DS603: Privacy Preserving GNNs

Ananya

Abstract—**The main aim of this project is to study the problem of learning Graph Neural Networks (GNNs) with Differential Privacy (DP). I will be conducting a study on GAP, a privacy-preserving GNN architecture that ensures both edge-level and node-level differential privacy for training and inference over sensitive graph data using aggregation perturbation. I will be analyzing the effect of different graph centrality measures on the accuracy/privacy performance on GAP-NDP (GAP's node-level Private Graph), perform different attacks (Link Inference attack) and draw some inferences based on the results.**

GAP's new architecture is composed of three separate modules: (i) the encoder module, where we learn private node embeddings without relying on the edge infor- mation; (ii) the aggregation module, where we compute noisy aggregated node embeddings based on the graph structure; and (iii) the classification module, where we train a neural network on the private aggregations for node classification without further querying the graph edges.

I would be evaluating accuracy/privacy performance over three real-world graph datasets, namely Facebook, Reddit, and Amazon.

Index Terms—**Differential Privacy; aggregation-perturbation; Graph Neural Networks; Node-level Privacy; Edge-level Privacy**

I. Introduction

Growing interest in adapting deep learning models for graph-structured data in recent years has popularised the idea of Graph Neural Networks (GNNs) [1]. In a variety of graph-based learning tasks, such as node classification [2], link prediction [3], and community discovery [4], GNNs have demonstrated superior performance in a wide range of applications in the social sciences, biology, molecular chemistry, and so forth. However, the majority of real-world graphs associated with people or activities involving humans, like social and economic networks, are frequently sensitive and may contain personal information. For instance, a person's buddy list, profile details, likes and comments, etc., may be private to the user on a social network. It is crucial to provide privacy-preserving GNN models for applications that rely on graphs to access users' personal data in order to meet users' privacy expectations and comply with current legal data protection rules.

We look into the issue of constructing privacy-preserving GNNs for private, sensitive graphs in light of these privacy concerns. Our objective is to use the Differential Privacy (DP) framework to safeguard the delicate graph structure and other related data [5]. Edge-level and node-level DP have both been characterised as DP variations in the context of graphs [6].

II. Background and Related Works

In a number of GNN learning settings, there have recently been attempts to employ DP to establish formal privacy guarantees.

In a distributed learning environment, where node features and labels are private but training the GNN is federated by a central server with access to graph edges, authors [7] offer a locally private GNN model. Their approach, however, is inapplicable to situations when the graph edges are private. Wu et al. [8] proposal for an edge-level DP learning approach for GNNs involves directly perturbing the input graph with either EdgeRand or the Laplace mechanism (called LapGraph). The generated noisy graph is then used to train a GNN. The node-level privacy setting cannot be easily added to their method, though. By modifying the PATE architecture, Olatunji et al. [9] present a node-level private GNN taking into account a centralised learning environment. However, their dependence on public graph data restricts the applicability of their method.

According to the needs of the application, GAP [10] is the first method to offer privacy guarantees at the edge or node level. GAP's method, in contrast to others, does not rely on publicly available information, may use multi-hop aggregations beyond first-order neighbours, and provides inference privacy at no extra cost.

There is still lot of work needed to improve the GAP method and make it more generalized.

III. System Model

Facebook: The UIUC student social network on Facebook, obtained in September 2005, is included in this dataset. Facebook members are represented as nodes, and friends are indicated by edges. Predicting the class year of users is the task, and each node (user) includes the following attributes: student/faculty status, gender, major, minor, and housing status.

Reddit: This dataset consists of a collection of comments made on Reddit, where each node represents a comment and an edge shows whether the same user posted on more than one article. Predicting the community (subreddit) that a post belongs to requires the extraction of node features based on the embedding of the post contents.

Amazon: This is the largest dataset I will be using in this study which depicts the Amazon product co-purchasing network, where nodes stand for items sold on Amazon and edges show if two items were bought at the same time. The aim is to predict the category of the products using node characteristics, which are bag-of-words vectors of the product description followed by PCA.

To reproduce paper's result I will be using my own local machine.

IV. Algorithms

This section will be updated as the project progresses.

Since GAP's architecture consists of 3 modules, namely: **the encoder module, the aggregation module an the classification module**

1) **Encoder module**, in which a pre-trained encoder to extract lower-dimensional node features without depending on the graph structure. Without utilising the private graph structure, this module transforms the supplied node features into a lower-dimensional representation.
2) **Aggregation module** in which uses aggregation perturbation to privately compute multi-hop aggregated node embeddings using the graph edges and the encoded features. This module uses the aggregation perturbation strategy, or introducing noise to the output of each aggregation step, to recursively compute private multi-hop aggregations using the encoded low-dimensional node attributes.
3) **Classification module**, in which a neural network is trained on the aggregated data for node classification without further examining the graph edges. Without further querying the edges, this module predicts the corresponding labels using the privately aggregated node attributes.

By adding calibrated Gaussian noise to the output of the aggregation function, GAP retains edge privacy and can successfully mask the existence of a single edge (edge-level privacy) or a collection of edges (node-level privacy). This method is inspired by the fact that changing an edge's destination node's neighbourhood aggregation function virtually translates to changing the sample used in the function. Therefore, we may effectively conceal the existence of a single edge, which assures edge-level privacy, or a group of edges, which is required for node-level privacy, by adding the right amount of noise to the aggregation function. However, in order to fully ensure node-level privacy, we also need to secure node features and labels in addition to the edges. This is easily accomplished by training EM and CM using common DP learning methods, such as DP-SGD.

To convert the original node features into an intermediate representation that is supplied to AM, GAP employs a multi-layer perceptron (MLP) model as an encoder. This module's major objective is to make AM's input less dimensional because the amount of Gaussian noise that is added to the aggregations increases as the dimensionality of the data increases. Therefore, minimising the dimensionality aids in improving DP's aggregate value.

GAP's privacy mechanism The aggregation perturbation approach—which uses the Gaussian mechanism to inject stochastic noise to the output of the aggregation function proportional to its sensitivity—is the suggested technique for maintaining the privacy of graph edges in AM. This method is inspired by the fact that changing an edge's destination node's neighbourhood aggregation function virtually translates to changing the sample used in the function. Consequently, we may effectively conceal the existence of a single edge, which assures edge-level privacy, or a group of edges, which is required for node-level privacy, by adding the right amount of noise to the aggregation function. Nevertheless, in order to fully ensure node-level privacy, we also need to secure node features and labels in addition to the edges. This is easily accomplished by training EM and CM using common DP learning methods, such as DP-SGD.

By assembling individual noisy aggregation steps, the GAP technique can take use of multi-hop aggregations. Since it is simple to determine the sensitivity of a single-step aggregation, AM applies the Gaussian mechanism immediately following each aggregation step, preventing the interdependence between node embeddings from expanding. The inference of a node depends on the aggregated data from its neighbours, which is privately computed by AM, hence GAP also offers inference privacy. GAP ensures inference-time privacy because the subsequent CM only post-processes these private aggregations.

The industry standard for evaluating the privacy assurances of algorithms that process sensitive data is differential privacy (DP). Informally, DP demands that, regardless of whether an individual's data are present in the dataset, the algorithm's output distribution be roughly the same. As a result, a foe with access to all except the target individual's data is unable to determine whether the target's record is present in the input data. The following is the official definition of DP.

Definition 1 (Differential Privacy [11]) $\epsilon > 0$ and $\delta > 0$, a randomized algorithm $\mathcal{A}$ satisfies (ϵ, δ)-differential privacy, iffor all possible pairs of adjacent datasets X and X' differing by at most one record, denoted as $X \sim X'$, and for any possible set of outputs $S \subseteq \operatorname{Range}(\mathcal{A})$, we have:

$$\operatorname{Pr}[\mathcal{A}(X) \in S] \leq e^{\epsilon} \operatorname{Pr}\left[\mathcal{A}\left(X'\right) \in S\right]+\delta$$

Here, the parameter ϵ is called the privacy budget (or privacy cost) and is used to tune the privacy-utility trade-off of the algorithm: a lower privacy budget leads to stronger privacy guarantees but reduced utility. The parameter δ is informally treated as a failure probability, and is usually chosen to be very small.

Definition 2(Rényi Differential Privacy). A randomized algorithm $\mathcal{A}$ is $(\alpha, \epsilon) - RDP$ for $\alpha > 1, \epsilon > 0$ if for every adjacent datasets $X \sim X'$, we have $D_\alpha\left(\mathcal{A}(X) \| \mathcal{A}\left(X'\right)\right) \leq \epsilon$, where $D_\alpha(P \| Q)$ is the Rényi divergence of order α between probability distributions P and Q defined as:

$$D_\alpha(P \| Q)=\frac{1}{\alpha-1} \log \mathbb{E}_{x \sim Q}\left[\frac{P(x)}{Q(x)}\right]^\alpha$$

As RDP is a generalization of DP, it can be easily converted back to standard $(\epsilon, \delta))$-DP using the following proposition:

Proposition 1 If $\mathcal{A}$ is an $(\alpha, \epsilon) - RDP$ algorithm, then it also satisfies $(\epsilon + \log(1/\delta)/\alpha - 1, \delta)$-DP for any $\delta \in (0, 1)$.

A basic method to achieve RDP is the Gaussian mechanism, where Gaussian noise is added to the output of the algorithm we want to make private. Specifically, let $f : X \rightarrow \mathbb{R}^d$ be the non-private algorithm taking a dataset as input and outputting a d-dimensional vector. Let the sensitivity of f be the maximum L_2 distance achievable when applying $f(\cdot)$ to adjacent datasets

X and X' as $\Delta_f = \max_{X \sim X'} \|f(X) - f(X')\|_2$. Then, adding Gaussian noise with variance σ^2 to f as $\mathcal{A}(X) = f(X) + \mathcal{N}(\sigma^2 \mathbb{I}_d)$, with $\mathbb{I}_d$ being $d \times d$ identity matrix, yields an (α, ϵ) RDP algorithm for all $\alpha > 1$ with $\epsilon = \Delta_f^2 \alpha / 2\sigma^2$.

Node level Privacy Coupled with aggregation perturbation, the proposed GAP architecture offers edge-level privacy by default. But, if we have bounded-degree graphs—i.e., the degree of each node should be bounded above by a constant D—it is easily extensible to provide node-level privacy assurances as well. This permits to bound the sensitivity of the aggregation function in the PMA process when adding/removing a node, as in this situation each node can influence at most D other nodes. When the input graph contains nodes with extremely high degrees, neighbour sampling can be used to asynchronously sample up to D neighbours for each node.

For bounded-degree graphs, adding or removing a node corresponds (in the worst case) to adding or removing D edges. Therefore, our PMA mechanism also ensures node-level privacy, albeit with increased privacy costs compared to the edge-level setting.

However, since the node features and labels are also private under node-level DP, both EM and CM need to be trained privately as they access node features/labels. To this end, we can simply use standard DP-SGD or any other differentially private learning algorithm for pre-training the encoder as well as training CM with DP.

Training privacy The node-level privacy guarantees of the PMA mechanism and the GAP training algorithm are as follows.

Theorem 1 Given the maximum degree $D \geq 1$, maximum hop $K \geq 1$, and noise variance σ^2, Algorithm 1 (PMA mechanism) satisfies node-level $(\alpha, DK\alpha/2\sigma^2)$-RDP for any $\alpha > 1$.

Proposition 2 For any $\alpha > 1$, let encoder pre-training (Step 1 of Algorithm 2) and CM training (Step 5 of Algorithm 2) satisfy $(\alpha, \epsilon_1(\alpha)) - RDP$ and $(\alpha, \epsilon_5(\alpha)) - RDP$, respectively. Then, for any $0 < \delta < 1$, maximum hop $K \geq 1$, maximum degree $D \geq 1$, and noise variance σ^2, Algorithm 2 satisfies node-level (ϵ, δ) DP with $\epsilon = \epsilon_1(\alpha) + \epsilon_5(\alpha) + DK\alpha/2\sigma^2 + \log(1/\delta)/\alpha - 1$.

Note that in Proposition 2, we cannot optimize α in closed form as we do not know the precise form of $\epsilon_1(\alpha)$ and $\epsilon_5(\alpha)$. However, in our experiments, we numerically optimize the choice of α on a per-case basis.

Algorithm 1: Private Multi-hop Aggregation

```
Input  : Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with adjacency matrix $\mathbf{A}$; initial
         normalized features $\check{\mathbf{X}}^{(0)}$; max hop $K$; noise variance $\sigma^2$;
Output : Private aggregated node feature matrices $\check{\mathbf{X}}^{(1)}, \dots, \check{\mathbf{X}}^{(K)}$

1 for $k \in \{1, \dots, K\}$ do
2     $\mathbf{X}^{(k)} \leftarrow \mathbf{A}^T \cdot \check{\mathbf{X}}^{(k-1)}$                 // aggregate
3     $\widetilde{\mathbf{X}}^{(k)} \leftarrow \mathbf{X}^{(k)} + \mathcal{N}(\sigma^2 \mathbb{I})$          // perturb
4     for $v \in \mathcal{V}$ do
5         $\check{\mathbf{X}}_v^{(k)} \leftarrow \widetilde{\mathbf{X}}_v^{(k)} / \|\widetilde{\mathbf{X}}_v^{(k)}\|_2$     // normalize
6     end
7 end
8 return $\check{\mathbf{X}}^{(1)}, \dots, \check{\mathbf{X}}^{(K)}$
```

Algorithm 2: GAP Training

```
Input  : Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with adjacency matrix $\mathbf{A}$; node
         features $\mathbf{X}$; node labels $\mathbf{Y}$; max hop $K$; noise variance $\sigma^2$;
Output : Trained model parameters
         $\{\mathbf{\Theta}^\star_{\text{enc}}, \mathbf{\Theta}^{\star(0)}_{\text{base}}, \dots, \mathbf{\Theta}^{\star(K)}_{\text{base}}, \mathbf{\Theta}^\star_{\text{comb}}, \mathbf{\Theta}^\star_{\text{head}}\}$;

1 Pre-train EM (Eq. 3) to obtain $\mathbf{\Theta}^\star_{\text{enc}}$.
2 Use the pre-trained encoder (Eq. 5) to obtain encoded features $\mathbf{X}^{(0)}$.
3 Row-normalize the encoded features (Eq. 6) to obtain $\check{\mathbf{X}}^{(0)}$.
4 Use Algorithm 1 to obtain private aggregations $\check{\mathbf{X}}^{(1)}, \dots, \check{\mathbf{X}}^{(K)}$.
5 Train CM (Eq. 10-12) to get $\mathbf{\Theta}^{\star(0)}_{\text{base}}, \dots, \mathbf{\Theta}^{\star(K)}_{\text{base}}, \mathbf{\Theta}^\star_{\text{comb}}, \mathbf{\Theta}^\star_{\text{head}}$.
6 return $\{\mathbf{\Theta}^\star_{\text{enc}}, \mathbf{\Theta}^{\star(0)}_{\text{base}}, \dots, \mathbf{\Theta}^{\star(K)}_{\text{base}}, \mathbf{\Theta}^\star_{\text{comb}}, \mathbf{\Theta}^\star_{\text{head}}\}$
```

V. Experiments and Results

`Requirements: Python version` $\geqslant$ `3.9, Anaconda, PyTorch-Geometric 2.1.0, PyTorch 1.12.1`

Following commands were run for environment set-up on MACOSX (M1) without CUDA, stable version of PyG was installed:

```
$ conda create -n venv python=3.9
$ conda activate venv
$ pip3 install -r requirements.txt
$ pip install git+https://github.com/yuxiangw/autodp
$ conda install -y clang_osx-arm64 clangxx_osx-arm64
  gfortran_osx-arm64
$ MACOSX_DEPLOYMENT_TARGET=12.3 CC=clang CXX=clang++
  python -m pip --no-cache-dir install torch
  torchvision torchaudio
$ python -c "import torch; print(torch.__version__)"
  #---> (Confirm the version is 1.11.0)
$ MACOSX_DEPLOYMENT_TARGET=12.3 CC=clang CXX=clang++
  python -m pip --no-cache-dir install
  torch-scatter -f https://data.pyg.org/whl/
  torch-1.11.0+${cpu}.html
$ MACOSX_DEPLOYMENT_TARGET=12.3 CC=clang CXX=clang++
  python -m pip --no-cache-dir install
  torch-sparse -f https://data.pyg.org/whl/
  torch-1.11.0+${cpu}.html
$ MACOSX_DEPLOYMENT_TARGET=12.3 CC=clang CXX=clang++
  python -m pip --no-cache-dir install
  torch-geometric
```

OR the Stable version of PyG can also be installed in the following way:

```
$ conda install pyg -c pyg
```

Now execute the following commands to replicate the results:

```
$ python experiments.py --generate
$ sh jobs/experiments.sh
```

or you can also parallelize the runs by:

```
Open "core/jobutils/scheduler.py"
Add the following import: "from dask.distributed
import LocalCluster"
Replace lines 45-49 with:
"cluster = LocalCluster(processes=True)"
Run experiments with:
"python experiments.py --generate --run --scheduler
sge"
```

After running the command: `python experiments.py --generate`, the file `jobs/experiments.sh` containing the commands to run all the experiments is created. The `jobs/experiments.sh` has 1227 lines. Each line corresponds to a distinct run. Each run has 10 trials. This is the visualization of 20 runs on the facebook dataset. This is the link to the different panels for the system metrics: WanDB

The `results.ipynb` file generates the following graph after completing the training on the datasets. Here is a description of the datasets:

Dataset	Nodes	Edges	Degree	Features	Classes
Facebook	26,406	$2,117,924$	62	501	6
Reddit	116,713	$46,233,380$	209	602	8
Amazon	$1,790,731$	$80,966,832$	22	100	10

TABLE I: Dataset

Using Rényi Differential Privacy, it is examined that the formal privacy guarantees provided by GAP and empirically assessed its accuracy-privacy performance using Facebook data. It is noted that GAP consistently outperforms a naive (privately trained) MLP model that ignores the graph's structure information, outperforming the competing baselines at (very) low privacy budgets under both edge-level DP and node-level DP.

Privacy Level	Method	$\epsilon(PrivacyCost)$	Facebook
None	GAP - ∞	∞	80.0 ± 0.48
None	SAGE - ∞	∞	83.2 ± 0.68
Edge	GAP-EDP	4	76.3 ± 0.21
Edge	SAGE-EDP	4	50.4 ± 0.69
Edge	MLP	0	50.8 ± 0.17
Node	GAP-NDP	8	63.2 ± 0.35
Node	SAGE-NDP	8	37.2 ± 0.96
Node	MLP-DP	8	50.2 ± 0.25

TABLE II: Trade-offs between Privacy and Accuracy

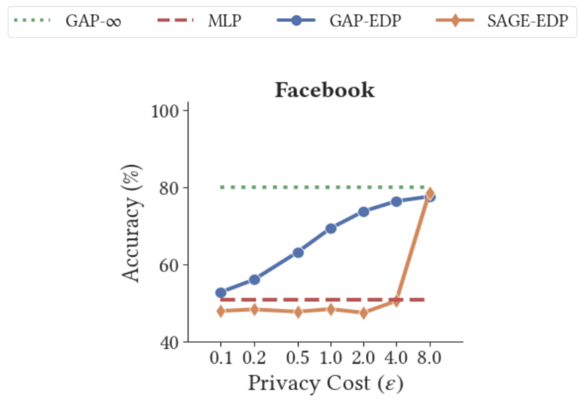


Fig. 1: Accuracy vs. privacy cost of edge-level private algorithms

Training and evaluation details Authors train the non-private and edge-level private methods using the Adam optimizer over 100 epochs with full-sized batches. For the node-level private algorithms (GAP-NDP, SAGE-NDP, MLP-DP), they used DP-Adam with maximum gradient norm set to 1, and train each model for 10 epochs with a batch size of 256, 2048, 4096 on Facebook, Reddit, and Amazon, respectively. For GAP models, they used the same parameter setting for training both the encoder and classification modules. We train all the methods with a learning rate of 0.01 and repeat each combination of possible hyperparameter values 10 times. We pick the best performing model based on validation accuracy, and report the average test accuracy with 95% confidence interval calculated by bootstrapping with 1000 samples.

Privacy accounting and calibration Privacy budget accounting is done via the Analytical Moments Accountant. We numerically calibrate the noise scale (i.e., the noise standard deviation σ divided by the sensitivity) of PMA (GAP-NDP), DP-SGD (for GAP-NDP) and the Gaussian mechanism (for inference privacy in SAGE-NDP) to achieve the desired (ϵ, δ)-DP. We report results for several values of ϵ, while δ is set to be smaller than the inverse number of private entities (i.e., edges for edge-level privacy, nodes for node-level privacy). For GAP-NDP, we use the same noise scale for perturbing the gradients (in DP-SGD) and the aggregations (in PMA and Gaussian mechanisms).

They have used the `autodp` library which implements analytical moments accountant, and utilize `Opacus` for training the node-level private models with differential privacy.

Summary of the Improving the Gaussian Mechanism for Differential Privacy: Analytical Calibration and Optimal Denoising paper: In this paper the authors revisit the Gaussian mechanism and show that the original analysis has several important limitations.The authors address these limitations by developing an optimal Gaussian mechanism whose variance is calibrated directly using the Gaussian cumulative density function instead of a tail bound approximation. The authors also propose to equip the Gaussian mechanism with a post-processing step based on adaptive estimation techniques by leveraging that the distribution of the perturbation is known. This paper's experiments show that analytical calibration removes at least a third of the variance of the noise compared to the classical Gaussian mechanism, and that denoising dramatically improves the accuracy of the Gaussian mechanism in the highdimensional regime. Well-known mechanisms in this class are the Laplace and Gaussian mechanisms. A numerical evaluation provided in Section 5.1 of this paper showcases the advantages of this paper's optimal calibration procedure. The second improvement equips the Gaussian perturbation mechanism with a post-processing step which denoises the output using adaptive estimation techniques from the statistics literature. This paper's contribution is to compile relevant results scattered throughout the literature in a single place and showcase their practical impact in synthetic (Section 5.2) and real (Section 5.3) datasets, thus providing useful pointers and guidelines for practitioners.

Let X be an input space equipped with a symmetric neighbouring relation $x'x'$. Let $\epsilon 0$ and $\delta \in [0,1]$ be two privacy parameters. A natural question one can ask about this result is whether this value of σ provides the minimal amount of noise required to obtain (ϵ, δ)-DP with Gaussian perturbations.

The Gaussian mechanism is an essential building block used in multitude of differentially private data analysis algorithms.

This paper's analysis reveals that the variance formula for the original mechanism is far from tight in the high privacy regime ($\epsilon \rightarrow 0$) and it cannot be extended to the low privacy regime ($\epsilon \rightarrow \infty$). The authors address these limitations by developing an optimal Gaussian mechanism whose variance is calibrated directly using the Gaussian cumulative density function instead of a tail bound approximation. The authors also propose to equip the Gaussian mechanism with a post-processing step based on adaptive estimation techniques by leveraging that the distribution of the perturbation is known.

Problem

- We aim to characterize the privacy cost of each of the GAP's modules (encoder, aggregator, classifier) and find the optimal privacy budget that could be devoted to each of these modules to gain a better privacy-accuracy trade-off.
- We aim to apply transfer learning on GAP so to achieve same privacy budget in less number of epochs. Here are different ways to achieve it:
  1. Source domain is a subset of the target domain
  2. Source domain Target domain
  a. Learn and train the network using structural features
  b. Augment the then add the features and re-train
  3. MLP INIT approach: Transfer learning

- Effectiveness of graph densification. It can be helpful for private training with GAP as we inject noise into the aggregations and thus a larger aggregation set could lead to higher signal to noise ratio, potentially leading to improved performance.

Here are the three stages which I will be accomplishing:

- **1st stage (MLPInit + GAP-EDP)** The proposed MLPInit method does not even need extra data for pre-training. We can simply use the node features to train the PeerMLP of GAP and then use its learned weights to initialize GAP. Under edge-level DP, this approach does not incur any privacy costs. Moreover, weight-scale tuning can also be done for GAP-EDP
- **2nd Stage (Edge Densification)** Add edges only for low degree nodes. Addition of edges requires computation of feature similarity.
- **3rd stage (MLP Init (subset) + EdgeDensification + GAP-NDP-WS)** Source domain is a subset of the target domain Partition – PyMETIS or any other partitioning code can be used for creating a small partition. Alternatively, we can use public data to train the PeerMLP and then fine-tune GAP on private data. This will have no additional privacy costs, neither for edge-level nor node-level settings. But again you need to split the dataset to make sure that you are not using the pre-training nodes for the main training. Test on GAP-NDP

MLPInit [11] Training graph neural networks (GNNs) on large graphs is complex and extremely time consuming. This is attributed to overheads caused by sparse matrix multiplication, which are sidestepped when training multi-layer perceptrons (MLPs) with only node features. MLPs, by ignoring graph context, are simple and faster for graph data, however they usually sacrifice prediction accuracy, limiting their applications for graph data. It is observed that for most message passing-based GNNs, can trivially derive an analog MLP (call this a PeerMLP) with an equivalent weight space, by setting the trainable parameters with the same shapes, making curious about how do GNNs using weights from a fully trained PeerMLP perform? Surprisingly, it is found that GNNs initialized with such weights significantly outperform their PeerMLPs, motivating us to use PeerMLP training as a precursor, initialization step to GNN training.

I have started making changes in the code base:

- Start by making changes to the GAP class. As GAP is not like a standard GNN, the PeerMLP of GAP is basically its classification module trained on node features instead of the aggregate inputs computed by the aggregation module.
- Insert the pre-training step into the fit method. It will then be inherited by GAP-EDP as well.

$$\textbf{MLP:} \quad \mathbf{H}^l = \sigma\left(\mathbf{H}^{l-1}\mathbf{W}^l_{mlp}\right)$$

$$\textbf{GNN:} \quad \mathbf{H}^l = \sigma\left(\mathbf{A}\mathbf{H}^{l-1}\mathbf{W}^l_{gnn}\right)$$

GNN empowers graph learning via message passing. GNNs are powerful for graph while MLPs are computationally efficient.

GNN and MLP have the same trainable weight

- If the dimensions of the hidden layers are the same
- We refer to that MLP as a PeerMLP

Weights from a fully-trained PeerMLP make GNN performs very well.

PeerMLP $f_{mlp}\left(\mathbf{X}; w_{mlp}\right)$; GNN $f_{gnn}\left(\mathbf{X}, \mathbf{A}; w_{mlp}\right)$ w_{mlp} is only trained by PeerMLP

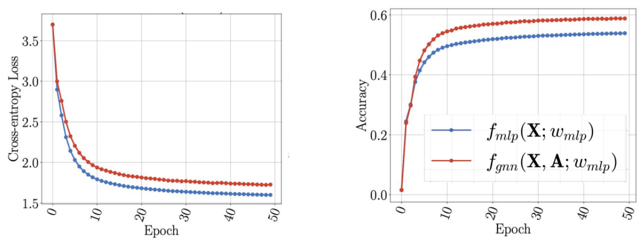


Fig. 2: The loss curve decreases while the accuracy curve increases.

The GNN can be optimized by updating its PeerMLP. For a target GNN,

1) Construct its PeerMLP
2) Train PeerMLP to converge $\rightarrow w^*_{mlp}$
3) Initialize GNN with w^*_{mlp}
4) Fine tune the GNN

```
# f_gnn: graph neural network model
# f_mlp: PeerMLP of f_gnn

# Train PeerMLP for N epochs
for X, Y in dataloader_mlp:
    P = f_mlp(X)
    loss = nn.CrossEntropyLoss(P, Y)
    loss.backward()
    optimizer_mlp.step()

# Initialize GNN with MLPInit
torch.save(f_mlp.state_dict(), "w_mlp.pt")
f_gnn.load_state_dict("w_mlp.pt")

# Train GNN for n epochs
for X, A, Y in dataloader_gnn:
    P = f_gnn(X, A)
    loss = nn.CrossEntropyLoss(P, Y)
    loss.backward()
    optimizer_gnn.step()
```

MLPInit obtain better accuracy, gain performance improvement.

Findings and understandings I tweaked the weight scale which added noise to the different modules in certain ratios. I did an exhaustive search to find the optimal noise to be added to each of the module. weights = [0.5, 1, 2, 4, 8, 16] and then 216 weight scale list existed. However 91 weight scales were unique as their ratios were unique. This was the task of hyperparameter search. Below is a heatmap visualisation for the ratio of the weights and the corresponding accuracy. We have three weights and a final accuracy. We fix the first weight (say 1) and express the other two with respect to the first one. For example, (1,2,4) becomes (2,4) and (2, 1, 8) becomes (0.5,4). In this heatmap matrix, where the rows and columns correspond to the first and second dimensions, respectively, in the resulting representation, and the value is the accuracy.

If the weight list is [1,4,1] and the noise scale is 0.5746, this means that the first and third modules perturb with a noise scale of 0.5746, while the second does with a noise scale of $4 \cdot 0.5746 = 2.2984$. The total epsilon value is a function of the three noise scales: $\epsilon = f(x, y, z)$ where x, y, and z are the first, second, and third noise scales, respectively. So if changing the noise scales (x, y, orz), we would probably get a different epsilon at the end. In the GAP paper, they have considered considered x, y, and z equal, so they had $\epsilon = f(x, x, x)$. Here, we are considering weights, so we have $\epsilon = f(ax, bx, cx)$, where a, b, and c are the weights of the noise scales. Now, calibrating means that we find the value of x such that the total epsilon value becomes equal to the given value.

For instance, when I run `python train.py gap-ndp --dataset facebook --epsilon 8 --encoder-epochs 1 --epochs 1` and the list is [1,4,1], we are finding x such that $8 = f(x, 4x, x)$.

Note that the total epsilon results from the composition of all three modules obtained using composition theorems of Renyi differential privacy (RDP).

We can compute the individual epsilon values of the three modules, but if we sum them together, the result will be bigger than the calculated epsilon for the whole three modules. This is because when you simply add the epsilon values, you are using the basic composition theorem of DP, which gives you looser bounds than RDP composition.

Another approach would be to compute the epsilon of the first module, then the epsilon of the composition of the first and second modules, and finally the epsilon of the composition of the three modules, and consider the difference between the obtained epsilon values as the additional privacy cost that is incurred at each step. However, this approach might not give a good estimate of the privacy cost of each module, as the first privacy mechanisms usually consume more privacy budgets than the last ones (epsilon is roughly proportional to the square root of the number of compositions).

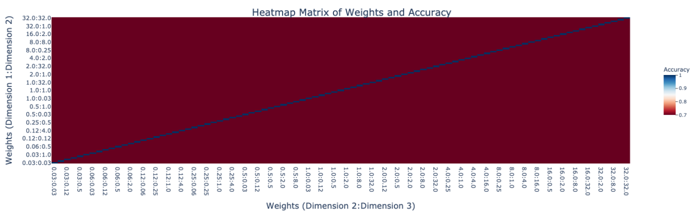


Fig. 3: Heatmap for facebook dataset for the ratio of noise added and the corresponding accuracy

There was a weight list in `core/method/composed.py`. The commands run were with a set values of arguments as follows:

```
$ python train.py gap-ndp --dataset facebook
--epsilon 8 --encoder-epochs 10 --epochs 10
$ python train.py gap-ndp --dataset reddit
--epsilon 8 --encoder-epochs 10 --epochs 10
$ python train.py gap-ndp --dataset amazon
--epsilon 8 --encoder-epochs 10 --epochs 10
```

And after checking for each weight scale, below are the results with the best test accuracy:
For Facebook dataset, the noises to the modules were added in the ratio of 1:1:4
For Reddit dataset, the noises to the modules were added in the ratio of 1:1:2
For Amazon dataset, the noises to the modules were added in the ratio of 1:2:1

For the facebook dataset, It seems that the third weight is quadruple the first two. The First 2 modules are more sensitive towards noise. That actually makes sense because EM and AM prepare data for CM, so the more accurate they are, the better.

Here is the per-epoch analysis for the optimal choice of the weight scales for each of the datasets, keeping the value of the other hyper parameters same.

epoch_number	noise_scale	test_acc	train_acc	val_acc
1	0.5424	75.87	75.24	76.12
2	0.5466	76.22	76.23	76.20
3	0.5493	76.40	76.35	76.43
4	0.5514	76.48	76.42	76.40
5	0.5532	76.71	76.46	76.49
6	0.5548	76.56	76.44	76.47
7	0.5562	76.46	76.51	76.50
8	0.5576	91.91	76.61	76.60
9	0.5589	91.98	76.52	76.60
10	0.5601	92.37	76.60	76.60

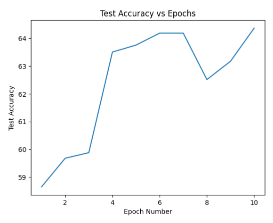
TABLE III: Amazon dataset

epoch_number	noise_scale	test_acc	train_acc	val_acc
1	0.9033	90.76	87.43	91.49
2	0.9048	91.78	92.26	91.82
3	0.9063	91.90	92.23	91.95
4	0.9078	91.79	92.55	92.07
5	0.9093	92.30	92.58	92.08
6	0.9108	92.17	92.61	92.04
7	0.9123	92.07	92.61	92.07
8	0.9138	92.25	92.86	92.13
9	0.9152	92.07	92.57	92.19
10	0.9167	92.37	92.70	92.20

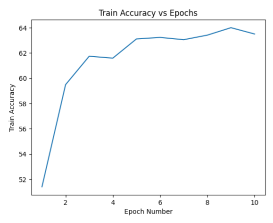
TABLE IV: Reddit dataset

epoch_number	noise_scale	test_acc	train_acc	val_acc
1	0.9083	58.65	51.40	58.29
2	0.9146	59.68	59.49	59.84
3	0.9208	59.88	61.74	60.90
4	0.9268	63.50	61.59	62.27
5	0.9325	63.75	63.11	62.31
6	0.9382	64.18	63.23	62.53
7	0.9437	64.18	63.05	63.03
8	0.9492	62.51	63.41	62.72
9	0.9544	63.17	64.00	62.04
10	0.9596	64.36	63.50	62.53

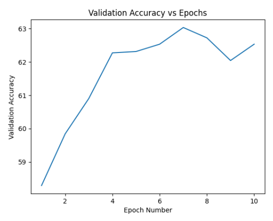
TABLE V: Facebook dataset



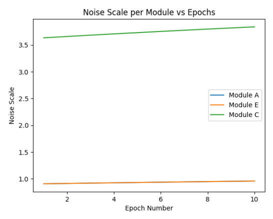
(a) Test accuracy v/s epochs



(b) Train accuracy v/s epochs

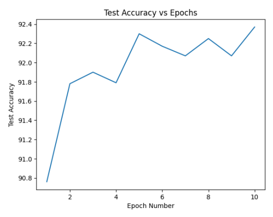


(c) Validation accuracy v/s epochs

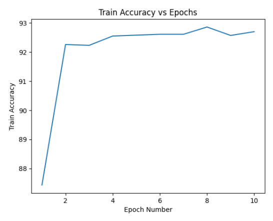


(d) Noise scale per module v/s epochs

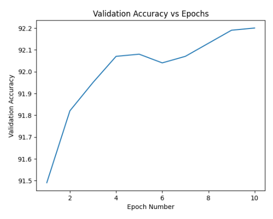
Fig. 4: Facebook dataset



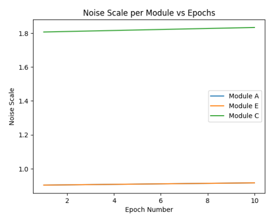
(a) Test accuracy v/s epochs



(b) Train accuracy v/s epochs

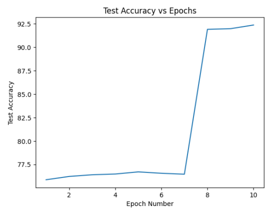


(c) Validation accuracy v/s epochs

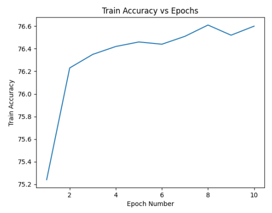


(d) Noise scale per module v/s epochs

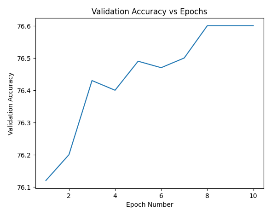
Fig. 5: Reddit Dataset



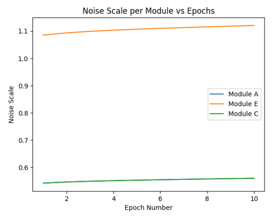
(a) Test accuracy v/s epochs



(b) Train accuracy v/s epochs



(c) Validation accuracy v/s epochs



(d) Noise scale per module v/s epochs

Fig. 6: Amazon dataset

VI. Conclusion

It has been found that tuning the weight scales for GAP-NDP method resulted in increase of accuracy. There some other novel methods to be also implemented to see their effect on accuracy. We saw that weight scale tuning increased accuracy for facebook dataset by 1.33%, reddit dataset by 0.7% and amazon dataset by 0.3%. By applying Graph transfer learning, and amalgation of certain methods discussed in Experiment section will most likely improve the privacy-accuracy trade-off. Notably, these techniques and methods are easy to implement and employ in real applications to improve privacy-accuracy trade-off in GAP architecture.

References

[1] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2009.

[2] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," 2016. [Online]. Available: https://arxiv.org/abs/1609.02907

[3] M. Zhang and Y. Chen, "Link prediction based on graph neural networks," 2018. [Online]. Available: https://arxiv.org/abs/1802.09691

[4] Z. Chen, X. Li, and J. Bruna, "Supervised community detection with line graph neural networks," 2017. [Online]. Available: https://arxiv.org/abs/1705.08415

[5] C. Dwork, "Differential privacy: A survey of results," in *Proceedings of the 5th International Conference on Theory and Applications of Models of Computation*, ser. TAMC'08. Berlin, Heidelberg: Springer-Verlag, 2008, p. 1–19.

[6] S. Raskhodnikova and A. Smith, "Differentially private analysis of graphs," *Encyclopedia of Algorithms*. [Online]. Available: https://par.nsf.gov/biblio/10092785

[7] S. Sajadmanesh and D. Gatica-Perez, "Locally private graph neural networks," 2020. [Online]. Available: https://arxiv.org/abs/2006.05535

[8] F. Wu, Y. Long, C. Zhang, and B. Li, "Linkteller: Recovering private edges from graph neural networks via influence analysis," 2021. [Online]. Available: https://arxiv.org/abs/2108.06504

[9] I. E. Olatunji, T. Funke, and M. Khosla, "Releasing graph neural networks with differential privacy guarantees," 2021. [Online]. Available: https://arxiv.org/abs/2109.08907

[10] S. Sajadmanesh, A. S. Shamsabadi, A. Bellet, and D. Gatica-Perez, "Gap: Differentially private graph neural networks with aggregation perturbation," 2022. [Online]. Available: https://arxiv.org/abs/2203.00949

[11] X. Han, T. Zhao, Y. Liu, X. Hu, and N. Shah, "Mlpinit: Embarrassingly simple gnn training acceleration with mlp initialization," 2023.