Federated Learning on Non-disjoint Graphs

by

Fatemeh Tavakoli

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

Abstract

Federated learning is in widespread use for learning a global model when data is distributed across various distributed clients. In much of the prior work, the data is assumed to consist of independent data points. However, there is often an underlying graph that structures the data points. Such structures emerge in data on social networks, content recommendations, bank transactions data, healthcare data, and other such data where there is a notion of similarity or relation that links data points. Standard federated learning frameworks are not designed specifically for graph data and thus cannot take advantage of graph structure for node classification. We consider federated learning on graph data where a global graph is split among a set of clients, In particular, we consider a non-disjoint split, where there are some nodes that we call anchor nodes, that are present at multiple clients. The learning task is node classification in a semi-supervised scenario where only a small set of nodes have labels. We propose a new federated learning algorithm for non-disjoint graphs that leverages anchor nodes to augment local graph structure for improved node classification. We show through extensive experiments on several graph datasets, that our method outperforms standard methods on the task of node classification.

Preface

This thesis is an original work by Fatemeh Tavakoli under the supervision of Professor Nidhi Hegde at the University of Alberta, Computing Science Department. Some parts of this work may be submitted for publication.

Imagination is more important than knowledge. For knowledge is limited, whereas imagination embraces the entire world, stimulating progress, giving birth to evolution.

– Albert Einstein.

Acknowledgements

I would like to sincerely thank my supervisor, Professor Nidhi Hegde, for her guidance and support throughout my time at the University of Alberta. I appreciate many great lessons I learned from my supervisor.

I am grateful for the support I have received from my parents and sister during my absence from home for over two years. Their unwavering love and care have meant the world to me. I am thankful to have friends who listened to my ideas and motivated me throughout the challenging times.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In many settings of data used for classification or other machine learning tasks, the data is distributed and held at independent organizations rather than at a centralized entity. In such cases, the Federated Learning (FL) [19] framework has been used to learn a global model by sharing locally trained models. Such a framework consists of several clients with their own datasets, that learn local models which are then aggregated by a server into a global model. In most cases, it is assumed that the data points are independent. However, in reality, these data points are often connected through an underlying graph. Examples of such data include social network analysis data, content recommendations, healthcare data, and other such data where there is a notion of similarity or relation that links the data points. The graph data may be distributed across different entities such that the graph is not partitioned but there are some nodes in the graph that appear at more than one entity. Such a non-disjoint split of a graph into subgraphs may happen in scenarios such as multiple social networks where people may be in more than one network but may have different connections in each. We may encounter such a situation in finance and banking where an individual may have accounts at different banks, or in healthcare where people may visit different hospitals. In such settings, the objective may be node classification. Standard FL frameworks are not designed specifically for graph data and thus cannot take advantage of the graph structure for node classification.

We consider a FL setting where various clients hold data that have a sub-

graph structure, as part of some unknown global structure. The objective is to perform node classification. When a single global graph is available, graph neural network models may be applied to this task. However, when subgraphs of a graph are distributed, we do not have access to the underlying global structure of the graph in which nodes are connected. What a graph neural network learns from all the subgraphs might be very different and incomplete from what it would learn from the global graph including all these subgraphs. This can be seen largely in situations where subgraph data of a large graph are owned by several institutions (or silos). For several reasons, like privacy, or graphs being too large, data owners may choose to keep their subgraphs on different servers and train a model on the decentralized data without sharing the raw data.

In graph neural networks especially when class labels are rare, the structure and connections around nodes have a significant role in deciding a node's class. This is more significant for so-called *anchor* nodes, nodes that are owned by more than one client, for which each client has different connections (partial connections). This creates incompleteness in terms of information coming from its neighboring nodes. This also may create disagreement among clients on the node embedding or class since the anchor node might end up with very different embeddings at different clients.

We propose a learning framework for this scenario of distributed subgraphs with the presence of anchor nodes. We leverage information about anchor nodes that are across different clients to design a new framework for non-disjoint subgraph federated learning. Our algorithm is built on top of the baseline federated learning algorithms such as FedAvg [19] and FedProx [13]. In our framework called Fed-GALA, each client uses a graph convolutional network (GCN) model for local training helped with global parameter updates from a central server. The clients then leverage information about the node embeddings of anchor nodes to augment local structure which helps the local GCN [12] learn a more globally structured node embedding. These embeddings are then used to make node classification with fairly high accuracy. Our proposed approach has little communication and computation overhead.

1.1 The Problem

Node data points in a graph are not independent. Links that connect any two nodes are an indication of dependence between the respective nodes. This means a node's neighboring nodes and structure alongside the node's attributes define the role or the behavior of the node in a graph. The amount of influence of the neighboring nodes is not the same; some may have a higher influence and some less. The learning task is also an important factor for determining the influence and importance of the neighboring nodes. A Graph Neural Network [25] handles this task by passing the neighboring node features through a neural network where the aggregated output is an embedding prediction defining the contribution of the neighboring nodes. The contribution of the respective node feature values is also determined using a neural network that connects the node to itself. This structure is explained with details in Section 2.2.2. This highlights the importance of the neighboring nodes and the structure around a node in defining its embeddings and predictions. However, having a graph split over several data owners creates several challenges. This especially emerges in the nodes that exist at multiple subgraphs of a larger graph. We call these anchor nodes.An anchor node has the same feature vector at all of its instances (in various subgraphs), but the local structures will be different. In other types of data, duplicated data points may also exist at different places. In graphs, however, we also have structural information. A data point in a graph is defined by its feature values and its neighboring nodes. Even if two nodes have the same exact feature values, they can be distinguished by their neighboring nodes. Two nodes are exactly the same if they have the same feature values and they have the same neighboring nodes.

When a graph is split, nodes at the cuts lose their structural information. We call these nodes anchor nodes, and they are duplicated in the subgraphs formed at the split, but their entire neighborhood structure is not replicated. However, having information about anchor nodes that have different structural information at the different subgraphs can help in training local and global models.

However, even knowing that another data owner has the same anchor node, and sharing neighboring node information with other owners may violate privacy policies.

For example, consider Figures 1.1 and 1.2. Figure 1.2, may represent subgraphs that are owned by different data owners or companies. If there was no privacy concerns, data owners could simply share their subgraphs and connect them using the anchor nodes to create the global view of Figure 1.1. In this example, node number 1 is an anchor node. Figure 1.2 illustrates a realistic scenario. In this figure, node color is a reflection of the true class, and numbers distinguish individual nodes Anchor node 1 has the same feature values at subgraphs A and B. In subgraph A it is only connected to a red node, and in subgraph B, it is connected to 4 nodes, 3 of which are blue. We have shown node 1 with the color gray, pointing to the possibility that we might not know its class. If node 1's features are not well representative of its class, the model trained on subgraph A might classify node 1 as red, and the model trained on subgraph B might classify this node as blue considering its neighbors. However, if the model is trained on the global graph in Figure 1.1, it will probably classify it as blue given the portion of its blue neighboring nodes. There are other characteristics of node 1 that vary considerably in subgraphs A and B. For example, the degree of node 1 is considerably different in subgraph A and subgraph B. If one could only see subgraph A, it would be reasonable to conclude that node 1 has a low probability to create connections with other nodes. However, looking at subgraph B, we can see node 1 is probably a central node with many connections. Thus, the view of the characteristics and neighborhood of a data point in a graph can considerably affect the correctness of the model.

1.2 Motivating Examples

In this section, we mention some realistic and popular examples of this problem to help understand its importance. In this work, we have not used these kinds of datasets as they are usually extremely private or there are not many graph-

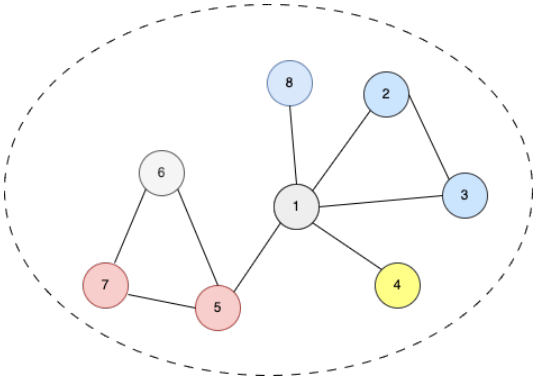


Figure 1.1: The global view of the graph without any privacy concerns.

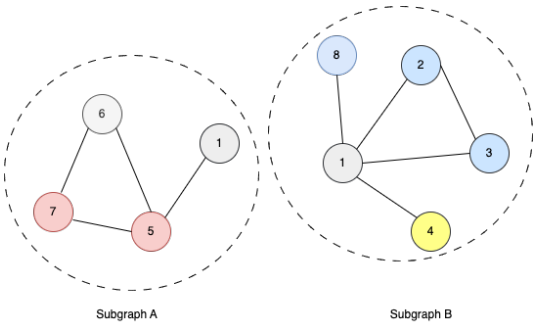


Figure 1.2: The local view of the graph where each subgraph is owned by a different data owner.

structured data available for this use up to our knowledge. We use other publicly available datasets that have characteristics similar to these scenarios.

Social Networks Consider a social network graph where nodes represent individuals, node features are their characteristics, and edges represent friendships and connections in social media. The task here may be classifying the nodes into different groups. For example, we may be interested to know an individual's interests, income and employment, geographical locations, etc. An individual may have accounts on different social networks with different connections. Companies would be able to improve their service if they could analyze and learn that person's behavior across all the social media networks, however, it is not directly feasible. Our method could be a suitable solution for such problems. Graph Neural Networks have been widely used for social network analysis and node classification in works such as [29], where they address the problem of class imbalance in semi-supervised node classification in social networks of the internet of people.

Banking systems Consider a graph where nodes are bank accounts including both customers and retail stores, and edges represent the transactions between accounts. Each bank keeps its graph of transaction data private in its own databases. One person could have accounts at different banks and have different transactions in these banks. It is not possible to collect data from all banks in one place and learn the transaction patterns of people. Knowing how people tend to spend money or do money transactions across all their accounts could improve bank models to suggest better services to each individual person. It also helps to create a unified network of information across all banks to find fraudulent activities. Recently, Graph Neural Networks have been used widely for applications such as Financial fraud detection. Some studies such as [16] use data from online credit payment services in the form of graphs where nodes are users and links represent relations such as fund transfers between users. In this work, the goal is to classify fraudulent nodes.

1.3 Contributions

Our contributions are as follows:

- Our main contribution is designing a framework for collaborative node classification on non-disjoint graphs. We compare our method with two state-of-the-art federated learning algorithms and show its considerably improved performance without posing additional communication overhead.
- We introduce a new weighted averaging algorithm for aggregating parameters in the server that is suitable for semi-supervised training on graph-structured data.
- We introduce a new way for creating non-disjoint subgraphs for subgraph federated learning without the loss of any edges.
- We introduce two other alternative algorithms to our main algorithm where the resources are limited. We show the two alternative algorithms yield improved performance compared to the baseline models.

Chapter 2

Background and related works

We now present some background material and related works on the most important components of our model: The federated Learning models that we use for client collaboration and the Graph Neural Network model we use for classification.

2.1 Federated Learning

Federated learning (FL) is a collaborative machine learning setting where a set of K clients each hold a local dataset, and collaboratively train a global model with the help of a server [10]. The motivation behind this framework is to keep the local data inside the data sources and share information in the form of the trained model rather than raw data. This could be due to privacy reasons, communication costs associated with data transfer, or having distributed computation resources. It is often assumed that the local datasets are disjoint partitions of some global dataset and the objective is to learn a global model.

The clients and server communicate in an iterative process where the server initiates and sends out an initial model to the clients. Clients train their local models on their local data in parallel, then share local model parameters with the server. The server performs aggregation on received model parameters and updates the clients with the aggregated model information. This iterative process enables clients to collaboratively train a common model on all local datasets without sharing raw data.

Cross-silo versus Cross-device FL There are two main variants of FL: cross-silo and cross-device federated learning. Cross-silo FL considers the scenario where the participating clients are companies or owners of large datasets. Some examples of Cross-silo FL are financial risk prediction performed by insurance companies, drug discovery done by collaborative medical centers, and medical data segmentation on several hospitals' data. In contrast, cross-device FL assumes the existence of a large number of clients, such as mobile devices and sensors. The difference between cross-silo and cross-device FL mostly is in the amount of data stored at or owned by individual clients. Also, clients in a cross-device setting might not be available throughout all the training rounds, therefore to simulate this setting, before the start of each round, a small portion of all the clients are chosen for that round. However, it is usually assumed that all the clients in a cross-silo setting are available and will collaborate in all of the training rounds. In our work, we assume our few clients own large portions of the dataset, therefore, we consider it as a cross-silo federated learning setting.

Data heterogeneity in FL Since the introduction of this algorithm, many challenges in the practical implementation of federated learning have been identified. One of the main challenges in federated learning is training on data that is not independent and identically distributed (non-iid) across the clients [10]. Some of the most studied non-iid data settings in FL are label distribution skew and quantity skew [10]. The authors in [9] studied and analyzed different federated learning algorithms in non-iid settings and proposed SkewScout as a decentralized training module for controlling the FL communication based on the accuracy loss of non-iid partitions. The authors in [32] proposed a client selection algorithm to minimize the effect of class imbalance. They proposed to analyze the gradient distribution per class in each client, and the clients with a more uniform per-class gradient distribution are chosen for training. The authors in [26] proposed a constraint federated training formula to control the closeness of training loss between each client and the average loss over all clients to account for class imbalance. These are just a few of

the many successful methods that address heterogeneous data distribution in federated learning.

Data partitioning schemes in FL Data can be partitioned in several ways across clients. Data partitioning by samples refers to when clients have different data points of a dataset [19]. Data partitioned by features is defined when clients have different features of the same data points (entities). In the latter case, studies have proposed exchanging specific intermediate results between clients rather than the model parameters [17]. The problem we consider is a combination of the mentioned two settings, as the clients own partitions of a global dataset, and may have a portion of overlapping data points.

2.1.1 Federated Averaging (FedAvg)

A fundamental algorithm for parameter aggregation at the server is FederatedAveraging (FedAvg) [19]. It assumes a synchronous update scheme during several rounds of communication.

After the server initialization step, at each round t, clients update their model parameters by several local training epochs, and send their new model parameters ($\mathbf{W}_{t+1}^k$ for client k) to the server. The server does a weighted averaging (2.1) on the local model parameters to generate a global model $\mathbf{W}_{t+1}$. The server then returns the global parameters to clients to start the next round of federated training.

The model parameters are aggregated at the server by weighted averaging with the weights corresponding to the number of training samples in each client. For n_k samples at client k and a total number n of samples, the weighted averaging is as follows:

$$\mathbf{W}_{t+1} \leftarrow \sum_{k \in K} \frac{n_k}{n} \mathbf{W}_{t+1}^k \tag{2.1}$$

FedAvg [19] mostly considers a cross-device scenario, where a fraction C of all k clients is randomly selected at the beginning of each round for that iteration of training. This is an essential step when the number of clients

is considerably high. In this setting, it is possible that not all clients have completed their local training, therefore only a fraction of them may take part in each round. However, our setting considers a cross-silo setting of a few clients. Therefore, we assume C to be 1 in the rest of this manuscript.

In Algorithm 1 we show the vanilla FedAvg algorithm applied to the cross-silo setting used in this work.

Algorithm 1 FederatedAveraging (FedAvg) algorithm for cross-silo setting where all available clients participate in every round. E is the number of local epochs and η is the local learning rate. ρ_k is the dataset owned by client k.

Server executes

initialize w_0
for each round $t = 1, 2, ...$ **do**
    $S_t \leftarrow$ available clients
    for each client $k \in S_t$ **in parallel do**
        $w_{t+1}^k \leftarrow \text{ClientUpdate}\ (k, w_t)$
    $w_{t+1} \leftarrow \sum_{k \in K} \frac{n_k}{n} w_{t+1}^k$ ▷ Server Aggregation

ClientUpdate (k, w): ▷ Run on client k

$w_0 \leftarrow w$
for each local epoch e from 1 to E **do**
    $w_e \leftarrow w_{e-1} - \eta \Delta l(w_{e-1}; \rho_k)$
return w_E to server

2.1.2 FedProx

As mentioned before, an important challenge in the practical implementation of federated learning is training on data that is not independent and identically distributed (non-iid) across the clients [10]. A recent and popular algorithm introduced to tackle non-iid data is FedProx [13] which is a generalization and re-parameterization of FedAvg that has been proven to be useful in addressing both the statistical heterogeneity of data and the system heterogeneity of clients.

If the local objective function of a client k in FedAvg is $F_k(.)$, then in FedProx it would be $F_k(w) + \frac{\mu}{2}||\mathbf{W}^k - \mathbf{w}^t||^2$, which has a normalization term added to the local loss. W^k is the local model parameter in training at client k, and w^t is the model parameter shared by the server at the round t. The

hyperparameter μ controls the proximal term which controls the impact of local updates on the current local model of the client, by penalizing local models that deviate too much from the global model. The FedProx server uses the same aggregation formula (2.1) as FedAvg.

Algorithm 2 FedProx algorithm for the cross-silo setting where all available clients participate every round. E is the number of local epochs and η is the local learning rate. ρ_k is the local data owned by client k.

Server executes

1: same as FedAvg 1

ClientUpdate (k, w): ▷ Run on client k

2: $\bar{w} \leftarrow w$

3: $w_0 \leftarrow w$

4: **for** each local epoch e from 1 to E **do**

5: $\quad w_e \leftarrow w_{e-1} - \eta \Delta l(w_{e-1}; \rho_k) - \frac{\mu}{2} \Delta ||w_{e-1} - \bar{w}||^2$

return w_E to server

2.2 Machine Learning on Graphs

Machine learning is widely used for analyzing and learning from graph-structured data and networks. Like any other type of data, machine learning tasks can be defined on graphs ranging from supervised to unsupervised. Machine learning tasks on graphs can assume nodes in graphs as data points, or single smaller graphs belonging to a set of graphs as individual data points. The former use case appears in applications such as network analysis and social media, while the latter use case is observed in molecular study and chemistry applications. In the following, we provide a brief taxonomy of standard machine learning tasks on graph data.

2.2.1 Tasks

Community detection While node classification and edge prediction infer missing information about nodes and edges in the graph and can be classified into supervised (or semi-supervised) machine learning tasks, community detection can be referred to as the unsupervised task of clustering on graphs [6]. While in clustering non-graph data, the distance between nodes is used as a

similarity measure, in community detection, graph edges are used as a piece of information to cluster nodes. Each community in a graph consists of nodes that are more likely to have an edge between them. The task here is to find communities in the form of subgraphs of the main graph. Finding communities in large graphs is often used for a better understanding of the various components of a graph, and in practical applications to find groupings of similar nodes.

The Louvain method for community detection first introduced in [1], is designed for fast unfolding and community detection in large graphs. Its objective is to maximize the modularity of subgraphs or communities. Modularity is a measure of the structural density of a subgraph in a graph and is defined as follows:

$$Q_c = \frac{\sum_{in}}{2m} - \left(\frac{\sum_{tot}}{2m}\right)^2 \tag{2.2}$$

where: $\sum_{in}$ is the sum of edge weights between nodes within the community c (each edge is considered twice), and $\sum_{tot}$ is the sum of all the edge weights for nodes in the community (including edges which link to other communities). m is the sum of all edge weights in the graph, and $Q_c \in [-0.5, 1]$. The graph datasets that are used in this thesis are all undirected and unweighted (unit weights).

Communities with higher modularity would have more edges within their community than edges pointing outwards to other communities. Maximizing the modularity is an NP-hard problem, and this method uses a heuristic. The Louvain method consists of two iterative steps:

Let c_i denote the community node i belongs to, and N_i denote the set of i's neighbors, $N_i = \{i : (i, j) \in E\}$. To start, each node is its own community, $c_i = \{i\}, i \in V$. For each node $i \in V$ the Louvain method considers its neighbors that are not in its community, that is, $\{j \in N_i : j \notin c_i\}$ For each such neighbor j, the method measures the modularity gain of removing this node from its current community c_j and adding it to node i's community c_i. If the modularity gain is positive, node j is added to c_{ij}, otherwise, no change is

made. This process continues for all the neighboring nodes of the given node as well as every other node in the graph until no further change happens in the community structures.

Node classification Considering a graph $G = (V, E)$, where V is the set of nodes and E is the set of edges, the node classification task can be defined as predicting the label y_n of a node $v_n \in V$ in the graph. Often we assume that we only have access to the label information of a very small subset of the nodes in G. Node classification is not limited to the described scenario, and can also be performed on several disconnected graphs, or in some cases, one can have access to many labeled nodes [6].

While node classification might seem like any other supervised task in machine learning, it has substantial differences. Nodes in a graph are not independent and identically distributed (iid), whereas, we usually assume in supervised machine learning that data points are independent. In supervised learning tasks on non-graph data, the assumption of iid data is important, as it allows us to generalize the trained models to new unseen data points. However, in the node classification task we model a set of non-iid points by explicitly considering the graph connections between data points in the training process. The graph edges are useful in node classification, where the notion of *homophily* can be assumed. The notion of *homophily* suggests that nodes tend to share similar attributes with their neighbors. Most of the successful neural structures designed for learning from graphs are based on the notion of *homophily* [6]. Many graph neural network architectures have been proposed for this task such as Graph convolutional networks [12] and GraphSAGE [7]. In this thesis, we focus on graph convolutional networks for semi-supervised node classification, where only a small number of nodes have labels. We explain this further in Section 2.2.2.

Link prediction Link prediction or relation prediction is another popular task in graph machine learning with numerous applications such as in recommender systems, drug-side effect prediction, or knowledge graph completion.

In this task, we assume access to a set of known edges (E) in our training set and try to predict the missing edges in the test set. Similar to the node classification task, link prediction also breaks the assumption of iid data points and also can be performed on either a single graph or several disconnected ones.

Node classification and link predictions on graphs are generally performed by training Graph Neural Networks. We give an overview on these models next.

2.2.2 Graph Neural Networks

Tasks such as node classification and link prediction rely on learning model representations or embeddings. There have been many methods proposed in the past for learning these embeddings including spectral methods. We focus on neural networks. These methods essentially rely on a message passing framework, where node information is exchanged over the graph structure and updated using neural networks.

Message Passing Framework

A graph neural network in its simplest form can be defined based on the message-passing framework [5], [6], where neighboring nodes exchange information and influence each other's embeddings. This framework is integrated into graph neural network layers to enable neighboring node information exchange.

A node's embedding (h_u) in a layer of message passing in GNN is updated as shown in (2.3). $N(u)$ is the set of node u's neighboring nodes and k represents a layer in GNN. UPDATE and AGGREGATE functions are differentiable functions of choice. For example, they could be neural networks such as MLP or recurrent networks, or simpler functions such as summation or averaging.

$$h_u^{(k+1)} = UPDATE^{(k)}\big(h_u^{(k)}, AGGREGATE^{(k)}(\{h_v^{(k)}, \forall v \in N(u)\})\big), \quad (2.3)$$

$$= UPDATE(h_u^{(k)}, \mathbf{m}_{N(u)}^{(k)}), \quad (2.4)$$

$h_u^{(k)}$ is the embedding of node u at layer k, while the result of the AGGREGATE function, $m_{N(u)}^{(k)}$ is the message passed from neighboring nodes as in (2.4).

The Basic Graph Neural Network

The most basic form of a graph neural network in the simplified form of the GNN introduced in [20] and [25] as mentioned in [6] is defined in the following equation:

$$h_u^{(k)} = \sigma\big(\ W_{self}^{(k)}\, h_u^{(k-1)} + W_{neigh}^{(k)} \sum_{v \in N(u)} h_v^{(k-1)} + b^k h_u^{(k-1)}\big), \tag{2.5}$$

In (2.5), $W_{self}^{(k)}$ and $W_{neigh}^{(k)}$ are trainable parameters. σ is a non-linear activation function such as RELU and b^k is the bias term, that is usually omitted in equations for simplicity but plays an important role in GNN performance.

2.2.3 Graph Convolutional Networks

We consider the setting of transductive semi-supervised node classification in graph data, where labels are available for a very small subset of nodes, and the task is to classify the unlabeled nodes. The complete dataset in our setting is a graph $G = (V, E)$ with n nodes $v_i \in V$ and edges $(v_i, v_j) \in E$ with $v_i, v_j \in V$, $i, j = \{1, \ldots, n\}$.

Throughout this document we consider only undirected graphs with unit edge weight. The adjacency matrix is denoted by A and includes self-edges for all nodes, a typical assumption in graph neural networks. Each node v_i has a feature vector X_i. Let $\tilde{V}$ denote the set of nodes with labels. A 2-layer Graph Convolutional Network (GCN) for node classification is a powerful framework that is fast and scalable and has shown good results on large graphs [12]. The forward propagation equation for this 2-layer GCN model is as follows:

$$Z = GCN(X, A) = \tilde{A}\ RELU\ (\tilde{A}XW_0)W_1, \tag{2.6}$$

where Z is the matrix of embeddings learned from X, the matrix of node feature vectors. The weights W_0 and W_1 correspond to the first and second layer parameters, respectively. $\tilde{A}$ is the symmetrically normalized adjacency matrix, and D is the degree matrix [12] as shown below.

$$\tilde{A} = D^{-1/2} A D^{-1/2} \tag{2.7}$$

For node classification, a softmax layer is applied on the embeddings Z calculated in (2.6) to get the probability of each class. The classification loss L_{C} is the cross-entropy loss calculated on the labeled nodes, as follows:

$$L_{\mathrm{C}} = - \sum_{Z_i \in Z : v_i \in \tilde{V}} \sum_{l \in L} Y_i \log(\mathrm{softmax}(Z_{il})), \tag{2.8}$$

where Y_i is the label for node v_i, Z_{il} is the l-th element of Z_i, corresponding to class l, and L is the number of classes.

2.2.4 Graph Auto-Encoders

The edge prediction task on graphs is efficiently achieved with self-supervised learning as training samples already exist in the structure of a graph [18]. While there are other methods for edge prediction, we use self-supervised edge prediction using graph auto-encoders (GAE) [11], which uses a graph convolutional network as its encoder. A two-layer GCN as described above is trained as an encoder to generate node embeddings Z.

The decoder is the logistic sigmoid function over the inner product of two node embeddings [11] and predicts the reconstructed adjacency matrix $\hat{A}$ as shown below.

$$\widehat{A} = \sigma(Z\, Z^T),\ \widehat{A_{ij}} = \sigma(z_i * z_j) \tag{2.9}$$

At each epoch of GAE training, the loss function, L_{GAE} shown in (2.10), is defined as a cross-entropy loss over positive and negative edges. Positive edges ($A_{ij} = 1$) are all the existing edges in the graph. Negative edges ($A_{ij} = 0$) are non-existing edges randomly sampled from the graph in an equal number to positive edges.

$$L_{\mathrm{GAE}} = \frac{-1}{|v|} \sum_{i=0}^{N} \sum_{j=i}^{N} A_{ij} \log \widehat{A_{ij}} + (1 - A_{ij}) \log(1 - \widehat{A_{ij}}). \tag{2.10}$$

2.3 Federated Learning on Graphs

We refer to Federated learning on graphs as the setting where clients own graph datasets that are subgraphs of a larger graph. This exact setting can take several forms depending on how the data is distributed among the clients. As in the more common use case, graph data can be distributed based on either nodes, edges, or subgraphs, as described in [8]. We further explain these settings in the next section (related works).

In this thesis, we consider the setting where a global graph data is distributed as various subgraphs at the client side. The federated learning scenario is that each client locally trains a GCN and the server aggregated the local models into a global model, over multiple rounds. Each client then makes a further local improvement before classifying unlabeled nodes. We give an overview of related works in the following section. In the next chapter, we provide a detailed presentation of our problem setting.

2.3.1 Related Works

While there is a strong line of research in federated learning for euclidian data such as text, audio, and video, there is less study on federated learning for graph datasets. One of the main challenges of federated learning in structured graph data is that partitioning a graph among data owners can significantly alter the graph structure [8], [15]. As suggested in [3], in addition to data heterogeneity, there is another challenge in graph datasets called *complementarity* where the graph structure around overlapping nodes that exist in multiple local clients in local clients is not complete and each client's structure only consists of a part of overlapping nodes' structure. Most of the work in this area fail to address this issue and remove overlapping nodes from their data, and only consider disjoint graphs.

Federated graph neural networks taxonomies The authors in [8] provide a taxonomy and framework for graph federated learning. They categorize federated graph learning into four groups. 1) Graph-level FL, where clients

own different sets of small graphs for tasks like graph classification [31]; 2) Subgraph-level FL where each client owns a subgraph of a large graph as in our work; 3) Node-level FL where nodes are owned by clients and the privacy of nodes are important (IoT is mentioned as an example of this setting); 4) Edge-level FL, where privacy of the edges are important as in social networks.

A different taxonomy of graph federated learning is defined in [15], where data partitioning of a graph dataset based on the overlap degree is categorized into three settings. Clients without overlapping nodes [4], [30], [35], clients with partially overlapping nodes [22], [28], and clients with completely overlapping nodes [2]. While many studies have focused on the first setting [4], [30], very few have studied the second and the last setting.

Subgraph federated learning A meta-learning approach has been proposed [28] for graph federated learning. In their experiments, the authors consider the case of having overlapping nodes. However, their approach does not explicitly take advantage of these nodes for node classification. Overlapping nodes [22] have been considered in knowledge graphs to translate knowledge. The authors assume that aligned entities and relations for any two knowledge graphs are given. They leverage this data to improve knowledge graph embeddings by proposing a GAN-based [34] approach. Their setting is quite different from ours: it has no server, communications happen between clients of different domains, and node classification is the objective. In addition to sharing models with the server, another proposed method [3] uploads all node embeddings along with their node label prediction from each client to the server. Then the method creates a pseudo-graph structure and pseudo-node labels for all the nodes that are distributed back to the clients. This method has potential privacy risks and communication overhead. We believe since most of the subgraphs are not overlapped, sharing info about all parts of the subgraphs is likely not going to be useful (for example aggregated pseudo labels only happen on overlapping areas). In our method we only send embeddings of a portion of the nodes (the anchor nodes) to the server, reducing the communication overhead.

In [24], clients extend their graph with node features and edges from other graphs in a k-hop distance from the overlapping nodes. Private Set Intersection (PSI) is used to find the intersection in clients' data. They propose to use differential privacy to preserve the privacy of node and edge information. They show results from experiments on two user-interaction graph datasets for the link prediction task using graph neural networks.

Subgraph federated learning as described in [8] is also considered [35]. However, the subgraphs are disjoint. The authors train a generative model in local subgraphs to create missing neighbors of nodes in each client. Reconstructed neighbor features are sent to other clients, and a feature reconstruction loss is defined to generate features similar to another node in the global graph. Besides possible privacy leakage by sharing generated node features, it creates considerable communication overhead and potential computational overhead while training generative models in local graphs. A proposed approach in [23] improves the previous work [35] further by using a GAN for generating the neighbors.

The setting of having scarce labels in clients with potentially different downstream tasks is considered in [27], where the authors propose to first train the self-supervised learning (SSL) model federally. After training the global SSL model, each client further trains a task-specific local model by freezing or fine-tuning it on top of the globally trained model. We also have proposed to incorporate SSL in graphs in order to take advantage of the knowledge of graph structure.

Unlike previous works in this line of research, we don't assume lost connections as a result of local graphs' separation. Rather, we assume graphs' structures around anchor nodes are incomplete and distributed as a result of splitting from anchor nodes. We propose to expand the local structure according to the nodes in overlapping areas (anchor nodes). While most of the subgraph federated learning algorithms have compared their method with only FedAvg which is proven to not be very suitable for highly non-iid settings, we also compare our work with FedProx which has proven to be very suitable for the non-iid data.

Chapter 3

Methodology

3.1 Problem Setting

Our setting is a federated learning scenario with a set of K clients, each with a subgraph of data that we assume is part of a global graph $G = (V, E)$. Each client k has data structured in a subgraph $G^k = (V^k, E^k)$, $V^k \subseteq V$ and $E^k \subseteq E$. We further assume the presence of so-called *anchor* nodes, which are nodes that may be present in multiple subgraphs, or clients. This would naturally occur in realistic scenarios where an individual's data may be present at multiple organizations, such as all hospital clinics they have been to, banks they have dealt with, or retail stores they have transactions at. We denote the set of anchor nodes at client k with $\ddot{V}^k$. When referring to a specific anchor node at client k, we will use the notation $\ddot{v}_i^k$ where $1 \leqslant i \leqslant \ddot{V}^k$.

Figure 3.1 depicts this scenario, where the colored and numbered nodes are the anchor nodes. We assume that the central server has knowledge of anchor nodes and which clients they belong to. Previous works have used private set intersection [24] or sharing of client graph with the server [3] to gather this information. Our methodology does not depend on how the server has information on anchor nodes, but only assumes that the server has information on the identity and location of anchor nodes. Further, only this information is known at the server - the identity of the remaining nodes or the presence of edges is not known.

Each node v_i^k , $1 \leqslant i \leqslant V^k$ at client k has a feature vector X_i^k. A very small set of nodes also have labels Y_i^k. Our task is transductive node classification,

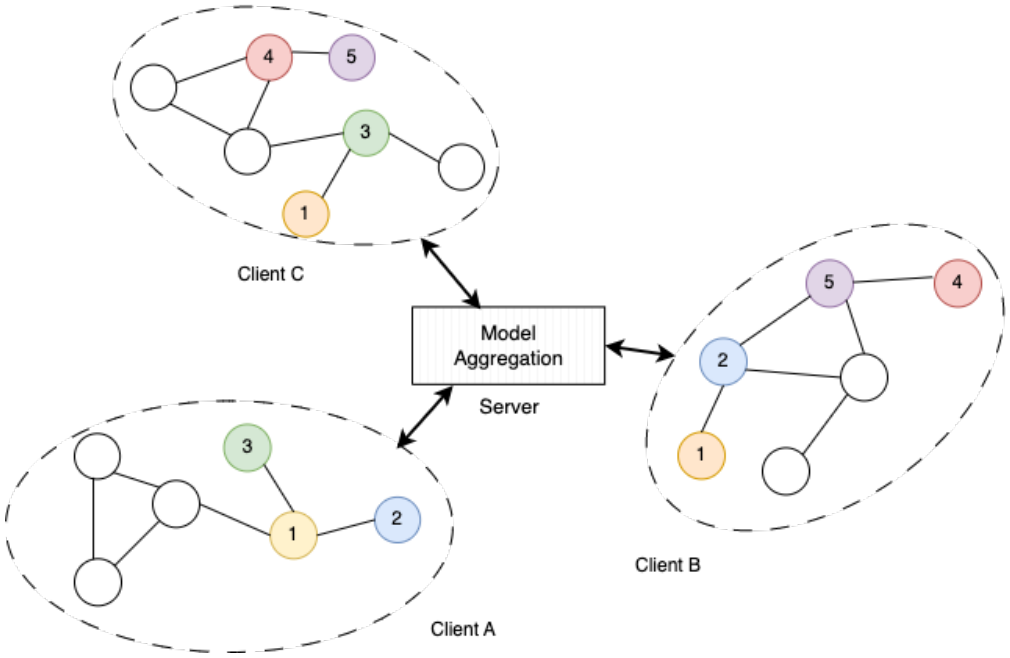


Figure 3.1: Federated learning scenario with subgraphs. The colored and numbered nodes are anchor nodes that are present in multiple clients.

that is, the classification of the remaining unlabeled nodes. Our algorithm to learn the classification model takes advantage of the local subgraph structure and aggregated embeddings of anchor nodes, resulting in improved prediction performance. In particular, our method starts with a phase of standard FL to learn node embeddings where locally trained parameters are shared with the server and aggregated parameters are received. Once clients converge, we add an augmentation phase. Here embeddings of only the anchor nodes are shared with the server which then sends back aggregated embeddings. These new embeddings now encode the global structure around the anchor nodes and thus represent structural information that is richer. With these new embeddings, we augment the local graphs with new predicted links. The final phase is standard FL training with the original loss function, with embeddings learned from the augmented local structures. Our method Fed-GALA, Federated Learning on Graphs with Anchors and Link Augmentation, is presented in more detail in the next section.

3.2 Fed-GALA

Our algorithm Fed-GALA is composed of three phases: a FL training phase (Section 3.2.2) where the loss functions for classification and link prediction are combined, a phase of boosting anchor embeddings (Section 3.2.4) and augmenting local graphs, and then a final phase (Section 3.2.5) of FL training for the classification task. The basic version of our algorithm, Fed-GALA, uses the FedAvg algorithm our own method of parameter aggregation at the server. The FedProx version of our algorithm, Fed-GALAp uses the proximal term shown in Section 2.1.2 in the local objective functions and our parameter aggregation method at the server.

3.2.1 Algorithm

Here we discuss a summary of all the pieces of the algorithm together. In the Algorithm 3, all the steps of Fed-GALA are shown.

This algorithm is executed at the server and the client side algorithm is described in the Algorithm 5. In line 1 of Algorithm 3, the server initializes the parameters of the global GCN model w_0 which in the case of a 2-layer GCN consists of a first layer set of parameters (W_0), and a second layer set of parameters (W_1), $w_0 = \{W_0, W_1\}$. In line 2 of Algorithm 3, the server calls `Phase1Training(K)` function which takes the set of all participating clients and the initialized model w_0 as its inputs and according to Algorithm 4 executes the phase 1 of training. The function `Phase1Training(K)` function in the server stores all anchor embeddings in Z_a, which is a set of anchors' embeddings received from the clients ($Z_a = \{Z_a^k | \forall k \in K\}$). The other output of this function is the round number corresponding to the last round of phase 1 training and the trained global model so far. The whole algorithm has a limit of max_R rounds, some of the rounds of training take place in `Phase1Training(K)` function. Then phase 3 of the training starting from line 6 resumes the training from round R to max_R round. For more details of `Phase1Training(K)` please refer to Section 3.2.2.

In line 3 of Algorithm 3, the server calls the function `Augment(Z_a,K)`

which aggregated anchor embeddings and sends these aggregated embeddings to the clients that have the respective anchor nodes. The result of this function is augmented clients' graphs. This phase is explained thoroughly in Section 3.2.4.

The rest of the algorithm is for phase 3 of the Fed-GALA where the server continues the main loop from the last round of phase 1 training (R) until reaching the max_R or until the convergence of all clients on the phase 3 task (node classification). Client convergence in this algorithm is determined in each client based on its loss function at this and the previous step. We now explain each phase in detail.

Algorithm 3 Fed-GALA algorithm. K is the set of all clients, K_i is the set of clients that have node i. E is the number of local epochs and η is the local learning rate. ρ_k is the local data owned by client k.

Server executes

initialize w_0 of a central GCN.
$w_R, Z_a, R \leftarrow$ Phase1Training(K, w_0) ▷ Phase 1, Algorithm 4
Aggregate (Z_a, K) ▷ Phase 2, Algorithm 6
for each round $t = R, R+1.., max\_R$ **do** ▷ Phase 3 training
    $S_t \leftarrow$ clients from k not yet converged on phase 3 target.
    for each client $k \in S_t$ **in parallel do**
        $w_{t+1}^k \leftarrow$ ClientUpdate $(k, w_t, phase = 3)$
    $w_{t+1} \leftarrow \sum_{k \in K} \frac{s_k}{s} \frac{n_k}{n} w_{t+1}^k$ ▷ Server Aggregation

3.2.2 Phase 1: Combined Training

The first phase in Fed-GALA is a standard FL training setting, where in each round clients train locally and then share their model weights with the server. The server then aggregates all local weights into global weights which are sent back to the clients. The clients then train locally with these updated weights. The local objective functions in this phase include both the loss function corresponding to the classification task, L_{C} (2.8) and the loss function for the link prediction task, L_{GAE} (2.9). In each local epoch, we combine these loss functions with equal weighting:

$$L_{total} = \frac{1}{2} L_{GAE} + \frac{1}{2} L_C. \tag{3.1}$$

We combine the two loss functions because while the main task is node classification, including the edge prediction loss will allow us to predict links for augmentation in phase 2. Interestingly, as we show through experimental results in Table 5.3 in Section 5, even without the step of edge prediction in phase 2, including the edge prediction loss in training improves the node classification accuracy. We note that the weighting used in combining the loss functions (3.1) need not be equal, and can instead be controlled by a hyperparameter. We leave the exploration of such hyperparameters for future work.

At the end of the local training in each round, the local GCN parameters, W_0^k and W_1^k for client k, computed as defined in (2.6), are communicated with the server. The server then aggregates the local parameters into global parameters. Our aggregation method differs slightly from the standard FedAvg to account for the small set of labeled nodes in our setting. We explain this method in detail in the next section.

This phase of FL training terminates when local training at all available clients has converged on their respective combined training loss as indicated in line 4 of Algorithm 4. Our criteria for local training convergence is that the difference between two consecutive losses be less than $\alpha \ll 1$. We explain further the tuning of α in Section 4.2.2.

Pseudocode

Algorithm 4 demonstrates the first phase of training. The local training on the client side is shown in Algorithm 5. The output of the first phase consists of the final aggregated parameters on the server, the anchor embeddings, and the final round number, where phase 1 training stopped. Moving forward, the server will continue to track the number of rounds in subsequent phases, beginning with the last round completed in phase one ($t+1$) to accurately keep track of all the rounds. This is needed to terminate the training if it reached the maximum number of rounds. The `ClientUpdate(k,w, phase)` function in the Algorithm 5 is executed in the client and chooses a loss function for optimization corresponding to the phase it is called in. In the first phase, the loss function in Algorithm 5 would be (3.1).

The function `PullClientAnchorEmbs(k)` in line 4, simply receives the anchor node embeddings from client k and saves them in the Z_a^k. Anchor embeddings are generated in clients by doing a forward pass on the local subgraphs according to equation (2.6). These embeddings are then stored in Z_a, which is the set of all anchor node embeddings used in the augmentation step.

The `for` statement in Algorithm 4 line 8 makes sure the model of converged clients is kept saved on the server side for the next rounds of server model aggregation. Even though converged clients do not participate in their local training, their latest model is used in server model aggregation in line 10.

Algorithm 4 Phase 1 training in Fed-GALA. The output of this phase is the latest global model parameter, the set of node embeddings of anchors from all the clients, and the final round number of phase 1.

Phase1Training (K, w_0): ▷ Phase 1 training coordinated on the server
1: $S \leftarrow$ all the K available clients
2: t = 0
3: **while** $|S| > 0$ **do**
4: **for** each client $k \in S$ **in parallel do**
5: $w_{t+1}^k \leftarrow$ ClientUpdate $(k, w_t, phase = 1)$ ▷ Algorithm 5
6: **if** client k has converged **then**
7: $Z_a^k \leftarrow$ PullClientAnchorEmbs(k)
8: $S \leftarrow S - k$
9: **for** client k in $K - S$ **do**
10: $w_{t+2}^k \leftarrow w_{t+1}^k$
11: $w_{t+1} \leftarrow \sum_{k \in K} \frac{s_k}{s} \frac{n_k}{n} w_{t+1}^k$
12: t++
13: **return** $(w_{t+1}, Z_a, t+1)$

3.2.3 Parameter Aggregation

In the standard scenario where all data points (nodes) are labeled, the averaging in FedAvg suffices. In our setting, only a small set of nodes are labeled, but the other unlabeled nodes also contribute to local parameter learning through their embeddings. Taking this into account, we propose an aggregation that appropriately weights clients according to both types of nodes

Algorithm 5 The function ClientUpdate describes local training on local data ρ_k with the server model parameters. Client k receives the server model parameters w, and performs E epochs of local training on its private data. The output of the function is the updated model parameters after E epochs.

ClientUpdate $(k, w,$ phase$)$: ▷ Run on client k

$w_0 \leftarrow w$
for each local epoch e from 1 to E **do**
    if phase $== 1$ **then**
        $L = \frac{1}{2}L_{GAE} + \frac{1}{2}L_C$.
    else
        $L = L_C$.
        $w_e \leftarrow w_{e-1} - \eta \Delta L(w_{e-1}; \rho_k)$
return w_E to server

$$\mathbf{W}_{t+1} \leftarrow \sum_{k \in K} \frac{s_k}{s} \frac{n_k}{n} \mathbf{W}_t^k, \tag{3.2}$$

Here, $\mathbf{W}_t$ and $\mathbf{W}_t^k$ are the global and local parameter vectors, respectively, s_k and s are the number of labeled nodes at client k and globally, respectively, and n_k is the number of nodes at client k. This weighting reflects the observation that clients with larger graphs and more labeled nodes tend to have more structural information and thus may lead to better node representations.

Our method of averaging is used for both phase 1 (Algorithm 4) and phase 3 (Algorithm 3) of the training.

Fed-GALAp, the FedProx version of our algorithm, follows the same phase 1 algorithm and server aggregation, with the exception that the local objective function (3.1) at client k includes the proximal term $\frac{\mu}{2}\|\mathbf{W}_t^k - \mathbf{W}_t\|$ in round t.

Our method of aggregation is particularly advantageous in scenarios where there is a significant disparity in the size of the graphs and the number of labeled nodes between clients. For instance, if a client has a small graph with only $\tilde{s}$ sampled nodes, and another client has a much larger graph with the same number of sampled nodes, the original FedAvg server aggregation formula assigns equal weight to both clients. However, this approach fails to account for the potential higher quality of the model generated by the larger graph. To address this issue, our proposed server aggregation formula assigns a larger weight to the model generated by the larger graph due to richer node

representation. Consequently, our approach significantly improves the overall performance in such scenarios. In cases where there is no significant mismatch between the number of labeled nodes and the size of the graph, our proposed averaging formula behaves similarly to the FedAvg averaging approach.

3.2.4 Phase 2: Augmentation

After the combined training phase we have an augmentation phase where anchor embeddings are shared with the server which aggregates the embeddings for each anchor node. These aggregated embeddings are used at the local clients for the augmentation of local structure through link prediction.

Aggregation of anchor embeddings

In this step, each client sends the node embedding of its anchors to the server. The server aggregates the embeddings of each anchor node by averaging. Consider an anchor node $\ddot{v}_i$ that exists in subgraphs at the set of clients $\mathcal{K}_i \subseteq \mathcal{K}$ where $\mathcal{K}$ is the set of all clients. Each client $k \in \mathcal{K}_i$ sends the locally computed embedding for anchor node $\ddot{v}_i$, Z_i^k calculated according to the equation (2.6), to the server, and the server aggregates as follows:

$$Z_i^{\mathrm{G}} = \sum_{k \in \mathcal{K}_i} \frac{Z_i^k}{|\mathcal{K}_i|}, \ \forall \ddot{v}_i \in \ddot{V} \tag{3.3}$$

where Z_i^{G} is the globally aggregated node embedding of node $\ddot{v}_i$, and $\mathcal{K}_i = \{k : k \in K, \ddot{v}_i \in V^k\}$ is the set of all the clients that contain anchor node $\ddot{v}_i$. $|\mathcal{K}_i|$ is the number of all the clients with anchor node $\ddot{v}_i$, $\ddot{V} = \{\ddot{v}_i, i = 0, ...|V|\}$ is the set of all anchor nodes, and $\ddot{v}^k = \{\ddot{v}_i : \ddot{v}_i \in V^k, \ddot{v}_i \in V_a\}$ is the set of anchor nodes in client k.

Pseudocode This part of the algorithm takes place in the server where the server receives the anchor node embeddings from clients after their convergence in phase 1 of the algorithm as in the line 4 of the phase 1 Algorithm 4.

Algorithm 6 This function in the server initiates the local augmentation in clients by aggregating anchor embeddings and sending these embeddings to their client owners. $\mathcal{K}_i$ is the set of clients that have node $\ddot{v}_i$. Z_i^k is the embedding of node $\ddot{v}_i$ in client k. $\ddot{V}$ is the set of all anchor nodes, and $\ddot{V}^k$ is the set of anchor nodes in client k.

Aggregate (Z_a, K):

1: **for** each node $\ddot{v}_i \in \ddot{V}$ **do**
2: $\quad Z_i^{\mathrm{G}} = \sum_{k \in \mathcal{K}_i} \frac{Z_i^k}{|\mathcal{K}_i|}, Z_i^k \in Z_a$
3: **for** each client $k \in K$ **in parallel do**
4: $\quad Z_a^k = \{(Z_a^G) : \ddot{v}_i \in \ddot{V}^k\}$
5: $\quad$ ClientAugment(k, Z_a^k) ▷ Augment clients according to (3.4)

Augmentation of local structure

Each client augments its local subgraph structure using the global embeddings for the anchor nodes and the original local embeddings Z^k for the other nodes. A single link is added to each anchor node according to a link prediction score. For each anchor node $\ddot{v}_i \in \ddot{V}^k$, with global embedding Z_i^{G}, $\ddot{V}^k$ the set of anchor nodes and V^k the set of all the nodes at client k, a link is added to the node $v_j(i) \in V^k$ with the highest link prediction score given by the following:

$$v_j(i) = \underset{v_l \in V^k}{\operatorname{argmax}} \operatorname{softmax}(\langle Z_i^G, Z_l^k \rangle) \tag{3.4}$$

Note that $<,>$ denotes the inner product. The goal of this step is to approximate each anchor node $\ddot{v}_i$'s global neighbors and augment the local graph with this information. This brings a richer structure locally and leads to a better node classification performance. We have used the softmax function of the inner product of the node embeddings which is the same edge prediction score used in [11]. We only add the link with the highest score to gain the most advantage without perturbing the structure too much. We can also envision adding more links. We leave this study for future work. An illustration of link addition is shown in Figure 3.2.

3.2.5 Phase 3: Training for the task

Once the local graph is augmented, this new subgraph is used in another phase of FL training, this time using only the classification loss function L_{C} 2.8.

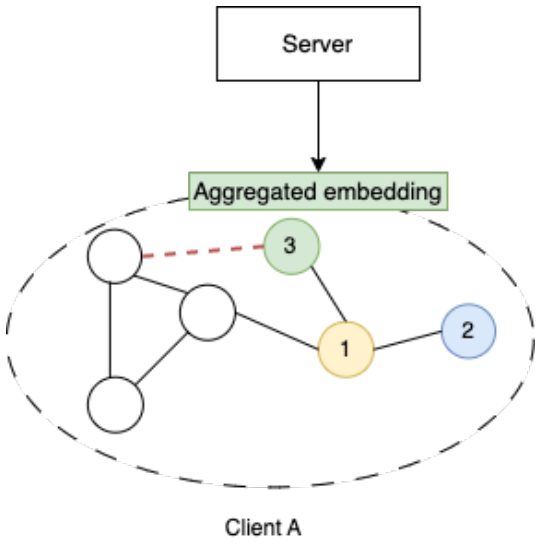


Figure 3.2: An illustration of augmenting a local subgraph using link prediction on an anchor node

When training ends, testing on node classification is carried out. The pseudocode for this phase is in the main loop of the main Algorithm 3.

Chapter 4

Experimental design

4.1 Datasets

We run experiments on four graph datasets with various sizes and statistics. Cora, Citeseer, and PubMed are academic citation networks from [33]. Here nodes are documents and links represent citations. WIKI-CS introduced in [21] is derived from Wikipedia. In this dataset, nodes correspond to computer science articles and edges represent hyperlinks. Nodes are classified into 10 classes which represent different branches in the field. All of these datasets are undirected graphs with unit edges.

There are publicly available train, test, and validation set splits for Cora, Citeseer, and PubMed [12], [33], which are widely used in previous works. In these splits, for each class, 20 nodes are labeled and used as the training set. 1000 nodes and 500 nodes are randomly selected for the test set and validation set respectively. We use these published splits for our experiments. For WIKI-CS [21] dataset we use a split introduced in the original paper, where, 50% of the nodes are randomly selected for testing. From the other 50% nodes, 5% of the nodes are randomly selected for training, and 22.5% are used as the validation set for hyper-parameter tuning. We select one of their publicly available splits for our experiments.

Graph properties such as density and connected components will clearly have an impact on the node classification performance. A theoretical study into the impact of each such property is out of the scope of this thesis. Rather, we list some graph statistics on these datasets in Table 4.1 and refer to these

Table 4.1: Statistics of datasets

Name	#nodes	#edges	Mean degree	#features	#classes	Label rate
Cora	2,708	5,429	4.0	1,433	7	3.6%
Citeseer	3,327	4,732	2.8	3,703	6	5.2%
PubMed	19,717	44,338	4.5	500	3	0.3%
Wiki-CS	11,701	216,123	36.9	300	10	5%

in analyzing performance.

4.1.1 Data Simulation

Since we have not been able to find publicly available graph data that is already split across clients, we simulate the local graphs by splitting the above graph datasets. We use the Louvain algorithm [1] to create communities as described in Section 2.2.1. All communities created by this method are first sorted based on size and allocated to the clients in the following way. To start, all clients are empty and numbered in an arbitrary way. First, the largest community is assigned to the first client. The second largest community is then assigned to the second client and so on. Not all clients will have exactly the same graph sizes. This method creates disjoint subgraphs.

To create the anchors, we duplicate some nodes in different clients based on the following rule: any node in a client that has a link in the global graph to a node in another subgraph, is duplicated in the other subgraph. This method of duplicating nodes makes sure there are no missing links between subgraphs owned by clients. Note that since the Louvain method creates communities based on modularity where nodes are likely to have more edges within the community than outwards, the anchors will be a small set of total nodes.

For instance, consider Figure 4.1, clients 'A' and 'B' own disjoint subgraphs first. However, in the original global graph, there is a link connecting node 1 to node 2, which are not in the same subgraph now. We create anchors by duplicating node 2 inside client A's subgraph and duplicating node 1 in client B's subgraph. By doing so, client A and client B would reserve the edge data between node 1 and node 2. Note that, even with the overlap assignment,

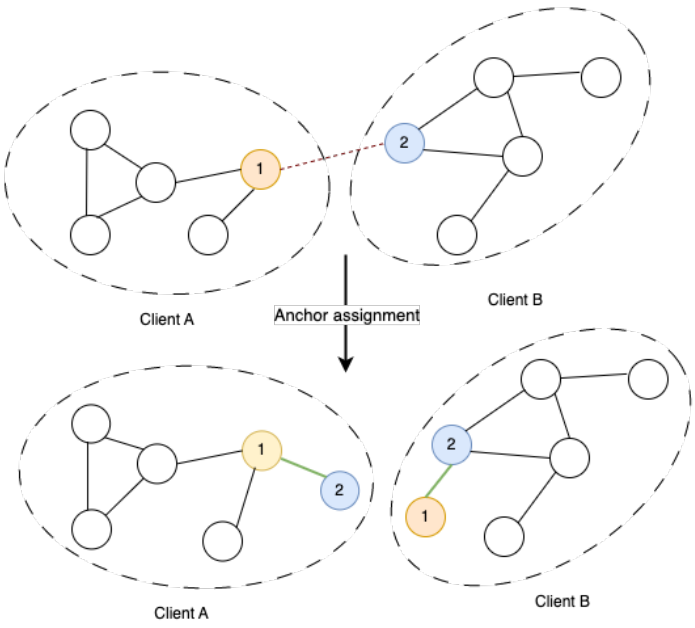


Figure 4.1: An illustration of assigning anchors to clients having disjoint subgraphs.

client A has node 2 but does not have access to its neighboring nodes at client B. Vise versa, client B has the anchor node 1 but does not have access to its other neighboring nodes and connections that exist in client A. So, while nodes are duplicated, their local graph structure is not. This may cause a discrepancy between the learned node representations of node 1 at client A and at client B; and similarly for node 2. This discrepancy is relevant when anchor node embeddings are aggregated in phase 2.

As the number of clients increases, the communities are further split between clients, therefore the quality of the local graph structure worsens. According to the definition of anchors, as the number of clients increases, the ratio of anchors to non-anchors increases naturally. The anchors then will play an important role in emulating neighborhood structure lost in the split.

Subgraph data statistics

Depending on the number of clients, the subgraphs would have different sizes and graph properties such as the number of nodes, edges, and anchor nodes.

Table 4.2: $|C^k|$ is the average number of communities in clients. $|n_a^k|$ is the average number of anchor nodes in each client. $|n^k|$ is the average number of nodes in all the local graphs and $|e^k|$ is the average number of edges in local graphs.

Dataset	# Clients	$\lvert C^k\rvert$	$\lvert n_a^k\rvert$	$\lvert n^k\rvert$	$\lvert e^k\rvert$	$\frac{\lvert n_a^k\rvert}{\lvert n^k\rvert}$
Cora	4	26	330.5	859.25	1582	0.384
	8	13	199.5	454.75	804.125	0.438
Citeseer	4	118	167.25	1228.3	1241.25	0.136
	8	59	90.875	465.375	620.5	0.195
PubMed	4	10.25	2753.5	6500.25	15294.75	0.423
	8	5.125	3255.5	6899.75	7835.25	0.471
Wiki-CS	4	251	4074.25	6230.75	93689	0.650
	8	125.5	2439.25	3975.875	59372.875	0.613

We present more statistics about the structure of local graphs in 4-client and 8-client scenarios in Table 4.2, as a demonstration of how the subgraph properties vary.

4.1.2 Data Visualization

In this section, we provide the visual graph structure of Cora and Citeseer, before and after splitting the graphs into 4 partitions. The color of the nodes is an indication of their true label. Red nodes are the anchor nodes. As you can see in the global graphs of Citeseer and Cora in Figures 4.2 and 4.4 respectively, and their local graphs in Figures 4.3 and 4.5 respectively, anchor nodes exist mostly in the central parts of the graphs. As expected, in our use of anchor nodes to augment local graphs, anchor nodes are usually influential nodes with many connections in the global graph. Their connections would exist at multiple clients, therefore using them as special nodes in our algorithm create the potential of learning beyond only local graphs.

4.2 Training Settings

We compare Fed-GALA with three other models: the baseline FL models FedAvg and FedProx, and the global model. Baseline FL models are trained,

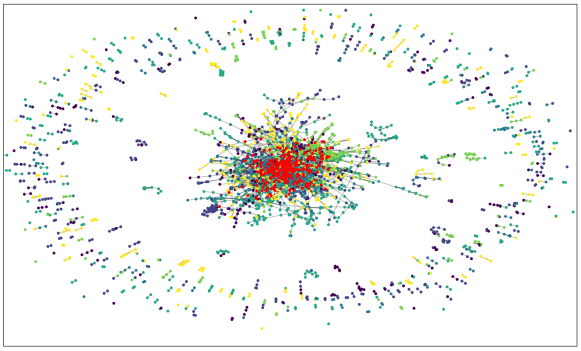
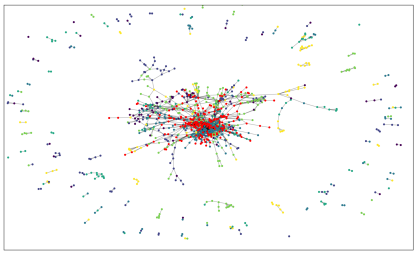
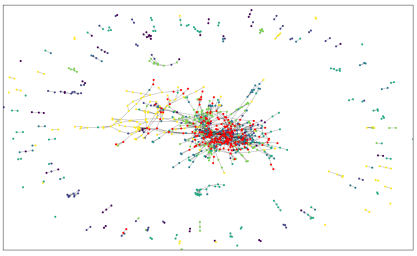


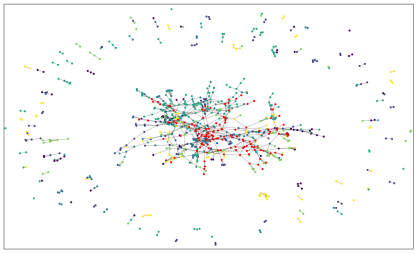
Figure 4.2: Citeseer before data split



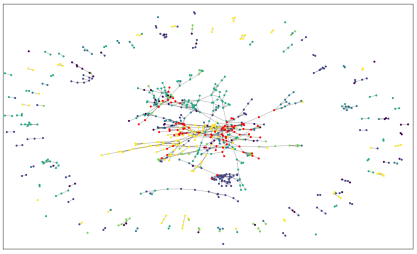
(a) Client 1 graph



(b) Client 2 graph



(c) Client 3 graph



(d) Client 4 graph

Figure 4.3: Citeseer data in 4-clients after the split

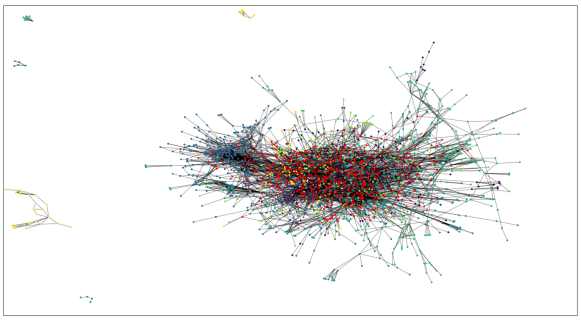
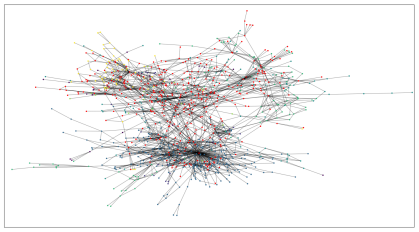
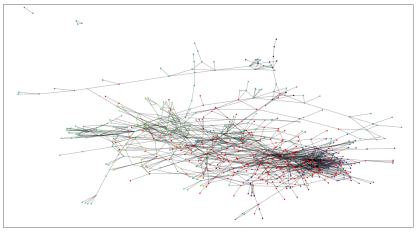


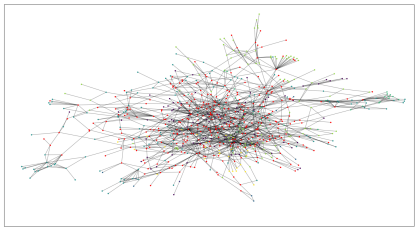
Figure 4.4: Cora before data split



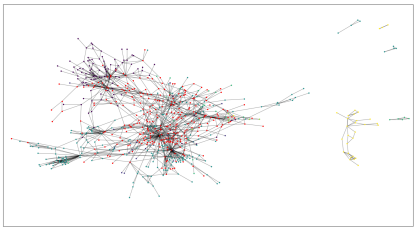
(a) Client 1 graph



(b) Client 2 graph



(c) Client 3 graph



(d) Client 4 graph

Figure 4.5: Cora data in 4-clients after the split

evaluated, and tested with the exact same data, network structure, and training settings as Fed-GALA and Fed-GALAp. The global model or the centralized model is a GCN model trained on the global graph, before being split into local subgraphs. The centralized model training is a hypothetical scenario and is usually not possible to achieve in real-world scenarios, but serves as an indication of what is usually expected to be the best performance.

4.2.1 Testing and Validation

The test accuracy results are reported based on two settings of local and global testing. In local testing, test nodes are located on local client graphs. Therefore each client tests its model only on test nodes located in its local graph. In global testing, each client's model is tested on the testing nodes located only on the global graph. Therefore, all clients are tested on the same graph, while none of them has seen the global graph during training. The global graph testing scenario may represent the case where each client may encounter nodes from other clients for classification. For example, one hospital's learned model may be used to classify a patient that is not at that hospital. The local testing scenario represents the case where clients are completely separate in that they don't encounter nodes from other clients, but learning through a server helps provide richer structural information.

Note that, the validation node set on the global graph is distributed to local subgraphs during the split phase in data simulation to generate local subgraphs. Clients use their validation nodes for local or global hyperparameter tuning. Therefore during training, no client has access to the global structure, in either testing scenario.

4.2.2 Hyper-parameter Tuning

We set most of the hyperparameters in our algorithms to match the settings described in the original papers. For the GCN algorithm, we used a 2-layer model with a hidden layer size of 128 units and applied a Relu activation function, a dropout rate of 0.3, and $5 * 10^{-4}$ L2 regularization. We used the same model for the GAE encoder. In our main experiments, we fixed

the maximum number of training rounds to 300 and each client trained their model for 3 local epochs at each round. Even though, in many works, clients are forced to participate for the whole number of rounds, in our work, we allow clients to finish their training whenever they converge locally. This allows for heterogeneity in client system capacity and size. The convergence of a client's model means that the client's local model parameter remains unchanged from that point on in the training of the given phase.

The training process of a client is considered converged if the difference in the client's loss between two consecutive rounds $|f_t - f_{t-1}|$ is smaller than α, or if the maximum number of rounds is reached, whichever came first. This parameter α is a threshold that stops the client's training based on the loss function. Once all the clients meet the convergence criteria, the whole algorithm is considered converged. The final model parameter in clients are then used for validation or testing. We set α to 0.0001 following the Fedprox [13] setting.

Phase 1 convergence Phase 1 of training is finished when all the clients have converged, and the, clients can converge at different rounds based on their training loss function. The loss function in this phase is the combined loss as explained in Section 3.2.2. If the difference in loss at a client ($|f_t - f_{t-1}|$) is less than a threshold (α), then that client is converged on phase 1 of the training. Other clients will continue the training until all the clients have converged. Then clients proceed to the next phases of training.

We optimized the α values used in the first training phase of our method based on local validation sets. The α value should be optimized according to the dataset and the training setting such as the number of anchor nodes. To optimize this threshold we did a grid search among a potential set of values $\{0.1, 0.01, 0.001, 0.0001\}$. Intuitively, the smaller the α value is, the more combined training we have in phase 1 to be able to do edge prediction in the next phase. The optimal values for the hyper-parameter α in various scenarios are shown in Table 4.3.

For a given dataset and setting, we find the same optimal values of α, for all

Table 4.3: Tuning of α, the convergence threshhold

Dataset	Testing method	#Clients	Algorithm	α value
Cora	Global	4/8-client	Fed-GALA/ Fed-GALAp	0.001
	Local	4/8-client	Fed-GALA/ Fed-GALAp	0.01
Citeseer	Global	4-client	Fed-GALA/ Fed-GALAp	0.001
		8-client	Fed-GALA/ Fed-GALAp	0.01
	Local	4/8-client	Fed-GALA/ Fed-GALAp	0.01
PubMed	Global/Local	4/8-client	Fed-GALA/ Fed-GALAp	0.0001
Wiki-CS	Global/Local	4/8-client	Fed-GALA/ Fed-GALAp	0.0001

FL algorithms (FedAvg and FedProx) in our experiments. From this finding, we can infer that the optimal value of α is dependent on the dataset and its distribution of anchor nodes, rather than the underlying federated learning algorithm used. According to Table 4.3, another factor that can change the optimal value of α is the testing method: local or global. Table 4.3 shows that, for larger datasets like PubMed and Wiki-CS, the optimal value is the same across all the settings that we experimented with. However, in smaller datasets like Cora and Citeseer, the optimal value of α might be different across different testing methods or even the number of clients. Yet, the underlying algorithm seems not to have any effect on the optimal values of α in all the datasets. In smaller datasets, the optimal values of α are usually smaller for the global testing method. This might be so because for testing a model on the global graph, we need more globally accurate augmented edges. In this case, models that can predict the structure of the graph are essential. So, a smaller α would yield better performance since it provides more convergence on the combined training loss.

Proximal term in FedProx and Fed-GALAp In the ClientUpdate function in Algorithm 2, we have a hyper-parameter μ that controls the regularizer term in the loss function. When μ is set to larger values, we force the local parameters to be closer to the global model parameters. The hyper-parameter μ in FedProx is optimized adaptively during training based on the suggested heuristic in the original paper [13]. They first initialize the μ to 1 which is an

adversarial initialization to their method. During the training, μ is decreased by 0.1 if the loss continues to decrease for 5 consecutive rounds. However, if the loss increases the value of μ would be increased by 0.1.

It should that all the hyper-parameters for training Fed-GALA or Fed-GALAp are again optimized and used for the respective baselines FedAvg and FedProx. This allows all the algorithms to be trained and tested on their optimal hyper-parameters and thus show their best performance.

Chapter 5

Experiments and results

We perform an extensive set of experiments and present their analysis in this chapter. We perform these experiments on the four datasets described in the last chapter, Cora, Citeseer, PubMed, and Wiki-CS. We begin with a set of experiments that demonstrate the overall performance of Fed-GALA and Fed-GALAp. Then we evaluate the impact of the number of clients, the link augmentation step, our aggregation method, and the training settings.

5.1 Experiment 1: Performance of Fed-GALA and Fed-GALAp

Experiment description Our main experiments consist of comparing Fed-GALA and Fed-GALAp with baseline models FedAvg and FedProx . We compare these models with the centralized model of a single global graph in a variety of settings. These models are compared in a 4-client and an 8-client scenario while being tested using the global and local testing approaches. Each result presented in Table 5.1 and Table 5.2 is from testing on 10 runs with randomly initialized model parameters. The mean of the results and its standard deviation in parenthesis is reported in these tables. For each run, the final accuracy is calculated by the weighted average of clients' accuracy on the test set using the same weights used for server model aggregation. This is to value clients' results that have more testing nodes as a result of owning larger graphs.

Main results Table 5.1 provides the performance results of Fed-GALA, and Table 5.2 shows the performance results of Fed-GALAp. The results of baseline FedAvg and FedProx are also provided to be compared with Fed-GALA and Fed-GALAp respectively. Across all datasets and settings, Fed-GALA and Fed-GALAp outperform the baseline models. The improvement becomes more significant as we increase the number of clients. FedProx is designed to tackle the problem of non-iid data in FL. Although our subgraphs may be non-iid and FedProx improves upon FedAvg considerably, the results show that FedProx alone is not sufficient for distributed subgraphs, and Fed-GALAp improves the accuracy considerably over FedProx. In most cases, even Fed-GALA achieves higher accuracy compared to the baseline FedProx. This demonstrates that our idea of using link prediction in local subgraphs to emulate the structure at the global level leverages the new structure for better node classification more so than using a proximal term in the local loss function to correct for non-iid data.

5.2 Experiment 2: Impact of the number of clients

We now analyze the impact of the number of clients in the FL scenario on the node classification performance.

Experiment description We consider the average accuracy of Fed-GALA in the two settings of FedAvg and FedProx across various numbers of clients. We experimented with 2 to 8 clients, the number of clients usually seen in cross-silo FL (FL between companies and large dataset owners) scenarios. As explained previously, Fed-GALA and Fed-GALAp refer to Fed-GALA deployed on FedAvg and FedProx algorithms respectively.

Rresults As the number of clients increases, the original dataset is split further, therefore the quality of local graphs decreases leading to decreased accuracy. However, we have seen that our method can take advantage of the

Table 5.1: Semi-supervised node classification accuracy results averaged over ten runs on four datasets using FedAvg and Fed-GALA on global and local testing modes.

Cora				
Centralized	# Clients	Testing method	FedAvg	Fed-GALA
0.803 (±0.005)	4 clients	Global	0.672 (±0.007)	**0.725** (±0.004)
		Local	0.717 (±0.007)	**0.729** (±0.005)
	8 clients	Global	0.469 (±0.011)	**0.623** (±0.012)
		Local	0.674 (±0.005)	**0.704** (±0.007)
CiteSeer				
0.703 (±0.006)	4 clients	Global	0.585 (±0.009)	**0.607** (±0.010)
		Local	0.611 (±0.009)	**0.631** (±0.007)
	8 clients	Global	0.465 (±0.013)	**0.566** (±0.011)
		Local	0.562 (±0.012)	**0.600** (±0.010)
PubMed				
0.79 (±0.004)	4 clients	Global	0.648 (±0.003)	**0.701** (±0.007)
		Local	0.731 (±0.004)	**0.741** (±0.008)
	8 clients	Global	0.620 (±0.005)	**0.667** (±0.007)
		Local	0.720 (±0.004)	**0.733** (±0.008)
Wiki-CS				
0.79 (±0.007)	4 clients	Global	0.665 (±0.010)	**0.692** (±0.006)
		Local	0.663 (±0.008)	**0.690** (±0.006)
	8 clients	Global	0.578 (±0.017)	**0.631** (±0.008)
		Local	0.575 (±0.016)	**0.603** (±0.008)

augmented structural information to limit the drop in performance. Figure 5.1 shows the improvements of Fed-GALA and Fed-GALAp compared to baseline methods over various numbers of clients. Results in Figure 5.1 demonstrate the effectiveness of Fed-GALA which is even more considerable as we increase the number of clients.

Figures 5.1a and 5.1b show the average accuracy results tested with global and local testing respectively. Starting with a smaller number of clients, when clients are tested on the global graph, the global graph provides a more complete structure during test time which means the FL models have results very close to the centralized model (79.2%) because the client graphs are likely

Table 5.2: Semi-supervised node classification accuracy results averaged over ten runs on four datasets using FedProx and Fed-GALAp on global and local testing scenarios.

Cora				
Centralized	# Clients	Testing method	FedProx	Fed-GALAp
0.803 (±0.005)	4 clients	Global	0.710 (±0.008)	**0.734** (±0.006)
		Local	0.724 (±0.007)	**0.743** (±0.006)
	8 clients	Global	0.502 (±0.011)	**0.631** (±0.013)
		Local	0.680 (±0.006)	**0.716** (±0.006)
CiteSeer				
0.703 (±0.006)	4 clients	Global	0.615 (±0.011)	**0.631** (±0.009)
		Local	0.628 (±0.011)	**0.646** (±0.007)
	8 clients	Global	0.535 (±0.011)	**0.592** (±0.012)
		Local	0.591 (±0.008)	**0.621** (±0.009)
PubMed				
0.79 (±0.004)	4 clients	Global	0.654 (±0.005)	**0.705** (±0.006)
		Local	0.738 (±0.003)	**0.758** (±0.007)
	8 clients	Global	0.634 (±0.006)	**0.677** (±0.008)
		Local	0.728 (±0.006)	**0.739** (±0.007)
Wiki-CS				
0.79 (±0.007)	4 clients	Global	0.681 (±0.008)	**0.698** (±0.006)
		Local	0.673 (±0.007)	**0.681** (±0.005)
	8 clients	Global	0.586 (±0.010)	**0.625** (±0.006)
		Local	0.569 (±0.013)	**0.631** (±0.100)

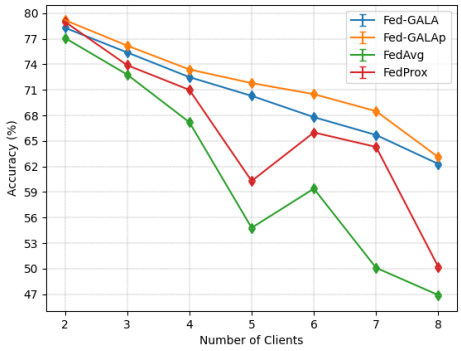
quite close to the global graph. However, as the number of clients increases, it is more challenging to test on the global graph that no client has seen during training - the client graphs are getting smaller and very different from the global graph. Therefore, the accuracy drops significantly from the 2-client setting to the 8-client setting by about 32%. Fed-GALA incorporates the global structure during training and thus the performance drop compared to the global graph is only 18% in the worst case, whereas the drop is 30% for FedAvg.

In the local testing setting as shown in Figure 5.1b, since the structural information inside clients' graphs is limited, the accuracy of the 2-client setting starts with a lower accuracy of about 0.78.4% compared to when tested on the global graph. However, as we increase the number of clients the accuracy drop is less compared to Figure 5.1a. The reason is that clients are trained and tested on their local graphs in contrast to global testing where they are tested on a new graph. Therefore the accuracy drop from a 2-client setting to an 8-client setting in Figure 5.1b is limited to 12%. Fed-GALA improves the accuracy in all of the settings and the accuracy drop from 12% to 8% in the worst case.

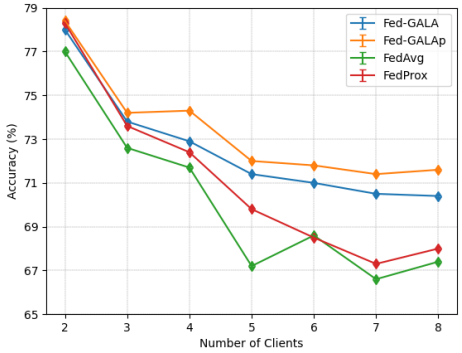
Figure 5.1c compares the improvement in accuracy of Fed-GALA compared to the baseline FL models in the same settings. We can see that as we have more clients, Fed-GALA's improvement is more considerable resulting in up to 15.5% improvement in the 8-client setting, a setting observed frequently in many applications.

5.3 Experiment 3: Impact of link augmentation

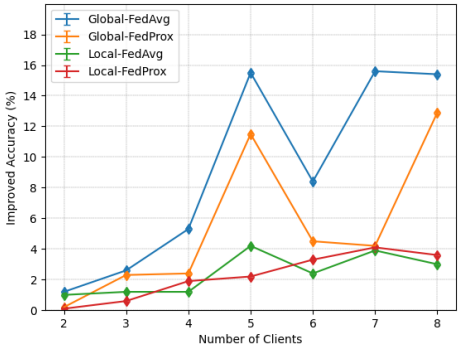
One of the key contributions of our work is the link augmentation step in our algorithm, which aims to improve node classification by enriching the local structure. However, the self-supervised training that precedes the augmentation phase also plays an important role in incorporating the local structural knowledge in the collaboratively trained global model by training it simul-



(a) Global testing accuracy



(b) Local testing accuracy



(c) Fed-GALA improvements

Figure 5.1: Impact of the number of clients on the semi-supervised node classification accuracy results on the Cora dataset.

taneously for node classification and edge prediction tasks. We investigated whether including the link prediction loss in the first training phase only, without performing the augmentation step, could still yield performance improvements. This indeed is the scenario where there is no access to anchor information and therefore no sharing of information with the server about which nodes are present at a client.

Experiment description To conduct this experiment, we define two alternative methods to ours. The first one is No-Augment which has phase 1 and phase 3 of Fed-GALA but does not have phase 2. In this method, no new links are added in phase 2, but we still have the combined training in phase 1. We compare with another method, Max-Augment, where there is no anchor embedding aggregation, but we augment local subgraphs by doing link prediction using only local embeddings. Max-Augment performs phase 1, phase 2 with these adjustments, and phase 3. This method is suitable for scenarios where anchor nodes are identified inside local graphs, but the server has no information about these nodes. Therefore, clients locally perform the augmentation of anchor nodes without any embedding communications with the server.

Fed-GALA is compared with these two alternative methods in both FedAvg and FedProx settings and accuracy results are demonstrated in Table 5.3.

Results We observe in Table 5.3 that No-Augment considerably improves the model performance compared to baseline models due only to the added edge training steps. Further, Max-Augment outperforms this result by augmenting the local structure of anchors in subgraphs. Adding edges based on the local node embedding in Max-Augment emphasizes the local structure of anchors even more, however, adding edges based on server supervision aggregated embeddings in Fed-GALA emphasizes the global structure based on the anchors' neighbors in other clients. As shown in Table 5.3, global structural information augmented in Fed-GALA can considerably improve the model performance compared to Max-Augment.

Table 5.3: Comparing Fed-GALA and Fed-GALAp with No-Augment, Max-Augment and baseline algorithms on Cora in the 8-client setting. The mean and the standard deviation of the test accuracy results over ten runs are reported.

Testing	FedAvg	Fed-GALA	Max-Augment	No-Augment
Global	0.469 (±0.011)	**0.623** (±0.012)	0.601(±0.007)	0.597 (±0.010)
Local	0.674 (±0.005)	**0.704** (±0.007)	0.697 (±0.005)	0.685 (±0.006)
	FedProx	Fed-GALAp	Max-Augment	No-Augment
Global	0.502 (±0.008)	**0.631**(±0.006)	0.612 (±0.005)	0.6115 (±0.006)
Local	0.680 (±0.007)	**0.716** (±0.006)	0.704 (±0.006)	0.700 (±0.007)

5.4 Experiment 4: Impact of our server aggregation

As explained previously in Section 3.2.3, we propose a new weighted server aggregation method that is designed for semi-supervised graph learning. In this formula (3.2), we give weights to the client's models to respect not only the number of available labeled nodes but also their total number of nodes. In this set of experiments, we investigate the impact of our proposed averaging formula on the performance of the trained models.

Experiment description We alter the server averaging formula in server from our averaging to the weighted averaging formula proposed in FedAvg. We can see the impact of our server averaging formula on the averaged accuracy over 10 runs on Citeseer in the 4-client and 8-client settings. We tested the models on both the global and local testing settings using both the FedAvg and FedProx baselines as well as Fed-GALA and algorithms.

Results As illustrated in Figures 5.2 and 5.3 our server aggregation has a positive impact on the performance of our framework compared to the standard model averaging formula proposed in FedAvg paper. This positive impact is consistent over all underlying algorithms, the number of clients, and the testing setting. Additionally, our server averaging formula in semi-supervised graph FL can be used even on the baseline FedAvg and FedProx algorithm to improve

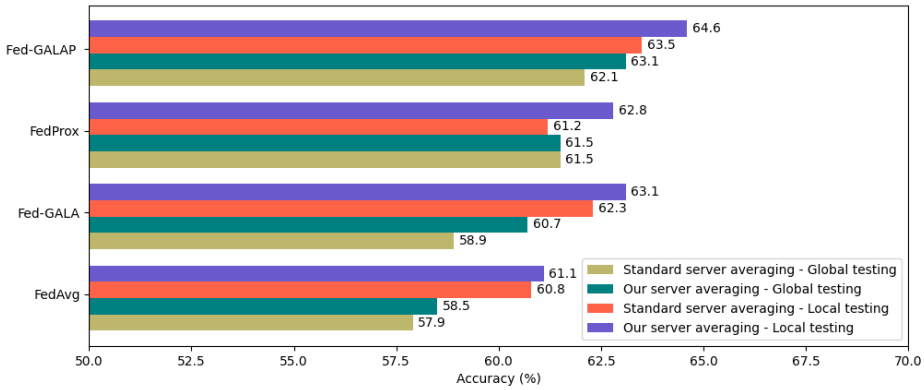


Figure 5.2: Aggregated models accuracy over 10 runs in the 4-client settings on the Citeseer dataset. This Figure shows how much our server averaging formula improves the client's models compared to the FedAvg server averaging formula.

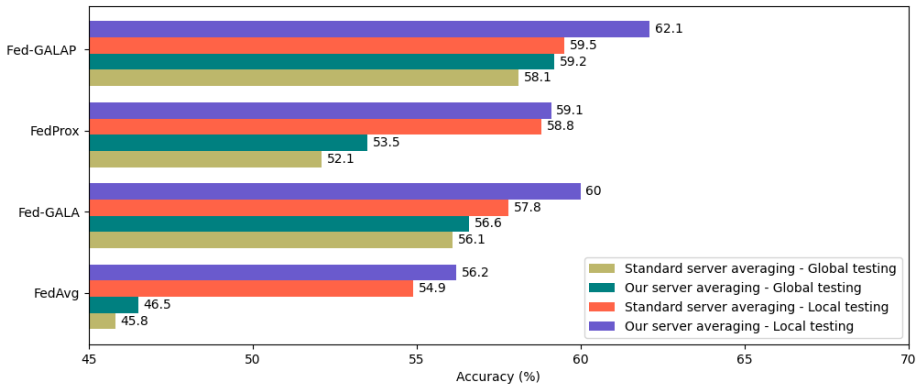


Figure 5.3: Aggregated models accuracy over 10 runs in the 8-client settings on the Citeseer dataset.

the performance of trained models.

5.5 Experiment 5: Impact of number of local epochs on performance and training time

In this experiment, we try to answer two questions. 1) What is the impact of the number of local epochs on the results? We compared the answer to this question in Fed-GALA and the baseline methods. 2) Over the different number of local epochs, how the training time of Fed-GALA compares to that of the baseline models? We measure the training time as a function of the total number of training rounds as the number of local epochs is the same over all the algorithms at any point. We conduct experiments with Fed-GALA and FedAvg in order to address these questions.

Experiment description In these experiments, we vary the number of local training epochs from 2 to 6, with steps of size 1. Note that, in all the other experiments, the number of local epochs is set to 3. We evaluate the performance of Fed-GALA and FedAvg in every local epoch setting, and the results are reported in the form of accuracy. In the same experiments, we also measure the training time of the algorithms by counting the total number of rounds it takes the whole algorithm to converge. We want to test training time overhead due to Fed-GALA. All the experiments are reported as the mean over 10 runs with random network initializations.

Results As shown in Figure 5.4, as we increase the number of local epochs, the averaged accuracy of FedAvg and Fed-GALA decreases. This is because having more local epochs leads to having locally better but more diverse local models such that their aggregation might not create the best possible global model. Models that are trained locally for longer, perform better on their local dataset, however, lack the ability to generalize well to other datasets. In contrast, with fewer local epochs, model updates are more synchronized by having more regular feedback from the server which leads to better globally

aggregated models. If clients only train their models with a few local epochs, they would need more server communication rounds which leads to more communication overhead. This phenomenon is mostly observed when clients have heterogeneous datasets where the optima of each client's local objective might be different from the global objective. The trade-off between minimizing the number of communication rounds and improving the aggregated model parameter by limiting the divergence of local models from the global model. This is a well-known trade-off in federated learning which is demonstrated in FedAvg [19] experiments and discussed in subsequent works such as FedProx [13] and more recent FL frameworks [14].

Our experiments also show that with more local epochs we need fewer communication rounds in both of the methods. The total number of training epochs depends on both the local epochs and the rounds. In fact, we can measure the total number of training epochs that have taken each method to converge by multiplying the number of local epochs and total rounds.

For the number of training rounds, the results are the mean over 10 runs. The mean of Fed-GALA is lower than that of FedAvg in these experiments, which means that the training time of Fed-GALA is less or equal to the baseline FedAvg in every setting. In fact, Fed-GALA is not adding to the training time by the measure of algorithm rounds. This could be considered an advantage as Fed-GALA improves performance, yet does not increase the total training rounds and in some cases, it even decreases that. In fact, the standard deviations of the illustrated mean values are mostly larger than the gap between the reported values of the two methods. Therefore, we don't assume that Fed-GALA always makes the training faster, but we can conclude that it does not make it slower. The sudden increase in the number of rounds in local epochs of 4 compared to the local epochs of 3 can be related to the many sources of randomness in the algorithm and the large standard deviation that we usually observe. Therefore, our conclusion that as we increase the number of local training epochs, the communication rounds decrease remains valid.

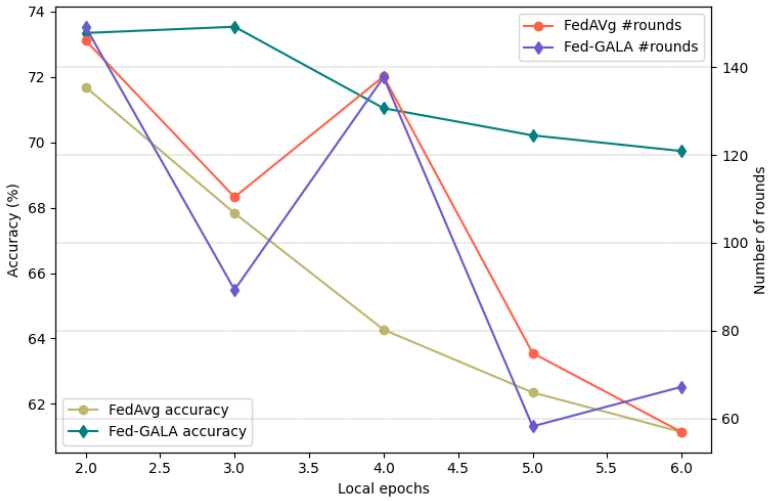


Figure 5.4: Impact of the local epochs in clients on the final aggregated accuracy results and the total number of communication rounds which is an indication of the training time. This experiment is done on the Cora dataset in the 4-client setting. All the results are averaged over 10 runs.

Chapter 6

Conclusion

6.1 Conclusion

We introduced a novel method for node classification on distributed non-disjoint subgraphs, Fed-GALA. Fed-GALA considers the anchor nodes (overlapping nodes) in subgraphs and augments their local structures collaboratively based on the global structure of data points. Fed-GALA has achieved considerable performance improvement without additional communication overhead. We have conducted extensive experiments on all the important factors in our algorithm and have compared it with two state-of-the-art baselines. However, there are some limitations that we discuss next.

6.2 Future Directions and Discussion

Future works on this topic can investigate on the privacy risk associated with sharing some data point embeddings with the server. Though, no raw data point is being shared with the server, having both the local model parameters and the node embeddings raises important privacy questions. Analyzing the actual privacy risks with this method is not very straightforward. This could be due to the fact that node embeddings in GNNs are not only dependent on the node itself but also on its neighboring nodes. So, it may not be correct to assume the server can infer the whole data point features by only having the node embeddings and the model.

Another promising future work lies in designing an algorithm for finding the

anchor nodes in the server without exposing private information to the server. We have mentioned a potentially suitable algorithm for this goal called private set intersection. However, more communication and computation-efficient algorithms are required for large scale use-cases.

As investigated by our experiments, Fed-GALA does not pose additional communication overheads in terms of the number of rounds. Clients are required to share their anchor embeddings with the server during only one round, which is not a significant additional overhead.

We note that there is potential computation overhead posed on to the client side during the augmentation step. This step requires a random search through the nodes to find the best link for each anchor. Other heuristics in searching the graph can be applied and investigated in this step to address this shortcoming.

We have suggested two alternative methods in Section 5.3 to remove the need for accepting any potential privacy risks due to sharing node embeddings. One of the introduced alternative algorithms removes any potential communication and computation overhead as well. These methods are designed to address the above mentioned concerns, but the performance does suffer a little. Future work may offer such methods that maintain the performance of Fed-GALA.

References

[1] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2008, no. 10, Oct. 2008.

[2] C. Chen, J. Zhou, L. Zheng, *et al.*, "Vertically federated graph neural network for privacy-preserving node classification," *arXiv preprint arXiv:2005.11903*, 2020.

[3] C. Chen, W. Hu, Z. Xu, and Z. Zheng, "Fedgl: Federated graph learning framework with global self-supervision," *arXiv preprint arXiv:2105.03170*, 2021.

[4] F. Chen, P. Li, T. Miyazaki, and C. Wu, "Fedgraph: Federated graph learning with intelligent sampling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 8, pp. 1775–1786, 2021.

[5] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *International conference on machine learning*, PMLR, 2017, pp. 1263–1272.

[6] W. Hamilton, "Graph representation learning," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 14, pp. 1–159, Sep. 2020. DOI: 10.2200/S01045ED1V01Y202009AIM046.

[7] W. L. Hamilton, R. Ying, and J. Leskovec, *Inductive representation learning on large graphs*, 2017. DOI: 10.48550/ARXIV.1706.02216. [Online]. Available: https://arxiv.org/abs/1706.02216.

[8] C. He, K. Balasubramanian, E. Ceyani, *et al.*, "Fedgraphnn: A federated learning system and benchmark for graph neural networks," *CoRR*, vol. abs/2104.07145, 2021. [Online]. Available: https://arxiv.org/abs/2104.07145.

[9] K. Hsieh, A. Phanishayee, O. Mutlu, and P. B. Gibbons, "The non-iid data quagmire of decentralized machine learning," *ICML'20: Proceedings of the 37th International Conference on Machine Learning*, 2020.

[10] P. Kairouz, H. B. McMahan, B. Avent, *et al.*, "Advances and open problems in federated learning," *Foundations and Trends® in Machine Learning*, vol. 14, no. 1–2, pp. 1–210, 2021, ISSN: 1935-8237. DOI: 10.1561/2200000083. [Online]. Available: http://dx.doi.org/10.1561/2200000083.

[11] T. N. Kipf and M. Welling, "Variational graph auto-encoders," *NIPS*, 2016.

[12] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *ICLR*, 2017.

[13] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated optimization in heterogeneous networks," in *Proceedings of Machine Learning and Systems 2020, MLSys 2020, Austin, TX, USA, March 2-4, 2020*, I. S. Dhillon, D. S. Papailiopoulos, and V. Sze, Eds., mlsys.org, 2020. [Online]. Available: https://proceedings.mlsys.org/book/316.pdf.

[14] T. Lin, L. Kong, S. U. Stich, and M. Jaggi, "Ensemble distillation for robust model fusion in federated learning," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33, Curran Associates, Inc., 2020, pp. 2351–2363. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2020/file/18df51b97ccd68128e994804f3eccc87-Paper.pdf.

[15] R. Liu and H. Yu, "Federated graph neural networks: Overview, techniques and challenges," 2022.

[16] Y. Liu, X. Ao, Z. Qin, *et al.*, "Pick and choose: A gnn-based imbalanced learning approach for fraud detection," in *Proceedings of the Web Conference 2021*, 2021, pp. 3168–3177.

[17] Y. Liu, Y. Kang, X. Zhang, *et al.*, "Fedbcd: A communication-efficient collaborative learning framework for distributed features," *IEEE Transactions on Signal Processing*, pp. 1–12, 2022. DOI: 10.1109/TSP.2022.3198176.

[18] Y. Liu, M. Jin, S. Pan, *et al.*, "Graph self-supervised learning: A survey," *IEEE Transactions on Knowledge and Data Engineering*, 2022.

[19] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS)*, vol. 54, 2017.

[20] C. Merkwirth and T. Lengauer, "Automatic generation of complementary descriptors with molecular graph networks," *Journal of Chemical Information and Modeling*, vol. 45, no. 5, pp. 1159–1168, 2005, PMID: 16180893. DOI: 10.1021/ci049613b. eprint: https://doi.org/10.1021/ci049613b. [Online]. Available: https://doi.org/10.1021/ci049613b.

[21] P. Mernyei and C. Cangea, "Wiki-cs: A wikipedia-based benchmark for graph neural networks," *arXiv preprint arXiv:2007.02901*, 2020.

[22] H. Peng, H. Li, Y. Song, V. Zheng, and J. Li, "Differentially private federated knowledge graphs embedding," in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, ser. CIKM '21, Virtual Event, Queensland, Australia: Association for Computing Machinery, 2021, pp. 1416–1425, ISBN: 9781450384469. DOI: 10.1145/3459637.3482252. [Online]. Available: https://doi.org/10.1145/3459637.3482252.

[23] L. Peng, N. Wang, N. Dvornek, X. Zhu, and X. Li, "Fedni: Federated graph learning with network inpainting for population-based disease prediction," *IEEE Transactions on Medical Imaging*, 2022.

[24] Y. Qiu, C. Huang, J. Wang, Z. Huang, and J. Xiao, "A privacy-preserving subgraph-level federated graph neural network via differential privacy," in *Knowledge Science, Engineering and Management*, 2022.

[25] 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. DOI: 10.1109/TNN.2008.2005605.

[26] Z. Shen, J. Cervino, H. Hassani, and A. Ribeiro, "An agnostic approach to federated learning with class imbalance," *ICLR*, 2022.

[27] S. Suresh, D. Godbout, A. Mukherjee, M. Shrivastava, J. Neville, and P. Li, *Federated graph representation learning using self-supervision*, 2022. DOI: 10.48550/ARXIV.2210.15120. [Online]. Available: https://arxiv.org/abs/2210.15120.

[28] B. Wang, A. Li, H. Li, and Y. Chen, "Graphfl: A federated learning framework for semi-supervised node classification on graphs," 2020.

[29] K. Wang, J. An, M. Zhou, Z. Shi, X. Shi, and Q. Kang, "Minority-weighted graph neural network for imbalanced node classification in social networks of internet of people," *IEEE Internet of Things Journal*, vol. 10, no. 1, pp. 330–340, 2022.

[30] C. Wu, F. Wu, Y. Cao, Y. Huang, and X. Xie, "Fedgnn: Federated graph neural network for privacy-preserving recommendation," *CoRR*, 2021.

[31] H. Xie, J. Ma, L. Xiong, and C. Yang, "Federated graph classification over non-iid graphs," *Advances in Neural Information Processing Systems*, vol. 34, pp. 18 839–18 852, 2021.

[32] M. Yang, X. Wang, H. Zhu, H. Wang, and H. Qian, "Federated learning with class imbalance reduction," in *2021 29th European Signal Processing Conference (EUSIPCO)*, IEEE, 2021, pp. 2174–2178.

[33] Z. Yang, W. W. Cohen, and R. Salakhutdinov, "Revisiting semi-supervised learning with graph embeddings," in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ser. ICML'16, New York, NY, USA: JMLR.org, 2016, pp. 40–48.

[34] J. Yoon, J. Jordon, and M. van der Schaar, "PATE-GAN: Generating synthetic data with differential privacy guarantees," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=S1zk9iRqF7.

[35] K. ZHANG, C. Yang, X. Li, L. Sun, and S. M. Yiu, "Subgraph federated learning with missing neighbor generation," in *Advances in Neural Information Processing Systems*, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., 2021. [Online]. Available: https://openreview.net/forum?id=SJHRf5nW93.