Distributed Training of Large Graph Neural Networks with Variable Communication Rates

Anonymous authors
Paper under double-blind review

Abstract

Training Graph Neural Networks (GNNs) on large graphs presents unique challenges due to the large memory and compute requirements. Distributed GNN training, where the graph is partitioned across multiple machines, is a common approach to train GNNs on large graphs. However, as the graph cannot generally be decomposed into small non-interacting components, data communication between the training machines quickly limits training speeds. Compressing the communicated node activations by a fixed amount improves the training speeds, but lowers the accuracy of the trained GNN. In this paper, we introduce a variable compression scheme for reducing the communication volume in distributed GNN training without compromising the accuracy of the learned model. Based on our theoretical analysis, we derive a variable compression method that converges to a solution that is equivalent to the full communication case. Our empirical results show that our method attains a comparable performance to the one obtained with full communication and that for any communication budget, we outperform full communication and any fixed compression ratio.

1 Introduction

Graph Neural Networks (GNNs) are a neural network architecture tailored for graph-structured data (Zhou et al., 2020; Wu et al., 2020; Bronstein et al., 2017). GNNs are multi-layered networks, where each layer is composed of a (graph) convolution and a point-wise non-linearity (Gama et al., 2018). GNNs have shown state-of-the-art performance in robotics (Gama et al., 2020a; Tzes et al., 2023), weather prediction (Lam et al., 2022), prediction of protein interactions (Jumper et al., 2021) and physical system interactions (Fortunato et al., 2022), to name a few. The success of GNNs can be attributed to some of their theoretical properties such as being permutation-invariant (Keriven and Peyré, 2019; Satorras et al., 2021), stable to perturbations of the graph (Gama et al., 2020b), transferable across graphs of different sizes (Ruiz et al., 2020), and their expressive power (Xu et al., 2018; Bouritsas et al., 2022; Kanatsoulis and Ribeiro, 2022; Chen et al., 2019).

In a GNN, the data is propagated through the graph via graph convolutions. In large-scale graphs, the data diffusion over the graph is costly in terms of compute and memory requirements. To overcome this limitation, several solutions were proposed. Some works have focused on the transferability properties of GNNs, i.e. training a GNN on a small graph, and deploying it on a large-scale graph (Ruiz et al., 2020). Other works have focused on training on a sequence of growing graphs (Cerviño et al., 2023). Though useful, these solutions either assume that an accuracy degradation is admissible (i.e. transferability bounds), or that all the graph data is readily accessible within the same training machine. These assumptions may not hold in practice, as we might need to recover the full centralized performance without having the data in a centralized manner.

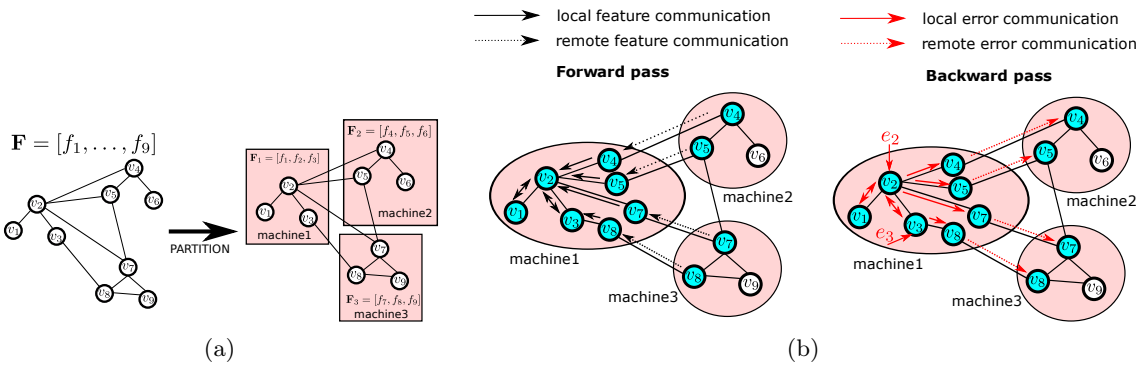

Figure 1: Distributed full-graph forward and backward pass for a GNN layer on a graph with 9 nodes distributed across 3 machines (a). The graph topology and input node features are partitioned among the 3 training machines. (b). The activity in machine 1 for the forward and backward passes. During the forward pass, machine 1 fetches the output features of the previous GNN layer of remote neighbors. During the backward pass, machine 1 sends back the gradients of the features fetched during the forward pass.

Real-world large-scale graph data typically cannot fit within the memory of a single machine or accelerator, which forces GNNs to be learned in a distributed manner. To do this efficiently, several solutions have been proposed. Distributing the datasets and learning a common representation, denoted Federated Learning is a useful technique (Bonawitz et al., 2019; Li et al., 2020a;b). The GNN FL counterpart has proven to be successful when the graph can be split into different machines(He et al., 2021; Mei et al., 2019). However, training locally while assuming no interaction between datasets is not always a reasonable assumption for graph data.

This work is drawn by two observations of GNNs. First, large graph datasets cannot be split into non-interacting pieces across a set of machines. Therefore, training GNNs distributively requires interaction between agents in the computation of the gradient updates. Second, the quality of the communicated data affects the performance of the trained model. In this paper, we posit that the compression rate in the communication between agents should vary between the different stages of the GNN training. Intuitively, at the early stages of training, the communication can be less reliable, but as training progresses, and we are required to estimate the gradient more precisely, the quality of the communicated data should improve.

In this paper, we consider the problem of efficiently learning a GNN across a set of machines, each of which has access to part of the graph data. Drawn from the observation that in a GNN, model parameters are significantly smaller than the graph's input and intermediate node feature, we propose to compress the intermediate GNN node features that are communicated between different machines. Given that this compression hurts the accuracy of the GNN, in this paper we introduce a variable compression scheme that trades-off the time needed to train a GNN distributively and the accuracy of the GNN.

The contributions of this paper are:

1. We present a novel algorithm to learn graph representations that compresses the data communicated between the training agents. We propose to vary the compression rate progressively, in order to achieve a comparable performance to the no-compression case at a fraction of the communication costs.

2. We theoretically show that our method converges to a first-order stationary point of the full graph training problem while taking distributed steps and compressing the communications.

3. We empirically show that our method attains a comparable performance to the full graph training scenario while incurring less communication costs. In particular, by plotting accuracy as a function of the communication costs, our method outperforms full communication and fixed compression rates in terms of accuracy achieved per communicated byte.

Related work

Mini-Batch Training. In the context of distributed GNN training, Zheng et al. (2020) proposes to distribute mini-batches between a set of machines, each of which computes a local gradient, updates the GNN, and communicates it back to the server. Zheng et al. (2020) uses METIS (Karypis and Kumar, 1998) to partition the graph, which reduces communication overhead and balances the computations between machines. Although (Zheng et al., 2020) provides good results in practice, the GNN does not process data on the full graph, only a partition of it, which can yield sub-optimal results compared to processing the full graph.

Memory Optimization. In this work we do full batch training, which has also been considered in the literature. Similar to us is *sequential aggregation and remasterization* (Mostafa, 2022), which sequentially re-constructs and frees pieces of the large GNN during the backward pass computation. Even in densely connected graphs, this deals with the memory limitations of the GNN, showing that the memory requirements per worker decrease linearly with the number of workers.

Quantization. A related approach to dealing with the limitations of a single server for training compression in communication is quantization in the architecture. In order to train GNNs more efficiently using quantization, the most salient examples are feature quantization (Ma et al., 2022), Binary Graph Neural Networks (Bahri et al., 2021), vector quantization (Ding et al., 2021), last bit quantization (Feng et al., 2020), and degree quantization (Tailor et al., 2020). Although related in spirit, quantizing a GNN is a local operation that differs from compressing the communication between servers.

2 Learning Graph Neural Networks

In this paper, we consider the problem of training GNNs on large-scale graphs that are stored in a set of machines. Formally, a graph is described by a set of vertices and edges $\mathcal{G} = (\mathbf{V}, \mathbf{E})$, where $\mathbf{V} = n$ is the set of vertices whose cardinality is equal to the number of nodes, and $\mathbf{E} \subseteq \mathbf{V} \times \mathbf{V}$ is the set of edges. The graph $\mathcal{G}$ can be represented by its adjacency matrix $\mathbf{S}$. Oftentimes, the graph includes vertex features, $F_V \in \mathbb{R}^{|\mathbf{V}| \times D_v}$, and edge features, $F_E \in \mathbb{R}^{|\mathbf{E}| \times D_v}$, where D_v and D_E are the vertex and edge feature dimensions, respectively. Graph problems fall into three main categories: node-level prediction where the goal is to predict properties of individual nodes; edge-level prediction where the goal is to predict edge features or predict the locations of missing edges; and graph-level prediction where the goal is to predict properties of entire graphs. In this paper, we focus on distributed training of GNNs for node classification problems. Our distributed training approach is still relevant to the other two types of graph problems, as they also involve a series of GNN layers, followed by readout modules for edge-level or graph-level tasks.

A GNN is a multi-layer network, where at each layer, messages are aggregated between neighboring nodes via graph convolutions. Formally, given a graph signal $\mathbf{x} \in \mathbb{R}^n$, where $[\mathbf{x}]_i$ represents the value

of the signal at node i, the graph convolution can be expressed as

$$\mathbf{z}_n = \sum_{k=0}^{K-1} h_k \mathbf{S}^k x_n, \tag{1}$$

where $\mathbf{h} = [h_0, \ldots, h_{K-1}] \in \mathbb{R}^K$ are the graph convolutional coefficients. In the case that $K = 2$, and $\mathbf{S}$ is binary, the graph convolution 1 translates into the `AGGREGATE` function in the so-called *message-passing neural networks*.

A GNN is composed of L layers, each of which is composed of a graph convolution (1) followed by a point-wise non-linear activation function ρ such as `ReLU`, or `tanh`. The l-th layer of a GNN can be expressed as,

$$\mathbf{X}_l = \rho\Big(\sum_{k=0}^{K-1} \mathbf{S}^k \mathbf{X}_{l-1} \mathbf{H}_{l,k} \Big), \tag{2}$$

where $\mathbf{X}_{l-1} \in \mathbb{R}^{n \times F_{l-1}}$ is the output of the previous layer (with $\mathbf{X}_0$ is equal to the input data $\mathbf{X} = [\mathbf{x}_0, \ldots, \mathbf{x}_0]$), and $\mathbf{H}_{l,k} \in \mathbb{R}^{F_{l-1} \times F_l}$ are the convolutional coefficients of layer l and hop k. We group all learnable coefficients $\mathcal{H} = \{\mathbf{H}_{l,k}\}_{l,k}$, and define the GNN as $\mathbf{\Phi}(\mathbf{x}, \mathbf{S}, \mathcal{H})$.

2.1 Empirical Risk Minimization

We consider the problem of learning a GNN that given an input graph signal $\mathbf{x}$ is able to predict an output graph signal $\mathbf{y}$ of an unknown distribution $p(X, Y)$,

$$\underset{\mathcal{H}}{\text{minimize}}\, \mathbb{E}_p[\ell(\mathbf{y}_i, \mathbf{\Phi}(\mathbf{x}_i, \mathbf{S}, \mathcal{H}))], \tag{SRM}$$

where ℓ is a non-negative loss function $\ell : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^+$, such that $\ell(\mathbf{y}, \hat{\mathbf{y}}) = 0$ iif $\mathbf{y} = \hat{\mathbf{y}}$. Common choices for the loss function are the cross-entropy loss and the mean square error. Problem (SRM) is denoted called Statistical Risk Minimization (Vapnik, 2013), and the choice of GNN for a parameterization is justified by the structure of the data, and the invariances in the graph Bronstein et al. (2017). However, problem (SRM) cannot be solved in practice given that we do not have access to the underlying probability distribution p. In practice, we assume to be equipped with a dataset $\mathcal{D} = \{x_i, y_i\}_i$ drawn i.i.d. from the unknown probability $p(X, Y)$ with $|\mathcal{D}| = m$ samples. We can therefore obtain the empirical estimator of (SRM) as,

$$\underset{\mathcal{H}}{\text{minimize}}\, \frac{1}{m} \sum_{i=1}^{m} \ell(\mathbf{y}_i, \mathbf{\Phi}(\mathbf{x}_i, \mathbf{S}, \mathcal{H})). \tag{ERM}$$

The empirical risk minimization problem (ERM) can be solved in practice given that it only requires access to a set of samples $\mathcal{D}$. The solution to problem ERM will be close to (SRM) given that the samples are i.i.d., and that there is a large number of samples m(Vapnik, 2013). To solve problem (ERM), we will resort to gradient descent, and we will update the coefficients $\mathcal{H}$ according to,

$$\mathcal{H}_{t+1} = \mathcal{H}_t - \eta_t \nabla_{\mathcal{H}} \ell(\mathbf{y}_i, f(\mathbf{x}_i, \mathbf{S}, \mathcal{H})), \tag{SGD}$$

where t represents the iteration, and η_t the step-size. In a centralized manner computing iteration (SGD) presents no major difficulty, and it is the usual choice of algorithm for training a GNN. When the size of the graph becomes large, and the graph data is partitioned across a set of agents, iteration (SGD) requires communication. In this paper, we consider the problem of solving the empirical risk minimization problem (ERM) through gradient descent (SGD) in a decentralized and efficient way.

3 Distributed GNN training

Consider a set $\mathcal{Q}, |\mathcal{Q}| = Q$ of machines that jointly learn a single GNN $\mathbf{\Phi}$. Each machine $q \in \mathcal{Q}$ is equipped with a subset of the graph $\mathbf{S}$, and node data, as shown in Fig. 1a. Each machine is responsible for computing the features of the nodes in its local partitions for all layers in the GNN. The GNN model $\mathcal{H}$ is replicated across all machines. In order to learn a GNN, we update the weights according to gradient descent (SGD), and average them across machines. This procedure is similar to the standard `FedAverage` algorithm used in *Federated Learning*(Li et al., 2020b).

Note that the gradient descent iteration (SGD) cannot be computed without communication given that the data in $(\mathbf{y}_i, \mathbf{x}_i)$ is distributed in the Q machines. Therefore, in order to compute the gradient step (SGD), we need the machines to communicate graph data. The simplest way to communicate could be to transmit the data of each node in the adjacent machines. That is, communicate the input feature x_j for each $j \in \mathcal{N}_i^k, j \in q^{'}$. That is to say, for each node that we would like to classify, we would require all the graph data belonging to the k-hop neighborhood graph. This procedure, however, is costly and grows with the size of the graph, which renders it unimplementable in practice.

As opposed to communicating the node features, and the graph structure for each node in the adjacent machine, we propose to communicate the features and activation of the nodes at the nodes in the boundary. Note that in this procedure the bits communicated between machines do not depend on the number of nodes in the graph and the number of features compressed can be controlled by the width of the architecture used (see Appendix A). The only computation overhead that this method requires is computing the value of the GNN at every layer for a given subset of nodes using local information only.

3.1 Computing the Gradient Using Remote Compressed Data

Following the framework of communicating the intermediate activation, in order to compute a gradient step (SGD), we require 3 communication rounds (i) the forward pass, in which each machine fetches the feature vectors of remote neighbors and propagates their messages to the nodes in the local partition, (ii) the backward pass, in which the gradients flow in the opposite direction and are accumulated in the GNN model weights, and (iii) the aggregation step in which the weight gradients are summed across all machines and used to update the GNN model weights. The communication steps for a single GNN layer are illustrated in Fig. 1b.

To compute a gradient step, we first compute the forward pass, for which we need the output of the GNN at a node i. To compute the output of a GNN at node i, we need to evaluate the GNN according to the graph convolution (1). Note that to evaluate (1), we require access to the value of the input features x_j for each $j \in \mathcal{N}_i^k$, which might not be on the same client as i. In this paper, we propose that the clients with nodes in $\mathcal{N}_i^k$, compute the forward passes *locally* (i.e. using only the nodes in their client), and communicate the the compressed outputs.

Once the values of the activations are received, they are decompressed, processed by the local machine, and the output of the GNN is obtained. To obtain the gradient of the loss ℓ, the output of the GNN is compared to the true label, and the gradient with respect to the parameters of the GNN is computed. The errors are once again compressed and communicated back to the clients. Which, will update the values of the parameters of the GNN $\mathcal{H}_{t_1}^q$. After every client updates the values of the parameters $\mathcal{H}$, there is a final round of communication where the values are averaged. Note that this procedure allows the GNN to be updated using the whole graph. The communication costs are reduced given that the number of bits communicated is controlled by the compressing-decompressing mechanism. The compressing and decompressing mechanism can be modeled as follows,

Definition 1 *The compression and decompression mechanism $g_{\epsilon,r}, g_{\epsilon^{-1},r}$ with compression error ϵ, and compression rate c, satisfies that given a set of parameters $\mathbf{x} \in \mathbb{R}^n$, when compressed and decompressed, the following relation holds i.e.,*

$$\mathbf{z} = g_{\epsilon,c}(\mathbf{x}), \text{ and } \tilde{\mathbf{x}} = g_{\epsilon}^{-1}(g_{\epsilon}(\mathbf{x})) \text{ and } \mathbb{E}[\|\tilde{\mathbf{x}} - \mathbf{x}\|] \leq \delta \text{ with } \mathbb{E}[||\tilde{\mathbf{x}} - \mathbf{x}||^2] \leq \epsilon^2, \tag{3}$$

where $\mathbf{z} \in \mathbb{R}^{n/c}$ with $n/c \in \mathbb{Z}$ is the compressed signal. If $\delta = 0$, the compression is lossless.

Intuitively, the error ϵ, and compression rate c are related; a larger compression rate c will render a larger compression error ϵ. Also, signals that are more compressed require less bandwidth to be communicated. Relying on the compression mechanism (3), a GNN trained using a *fixed* compression ratio c during training, will converge to a neighborhood of the optimal solution. The size of the neighborhood will be given by the variance of the compression error ϵ^2.

AS1 *The loss ℓ function has L Lipschitz continuous gradients, $||\nabla \ell(\mathbf{y}_1, \mathbf{x}) - \nabla \ell(\mathbf{y}_2, \mathbf{x})|| \leq L||\mathbf{y}_1 - \mathbf{y}_2||$.*

AS2 *The empirical estimator of the gradient $\hat{\nabla}_{\mathcal{H}} \ell(y_i, \Phi(x_i, \mathbf{S}; \mathcal{H}_t))$ is an unbiased estimator of the gradient $\nabla_{\mathcal{H}} \ell(y_i, \Phi(x_i, \mathbf{S}; \mathcal{H}_t))$, and the variance is σ^2,*

$$\mathbb{E}\left[||\hat{\nabla}_{\mathcal{H}} \ell(y, \Phi(x, \mathbf{S}; \mathcal{H}_t)) - \nabla_{\mathcal{H}} \ell(y, \Phi(x, \mathbf{S}; \mathcal{H}_t))||^2\right] \leq \sigma^2. \tag{4}$$

AS3 *The non-linearity ρ is normalized Lipschitz.*

AS4 *The GNN, and its gradients are M-Lipschitz with respect to the parameters $\mathcal{H}$.*

AS5 *The graph convolutional filters in every layer of the graph neural network are bounded, i.e.*

$$||h_{*\mathbf{S}} x|| \leq ||x|| \lambda_{max}\left(\sum_{t=0}^{T} h_t \mathbf{S}^t\right), \text{ with } \lambda_{max}\left(\sum_{t=0}^{T} h_t \mathbf{S}^t\right) < \infty. \tag{5}$$

Assumption 1 holds for most losses used in practice over a compact set. Assumption 2 is true for i.i.d. data, the exact characterization of σ depends on the architecture. Assumption 3 holds for most non-linearities used in practice over normalized signals. Assumption 4 is a standard assumption, and the exact characterization is an active area of research (Fazlyab et al., 2019). Assumption 5 holds in practice when the weights are normalized.

Proposition 1 (Convergence of GNNs Trained on Fixed Compression) *Under assumptions 1 through 5, consider the iterates generated by equation* (SGD) *where the normalized signals $\mathbf{x}$ are compressed using compression rate c with corresponding error ϵ(cf. Definition 3). Consider an L layer GNN with F, and K features and coefficients per layer respectively. Let the step-size be $\eta \leq 1/L_{\nabla}$, with $L_{\nabla} = 4M\lambda_{max}^{L}\sqrt{KFL}$ if the compression error is such that at every step k, and any $\beta > 0$,*

$$\mathbb{E}[||\nabla_{\mathcal{H}} \ell(y, \Phi(x, \mathbf{S}; \mathcal{H}_t))||^2] \geq L_{\nabla}^2 \epsilon^2 + \sigma^2 + \beta, \tag{6}$$

then the fixed compression mechanism converges to a neighborhood of the first-order stationary point of SRM in $K \leq \mathcal{O}(\frac{1}{\beta})$ iterations, i.e.,

$$\mathbb{E}[||\nabla_{\mathcal{H}} \ell(y, \Phi(x, \mathbf{S}; \mathcal{H}_t))||^2] \leq L_{\nabla}^2 \epsilon^2 + \sigma^2 + \beta. \tag{7}$$

Algorithm 1 VARCO: Distributed Graph Training with **VAR**iable **CO**mmunication Rates

Split graph $\mathcal{G}$ into W partitions and assign them to each worker w_i
Initialize GNN with weights $\mathcal{H}_0$ and distribute to all workers w_i
Fix initial compression rate c_0, and scheduler r
repeat
Each Worker w_i: Compute forward pass by using non-compressed activations for local nodes and compressed activations for non-local nodes that are fetched from other machines.
Each Worker w_i: Compute parameter gradients by backpropagating errors across machines and through the differentiable compression routine. Apply the gradient step to the parameters in each worker.
Server: Average parameters from workers, and send them back to workers.
Update compression rate c_{t+1}
until Convergence

Proof: The proof can be found in Appendix C ■

Proposition 1 is important because it allows us to show that the fixed compression mechanism is able to converge to a neighborhood of the first-order stationary point of 1. The size of the neighborhood can be controlled by the compression rate ϵ, as well as the variance of the estimator σ. Although useful, Proposition 1, presents a limitation on the quality of the solution that can be obtained through compression. In what follows we introduce a variable compression scheme that is able to trade-off between efficient communication and sub-optimality guarantees.

4 VARCO - Variable Compression For Distributed GNN

In this paper, we propose variable compression rates as a way to close the gap between training in a centralized manner and efficient training in a distributed manner. We use proposition (1) as a stepping stone towards a training mechanism that reduces the compression ratio r_t as the iterates progress. We begin by defining a scheduler $s : \mathbb{Z} \to \mathbb{R}$ as a monotonically decreasing function that given a train step $t \in \mathbb{Z}$, it returns a compression ratio $s(t)$, such that $r(k^{'}) \leq r(k)$ if $k^{'} \geq k$. The scheduler r will be in charge of reducing the compression ratio as the iterates increase. An example of scheduler can be the linear $r_{lin}(k) = \frac{c_{min}-c_{max}}{K}k + c_{max}$ scheduler (see Appendix 2). Note that the previous section considers a constant scheduler $r(k) = c$. In this paper, we propose to train GNN by following a compression scheme by a scheduler r, given a GNN architecture, a number of clients C, a dataset $\mathcal{D}$.

After choosing a scheduler r, for every step t, we obtain the compression rate $r(t)$. During the forward pass, we compute the output of the GNN at a node n using the local data, and the compressed data from adjacent machines. The compress data, encompasses both the features at the adjacent nodes, as well as the compress activations of the intermediate layers for that node. The compression mechanism, compresses the values of the GNN using scheduler $r(t)$, and communicates them the to machine that own node n. The backward pass, receives the gradient from the machine that own node n, and updates the values of the GNN. After the GNN values are updates, the coefficients of the GNN are communicated to a centralized agent that averages them and sends them back to the machines. A more succinct explanation of the aforementioned procedure can be seen in Algorithm 1.

Algorithm	OGBN-Products				OGBN-Arxiv			
	Number of Servers							
	2	4	8	16	2	4	8	16
Full Comm	78.46	78.43	78.41	78.44	69.64	69.04	69.71	69.76
No Comm	77.2	75.09	72.28	69.05	65.74	59.96	57.36	55.46
Fixed Comp Rate 2	79.57	78.86	78.25	78.38	66.71	66.48	65.59	64.62
Fixed Comp Rate 4	79.32	78.13	76.94	75.58	66.08	65.53	63.58	64.38
Fixed Comp Rate 8	78.83	77.49	74.53	70.45	67.41	66.13	63.39	62.5
Fixed Comp Rate 16	78.52	76.6	70.25	70.04	68.21	66.58	65.45	62.06
Fixed Comp Rate 32	78.64	75.26	71.29	68.31	67.76	66.14	64.74	63.94
Fixed Comp Rate 64	78.57	75.53	70.10	67.19	68.43	67.16	64.04	64.8
Variable Compression	79.27	79.06	79.75	78.72	68.12	68.76	69.02	68.01

Table 1: Accuracy results when training GNNs with full-communication, no communication, fixed and variable compression in both OGBN-Arxiv, and OGBN-Products. We test our algorithm with a GNN based on Graph-Sage, and Graph Convolutions for $2, 10$ and 20 clients.

4.1 Convergence of the VARCO Algorithm

We characterize the convergence of the VARCO algorithm in the following proposition.

Proposition 2 (Scheduler Convergence) *Under assumptions 1 through 5, consider the iterates generated by equation* (SGD) *where the normalized signals* $\mathbf{x}$ *are compressed using compression rate* r *with corresponding error* ϵ*(cf. Definition 3). Consider an* L *layer GNN with* F*, and* K *features and coefficients per layer respectively. Let the step-size be* $\eta \leq 1/L_{\nabla}$*, with* $L_{\nabla} = 4M\lambda_{max}^{L}\sqrt{KFL}$*. Consider a scheduler such that the compression error* ϵ_t *decreases at every step* $\epsilon_{t+1} < \epsilon_t$*, then*

$$\mathbb{E}[||\nabla_{\mathcal{H}}\ell(y, \Phi(x, \mathbf{S}; \mathcal{H}_t))||^2] \leq \sigma^2. \tag{8}$$

happens infinitely often.

Proof: The proof can be found in Appendix D. ■

According to Proposition 2, for any scheduler, we are able to obtain an iterate t, whose gradient is smaller than σ^2. The conditions on the scheduler are simple to satisfy, the compression error ϵ needs to decrease on every step (see Appendix D). It is important to note that the scheduler does not require information about the gradient of the GNN. Compressing the activations in a GNN can prove to reduce the communication required to train it. However, keeping a fixed compression ratio might not be enough to obtain a GNN of a comparable accuracy than the one trained using no compression. By using a variable compression for the communications, we obtain the best of both worlds – we reduce the communication overhead needed to train a GNN, without compromising the overall accuracy of the learned solution. The key observation is that in the early stages of training, an estimator of the gradient with larger variance is acceptable, while in the later stages, a more precise estimator needs to be used.

5 Experiments

To illustrate the benefits our method we tested our method in 2 real-world datasets: OGBN-Arxiv (Wang et al., 2020) and OGBN-Products (Bhatia et al., 2016). In the case of OGBN-Arxiv, the graph has $169,343$ nodes, and $1,166243$ edges and it represents the citation network between computer

science arXiv papers. The node features are 128 dimensional embeddings of the title and abstract of each paper Mikolov et al. (2013). The objective is to predict which of 40 categories does the paper belong to. In the case of OGBN-Products, the graph represents products that were bought together on an online marketplace. There are $2,449,029$ nodes, that represent products, and $61,859,140$ edges between nodes represent that those products were bought together. Feature vectors are 100 dimensional and the objective is to classify each node into 47 categories. For each dataset, we partition the graph at random and distribute it over $\{2, 4, 8, 16\}$ machines. In all cases, we trained for 300 epochs, and we reported the best accuracy achieved by the model over the unseen data. We benchmarked `VARCO` against full-communication, no intermediate communication, and fixed compression for $\{2, 4, 8, 16, 32, 64\}$ compression ratios. For the GNNs, we used a 3 layered GNN with 256 hidden units per layer, `ReLU` non-linearity, and `SAGE` convolution (Hamilton et al., 2017). For `VARCO`, we used a linear compression with slope 2, and 128 and 1.5 maximum and minimum compression ratio respectively (see Appendix D).

In Table 1 we show the results of our Algorithm `VARCO`. For all configurations, `VARCO` attains a comparable performance to the full communication setup. It should also be noted that `VARCO` is robust given that it does not depend on the the number of servers or the dataset. Table 1 therefore validates the claim that our `VARCO` attains a comparable performance to the one obtained with full communication. In Figure 2, we show the accuracy as a function of the communicated floating points values for OGBN-Arxiv with 16 machines. In Figure 2, we show that `VARCO` obtains the most efficient training curve in terms of communication costs, given that the variable compression line is above all other curves. Intuitively, all learning curves show a similar slope at the beginning of training, and they converge to different values in the later stages, decreasing the compression rate is an efficient way of learning a GNN as shown in Figure 2. Given that the `VARCO` curve in Figure 2 is above all curves, this means that for any communication budget i.e. number of bits, `VARCO` obtains the best accuracy of all methods considered. This indeed validates our claim that using `VARCO` is an efficient way of training a GNN. In all, our experiments on real-world datasets show that `VARCO` is an efficient way of obtaining GNNs that attain a comparable performance to the one obtained using full communication, but, at a fraction of the communication costs. In Appendix D, we show different combinations of variable compressions, all of which attain same similar results, showcasing the robustness of our algorithm.

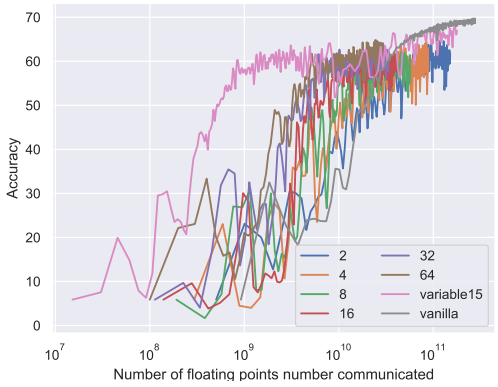

Figure 2: Accuracy of the GNNs as a function of the communicated floats for the OGBN-Arxiv with 16 machines.

6 Conclusion

In this paper, we presented a distributed method to train GNNs by compressing the communications. We theoretically showed that our method converges to a neighborhood of the optimal solution. Empirically, we benchmarked our algorithm in two real-world datasets and showed that our method obtains a GNN that attains a comparable performance to the one trained on full communication, at a fraction of the communication costs.

References

Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI open*, 1:57–81, 2020.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.

Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

Fernando Gama, Antonio G Marques, Geert Leus, and Alejandro Ribeiro. Convolutional neural network architectures for signals supported on graphs. *IEEE Transactions on Signal Processing*, 67(4):1034–1049, 2018.

Fernando Gama, Qingbiao Li, Ekaterina Tolstaya, Amanda Prorok, and Alejandro Ribeiro. Decentralized control with graph neural networks. *arXiv preprint arXiv:2012.14906*, 2020a.

Mariliza Tzes, Nikolaos Bousias, Evangelos Chatzipantazis, and George J Pappas. Graph neural networks for multi-robot active information acquisition. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3497–3503. IEEE, 2023.

Remi Lam, Alvaro Sanchez-Gonzalez, Matthew Willson, Peter Wirnsberger, Meire Fortunato, Alexander Pritzel, Suman Ravuri, Timo Ewalds, Ferran Alet, Zach Eaton-Rosen, et al. Graphcast: Learning skillful medium-range global weather forecasting. *arXiv preprint arXiv:2212.12794*, 2022.

John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021.

Meire Fortunato, Tobias Pfaff, Peter Wirnsberger, Alexander Pritzel, and Peter Battaglia. Multiscale meshgraphnets. In *ICML 2022 2nd AI for Science Workshop*, 2022.

Nicolas Keriven and Gabriel Peyré. Universal invariant and equivariant graph neural networks. *Advances in Neural Information Processing Systems*, 32, 2019.

Vıctor Garcia Satorras, Emiel Hoogeboom, and Max Welling. E (n) equivariant graph neural networks. In *International conference on machine learning*, pages 9323–9332. PMLR, 2021.

Fernando Gama, Joan Bruna, and Alejandro Ribeiro. Stability properties of graph neural networks. *IEEE Transactions on Signal Processing*, 68:5680–5695, 2020b.

Luana Ruiz, Luiz Chamon, and Alejandro Ribeiro. Graphon neural networks and the transferability of graph neural networks. *Advances in Neural Information Processing Systems*, 33:1702–1712, 2020.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2018.

Giorgos Bouritsas, Fabrizio Frasca, Stefanos Zafeiriou, and Michael M Bronstein. Improving graph neural network expressivity via subgraph isomorphism counting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(1):657–668, 2022.

Charilaos I Kanatsoulis and Alejandro Ribeiro. Graph neural networks are more powerful than we think. *arXiv preprint arXiv:2205.09801*, 2022.

Zhengdao Chen, Soledad Villar, Lei Chen, and Joan Bruna. On the equivalence between graph isomorphism testing and function approximation with gnns. *arXiv preprint arXiv:1905.12560*, 2019.

Juan Cerviño, Luana Ruiz, and Alejandro Ribeiro. Learning by transference: Training graph neural networks on growing graphs. *IEEE Transactions on Signal Processing*, 71:233–247, 2023.

Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konečnỳ, Stefano Mazzocchi, Brendan McMahan, et al. Towards federated learning at scale: System design. *Proceedings of machine learning and systems*, 1: 374–388, 2019.

Li Li, Yuxi Fan, Mike Tse, and Kuo-Yi Lin. A review of applications in federated learning. *Computers & Industrial Engineering*, 149:106854, 2020a.

Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated learning: Challenges, methods, and future directions. *IEEE signal processing magazine*, 37(3):50–60, 2020b.

Chaoyang He, Keshav Balasubramanian, Emir Ceyani, Carl Yang, Han Xie, Lichao Sun, Lifang He, Liangwei Yang, Philip S Yu, Yu Rong, et al. Fedgraphnn: A federated learning system and benchmark for graph neural networks. *arXiv preprint arXiv:2104.07145*, 2021.

Guangxu Mei, Ziyu Guo, Shijun Liu, and Li Pan. Sgnn: A graph neural network based federated learning approach by hiding structure. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 2560–2568. IEEE, 2019.

Da Zheng, Chao Ma, Minjie Wang, Jinjing Zhou, Qidong Su, Xiang Song, Quan Gan, Zheng Zhang, and George Karypis. Distdgl: distributed graph neural network training for billion-scale graphs. In *2020 IEEE/ACM 10th Workshop on Irregular Applications: Architectures and Algorithms (IA3)*, pages 36–44. IEEE, 2020.

George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, 20(1):359–392, 1998.

Hesham Mostafa. Sequential aggregation and rematerialization: Distributed full-batch training of graph neural networks on large graphs. *Proceedings of Machine Learning and Systems*, 4:265–275, 2022.

Yuxin Ma, Ping Gong, Jun Yi, Zhewei Yao, Cheng Li, Yuxiong He, and Feng Yan. Bifeat: Supercharge gnn training via graph feature quantization. *arXiv preprint arXiv:2207.14696*, 2022.

Mehdi Bahri, Gaétan Bahl, and Stefanos Zafeiriou. Binary graph neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9492–9501, 2021.

Mucong Ding, Kezhi Kong, Jingling Li, Chen Zhu, John Dickerson, Furong Huang, and Tom Goldstein. Vq-gnn: A universal framework to scale up graph neural networks using vector quantization. *Advances in Neural Information Processing Systems*, 34:6733–6746, 2021.

Boyuan Feng, Yuke Wang, Xu Li, Shu Yang, Xueqiao Peng, and Yufei Ding. Sgquant: Squeezing the last bit on graph neural networks with specialized quantization. In *2020 IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 1044–1052. IEEE, 2020.

Shyam Anil Tailor, Javier Fernandez-Marques, and Nicholas Donald Lane. Degree-quant: Quantization-aware training for graph neural networks. In *International Conference on Learning Representations*, 2020.

Vladimir Vapnik. *The nature of statistical learning theory*. Springer science & business media, 2013.

Mahyar Fazlyab, Alexander Robey, Hamed Hassani, Manfred Morari, and George Pappas. Efficient and accurate estimation of lipschitz constants for deep neural networks. *Advances in Neural Information Processing Systems*, 32:11427–11438, 2019.

Kuansan Wang, Zhihong Shen, Chiyuan Huang, Chieh-Han Wu, Yuxiao Dong, and Anshul Kanakia. Microsoft academic graph: When experts are not enough. *Quantitative Science Studies*, 1(1): 396–413, 2020.

K. Bhatia, K. Dahiya, H. Jain, P. Kar, A. Mittal, Y. Prabhu, and M. Varma. The extreme classification repository: Multi-label datasets and code, 2016. URL `http://manikvarma.org/downloads/XC/XMLRepository.html`.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In C.J. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013. URL `https://proceedings.neurips.cc/paper_files/paper/2013/file/9aa42b31882ec039965f3c4923ce901b-Paper.pdf`.

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

Dimitri P Bertsekas and John N Tsitsiklis. Gradient convergence in gradient methods with errors. *SIAM Journal on Optimization*, 10(3):627–642, 2000.

Rick Durrett. *Probability: Theory and Examples*. Cambridge University Press, 2019.