length ℓ. It consider two nodes similar if they co-occur on many random walks together. NODE2VEC uses a model such as WORD2VEC [14] to compute a node embedding for each node in the graph, E of dimensions $N \times d$.

We normalize this node embedding so that each row has norm 1 to get a new matrix N. We then compute the cosine similarity between two node embeddings. Finally, we consider two nodes similar if their cosine similarity is above a set threshold, δ.

5 Experiments and Analysis

We consider K-Prop from LPGNN [16] as the baseline to compare our work with. For the experimental settings, we take the privacy budget on the node features as 0.01 ($\epsilon_x = 0.01$), which is the most strict privacy budget used in LPGNN. Furthermore, we take the number of aggregation steps (K') in SIMGAT from the set $\{0, 2, 4, 8, 16\}$, which is the same as that of LPGNN. It should be noted that in this section, K represents the number of aggregation steps in K-Prop, while K' represents the number of aggregation steps in SIMGAT.

We use the same 2 datasets from LPGNN, Cora, and LastFM. Cora is a citation dataset with a low average node degree. On the other hand, LastFM is a social media music dataset, with a high average node degree. We perform three sets of experiments. In the first scenario, we compare the accuracy of SIMGAT and K-Prop for different values of K and K' for the Cora dataset. In the second scenario, we vary the similarity thresholds with a fixed value of K and K' on the Cora dataset to study the effect of the similarity threshold. Finally, we study the effect of the node degree on accuracy by comparing the results for the LastFM and Cora datasets. All three sets of experiments are explained in the following sections, accompanied by a discussion of the results. Further details regarding the setup used for experiments are presented in the Appendix A.2.

5.1 Effect of Number of Hops (K')

To observe how the number of hops impacts accuracy, we perform experiments with three values of K', given by $K' = K$, $K' = K - 1$, and $K' = K - 2$. In these, we keep the number of hops K constant for K-Prop. From Figure 1, we can see that one or more similarity methods from SIMGAT perform better than K-Prop for any given value of K'.

For $K' = K$, we see that as the number of hops increases, all of our methods perform better than K-Prop. However, at smaller values of K', K-Prop is competitive with GRAREP and NODE2VEC but is still worse than DEEPWALK which performs better than K-Prop for almost all values of K'. At $K' = 16$, GRAREP performs better among all.

For $K' = K - 1$ and $K' = K - 2$, only DEEPWALK performs better than K-Prop. This shows that DEEPWALK is more invariant to the number of hops. However, for the rest of the similarity methods, as we decrease the value of K', the number of nodes considered for aggregation decreases, and we therefore observe a decrease in accuracy.

An interesting takeaway from this set of experiments is that not only the number of nodes but also the similarity of the nodes that we consider in the aggregation of the first layer influences the accuracy of the denoising method. This can be concluded from the observation that DEEPWALK at $K' = K - 1$ and $K' = K - 2$ is performing better than K-Prop at K, and they only differ in the type of nodes considered.

5.2 Effect of Similarity Threshold (δ)

Similarity threshold is another factor that influences the initial aggregation layer approximation. The number of similar nodes for a given node increase as the thresholds becomes less strict. Strict thresholds give highly similar nodes but the number of nodes considered for the aggregation decrease.

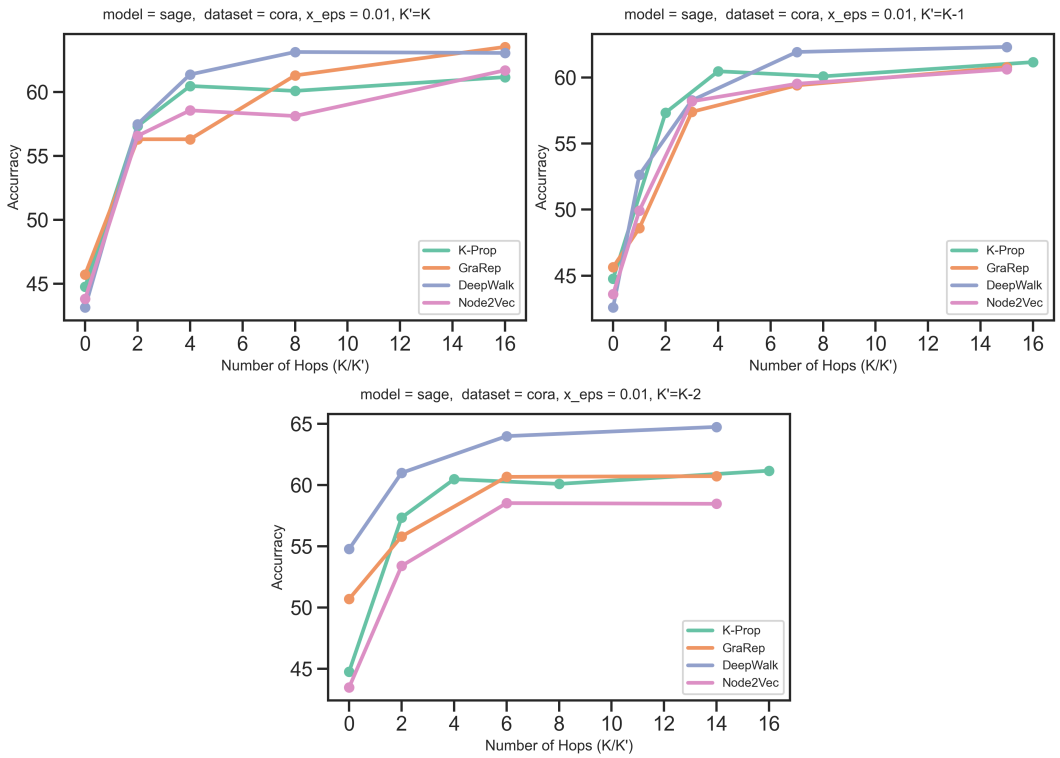

Figure 1: Effect of reducing the number of hops (K/K') on the accurracy.

Thresholds that are not strict consider a large number of nodes as similar to any given node but they can be dissimilar to the node of concern. To analyze this pattern, we perform this set of experiments, where we vary the threshold differently for each of the three methods, with one weak, one moderate, and one strict threshold. These exact thresholds are given in Section A.2. We keep the number of hops the same (K'=K) for both SIMGAT and K-Prop.

In Figure 2 (top left), at the strict threshold, we see that the DEEPWALK similarity method with SIMGAT performs better than K-Prop. Since SIMGAT similarity methods have fewer similar nodes to aggregate, there is less room for aggregating nodes that are less related to the given node. Therefore, SIMGAT only aggregates the highly similar nodes, which improves the accuracy of our model.

For the moderate threshold (top right), DEEPWALK continues to perform better than K-Prop, however, K-Prop is competitive with NODE2VEC and GRAREP. K-Prop is able to remain more competitive with these other methods since they begin to aggregate more dissimilar nodes.

Finally, for the weaker threshold (bottom), K-prop performs better than all of the other similarity methods. At this weaker threshold, all of the similarity methods aggregate many nodes, some of which may be completely dissimilar to the original node. Therefore, the impact of aggregating similar nodes is decreased significantly, since it no longer aggregates the similar nodes to a given node. In conclusion, we can say that a fine-tuned threshold is important to filter the similar nodes and achieve good accuracy. While for a weaker threshold, we tend to accumulate dissimilar nodes and lose accuracy, which aligns well with our intuition.

5.3 Effect of Node Degree

To study the impact of node degree on the approximation of the initial layer aggregation compare SIMGAT and K-Prop on two different datasets, namely Cora and LastFm. Cora represents a dataset with a low average node degree (3.90) and LastFM has a high average node degree (7.29). For both of these methods, we keep the value of K' and K constant and choose the best threshold for each of the three methods.

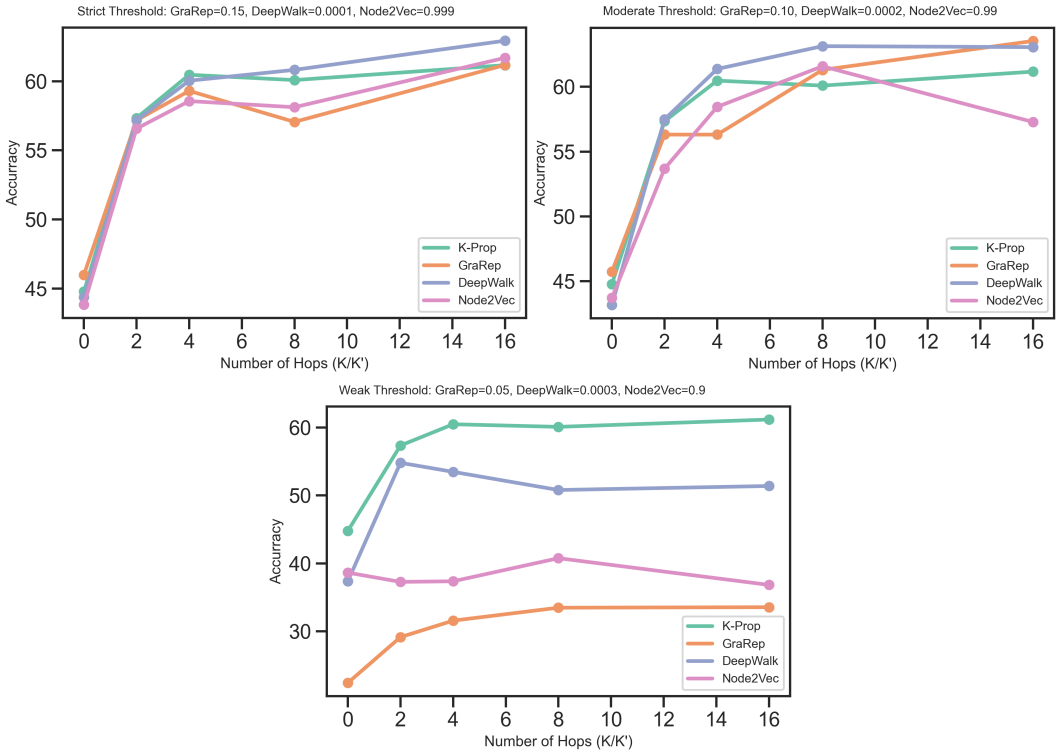

Figure 2: Effect of reducing threshold for fixed number of hops (K/K') on the accuracy.

For the LastFM dataset (right) in Figure 3, we see that K-Prop's performance degrades drastically as we increase the value of K. This is expected, since when the average number of neighbors is larger, they will aggregate many nodes which are dissimilar from the original node, leading to worse performance. Since our similarity methods utilize similar nodes instead, they are more permeable to larger values of K', as we see that their performance doesn't degrade as much as K-Prop. It may be noted that at a lower number of hops ($K' = K = 4$), all of the methods give their best accuracy. This shows that we need to aggregate fewer nodes when the average node degree is high.

For Cora, since it has a lower number of average node degrees, the difference in the performance between K-Prop and SIMGAT is less pronounced. For all of the methods, the accuracy increase as we increase the number of hops. This shows that, for higher average degree nodes, we need a fewer number of hops to achieve good accuracy and vice versa. Irrespective of average degree, one or more SIMGAT methods achieve better accuracy.

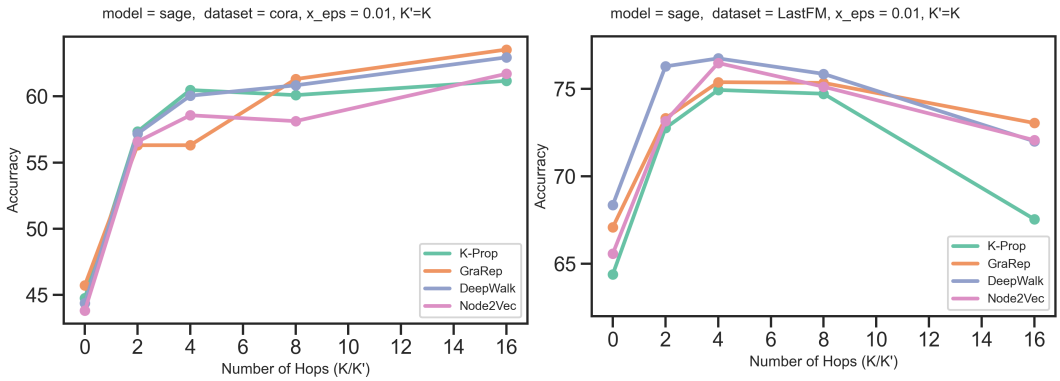

Figure 3: Effect of node degree on performance of SIMGAT

6 Limitations and Future Work

The methods based on similarity measures have an inherent limitation since they only consider nodes that are similar within the range of similarity distance (number of transition hops in GRAREP, random walk length in DEEPWALK, and NODE2VEC). Our methods do not consider the similar nodes that are outside the similarity distance. When the graphs are large and sparse, a greater number of similar nodes exist in the graph very far from the node, so methods based on proximity-based similarity do not scale well in such scenarios. In large and sparse graphs, node similarity measures that work on graph level features like graphlets and graph kernels scale well.

One interesting direction of research is to find similar nodes based on graph structural features instead of proximity-based similarity measures so that our work can be scaled for larger graphs. Another direction of research can be in adopting a hybrid approach that leverages the benefits of both similarity measures as well as graph level features to find similar nodes.

We further saw how our methods were negatively impacted by moving to a weaker similarity threshold. Therefore, to use our similarity methods in practice, a practitioner would have to carefully tune the threshold hyperparameter to obtain competitive performance with K-Prop. This creates extra work on their part, and it would be interesting to study how we could build these methods to be more robust with varying thresholds.

7 Conclusion

In our work, we presented a novel denoising algorithm called SIMGAT to address the shortcomings of the K-Prop algorithm from the LGPNN[16] paper. Our algorithm, SIMGAT identifies similar nodes using GRAREP, DEEPWALK, and NODE2VEC as its similarity methods. Then it aggregates the similar nodes after running K-Prop for a smaller value of K'. By aggregating similar nodes, we improve the neighborhood expansion while considering fewer far away dissimilar nodes. As a result, we are better able to denoise the initial aggregation layer by improving the accuracy.

To evaluate our methodology, we compared our results with K-Prop under different experimental settings. We found that DEEPWALK performed the best in terms of accuracy among all. One or more similarity methods from SIMGAT always performed better than K-Prop, except for in the weak threshold setting. Furthermore, when we used a dataset with a high average node degree, all three similarity methods from SIMGAT significantly outperformed K-Prop, showing that our proposed mechanism is more versatile to different datasets.

References

[1] Umesh S Vaishampayan Gaurav Kapoor Julien Freudiger Vivek Rangarajan Sridhar Abhradeep Guha Thakurta, Andrew H Vyrros and Doug Davidson. 2017. Learning new words. US Patent 9594741.

[2] David Bieber, Charles Sutton, Hugo Larochelle, and Daniel Tarlow. Learning to execute programs with instruction pointer attention graph neural networks. *CoRR*, abs/2010.12621, 2020.

[3] Shaosheng Cao, Wei Lu, and Qiongkai Xu. Grarep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, CIKM '15, page 891–900, New York, NY, USA, 2015. Association for Computing Machinery.

[4] Ameya Daigavane, Balaraman Ravindran, and Gaurav Aggarwal. Understanding convolutions on graphs. *Distill*, 2021. https://distill.pub/2021/understanding-gnns.

[5] Bolin Ding, Janardhan Kulkarni, and Sergey Yekhanin. Collecting telemetry data privately. *arXiv preprint arXiv:1712.01524*, 2017.

[6] Vasisht Duddu, Antoine Boutet, and Virat Shejwalkar. Quantifying privacy leakage in graph embedding. *MobiQuitous 2020 - 17th EAI International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, Dec 2020.

[7] Cynthia Dwork, Aaron Roth, et al. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9(3-4):211–407, 2014.

[8] Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. Rappor: Randomized aggregatable privacy-preserving ordinal response. In *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, pages 1054–1067, 2014.

[9] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Yihong Eric Zhao, Jiliang Tang, and Dawei Yin. Graph neural networks for social recommendation. *CoRR*, abs/1902.07243, 2019.

[10] 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.

[11] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

[12] Xinlei He, Jinyuan Jia, Michael Backes, Neil Zhenqiang Gong, and Yang Zhang. Stealing links from graph neural networks. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 2669–2686. USENIX Association, August 2021.

[13] Shiva Prasad Kasiviswanathan, Homin K Lee, Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. What can we learn privately? *SIAM Journal on Computing*, 40(3):793–826, 2011.

[14] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[15] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710, 2014.

[16] Sina Sajadmanesh and Daniel Gatica-Perez. Locally private graph neural networks. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, CCS '21, page 2130–2145. Association for Computing Machinery, 2021.

[17] Benjamin Sanchez-Lengeling, Emily Reif, Adam Pearce, and Alexander B. Wiltschko. A gentle introduction to graph neural networks. *Distill*, 2021. https://distill.pub/2021/gnn-intro.

[18] Chuanqiang Shan, Huiyun Jiao, and Jie Fu. Towards representation identical privacy-preserving graph neural network via split learning. *CoRR*, abs/2107.05917, 2021.

[19] Jeffrey Skolnick, Mu Gao, Hongyi Zhou, and Suresh Singh. Alphafold 2: Why it works and its implications for understanding the relationships of protein sequence, structure, and function. *Journal of Chemical Information and Modeling*, 61(10):4827–4831, 2021. PMID: 34586808.

[20] Bang Wu, Xiangwen Yang, Shirui Pan, and Xingliang Yuan. Model extraction attacks on graph neural networks: Taxonomy and realization. *CoRR*, abs/2010.12751, 2020.

[21] Chuhan Wu, Fangzhao Wu, Yang Cao, Yongfeng Huang, and Xing Xie. Fedgnn: Federated graph neural network for privacy-preserving recommendation. *CoRR*, abs/2102.04925, 2021.

[22] Depeng Xu, Shuhan Yuan, Xintao Wu, and Hai Nhat Phan. Dpne: Differentially private network embedding. In *Advances in Knowledge Discovery and Data Mining - 22nd Pacific-Asia Conference, PAKDD 2018, Proceedings*, pages 235–246, Germany, 2018. Springer Verlag.

[23] Sen Zhang and Weiwei Ni. Graph embedding matrix sharing with differential privacy. *IEEE Access*, 7:89390–89399, 2019.

A Appendix

A.1 Datasets

We considered the datasets Cora and LastFM for our experiments from [16]:

Datasets	#Classes	#Nodes	#Edges	#Features	Avg.Deg
Cora	7	2,708	5,278	1,433	3.90
LastFM	10	7,083	25,814	7,842	7.29

A.2 Similarity method parameters

We used the following hyperparameters in our experiments. The intuition for the different thresholds is we wanted one 'low' threshold such that there would be a lot of similar nodes, one intermediate threshold, and 'high' threshold such that there would be few similar nodes.

Similarity method	Walk length	Embedding Dimension	Number of walks	Thresholds used
GRAREP	30	-	-	δ =(0.05,0.1,0.15)
DEEPWALK	40	68	10	$\delta = (0.0001, 0.0002, 0.0003)$
NODE2VEC	35	64	10	$\delta = (0.9, 0.99, 0.999), p = 0.5, q = 3$

A.3 Pseudocode for Similarity Methods

Algorithm 2: GRAREP Similarity

Input : Adjacency matrix $\mathcal{A}_{n\times n}$, number of transition steps m, similarity threshold δ
Output : Similarity matrix $\mathcal{S}$

Transition matrix $\mathcal{T} \leftarrow \sum_{k=1}^{m} \mathcal{A}^k$

Transition probability matrix $\mathcal{P}$: $\mathcal{P}_{ij} \leftarrow \frac{\mathcal{T}_{ij}}{\sum_c \mathcal{T}_{ic}}$

Similarity matrix $\mathcal{S}$: $\mathcal{S}_{ij}$ = $\begin{cases} 1, & \text{if } \mathcal{P}_{ij} > \delta \\ & \text{or } \mathcal{A}_{ij} = 1 \\ 0, & \text{otherwise} \end{cases}$

return $\mathcal{S}$

Algorithm 3: DEEPWALK Similarity

input : Adjacency matrix $\mathcal{A}_{n\times n}$, random walk length ℓ, number of walks m, and a similarity threshold δ.
output : Similarity matrix $\mathcal{S}$

Embeddings matrix $E_{n\times d} \leftarrow$DEEPWALK($\mathcal{A}$,ℓ,m)

Difference matrix $\mathcal{D}$: $\mathcal{D}_{i,j}$= $\|e_i - e_j\|_2 \, \forall i, j \leq n$

Normalized norms $\mathcal{N} \leftarrow$ NORMALIZE($\mathcal{D}$)

Similarity matrix $\mathcal{S}$: $\mathcal{S}_{ij}$ = $\begin{cases} 1, & \text{if } \mathcal{N}_{ij} < \delta \\ 0, & \text{otherwise} \end{cases}$ **return** $\mathcal{S}$

Algorithm 4: NODE2VEC Similarity

input : Adjacency matrix $\mathcal{A}_{n\times n}$, breadth first search parameter p, depth first search parameter q, random walk length ℓ, number of walks m, similarity threshold δ
output : Similarity matrix $\mathcal{S}$

Embedding Matrix $E_{n\times d} \leftarrow$ NODE2VEC$(\mathcal{A}, \ell, p, q, m)$

Normalized embedding matrix $N \leftarrow$ NORMALIZE(E)

Cosine similarity matrix $\mathcal{C} \leftarrow N \cdot N^T$

Similarity matrix $\mathcal{S}$: $\mathcal{S}_{ij} = \begin{cases} 1, & \text{if } \mathcal{C}_{ij} > \delta \\ 0, & \text{otherwise} \end{cases}$

return $\mathcal{S}$