NEIGHBORHOOD GRADIENT CLUSTERING: AN EFFICIENT DECENTRALIZED LEARNING METHOD FOR NON-IID DATA DISTRIBUTIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Decentralized learning algorithms enable the training of deep learning models over large distributed datasets generated at different devices and locations, without the need for a central server. In practical scenarios, the distributed datasets can have significantly different data distributions across the agents. The current state-of-the-art decentralized algorithms mostly assume the data distributions to be Independent and Identically Distributed (IID). This paper focuses on improving decentralized learning over non-IID data distributions with minimal compute and memory overheads. We propose *Neighborhood Gradient Clustering (NGC)*, a novel decentralized learning algorithm that modifies the local gradients of each agent using self- and cross-gradient information. Cross-gradients for a pair of neighboring agents are the derivatives of the model parameters of an agent with respect to the dataset of the other agent. In particular, the proposed method replaces the local gradients of the model with the weighted mean of the self-gradients, model-variant cross-gradients (derivatives of the received neighbors' model parameters with respect to the local dataset - computed locally), and data-variant cross-gradients (derivatives of the local model with respect to its neighbors' datasets - received through communication). The data-variant cross-gradients are aggregated through an additional communication round without breaking the privacy constraints of the decentralized setting. Further, we present *CompNGC*, a compressed version of NGC that reduces the communication overhead by $32\times$ through cross-gradient compression. We demonstrate the empirical convergence and efficiency of the proposed technique over non-IID data distributions sampled from the CIFAR-10 dataset on various model architectures and graph topologies. Our experiments demonstrate that *NGC* and *CompNGC* outperform the existing state-of-the-art (SoTA) decentralized learning algorithm over non-IID data by $1-5\%$ with significantly less compute and memory requirements. Further, we also show that the proposed *NGC* method outperforms the baseline by $5-40\%$ with no additional communication.

1 INTRODUCTION

The remarkable success of deep learning is mainly attributed to the availability of humongous amounts of data and compute power. Large amounts of data is generated on a daily basis at different devices all over the world which could be used to train powerful deep learning models. Collecting such data for centralized processing is not practical because of the communication and privacy constraints. To address this concern, a new interest in developing distributed learning algorithms Agarwal & Duchi (2011) has emerged. Federated learning (centralized learning) Konečnỳ et al. (2016) is a popular setting in the distributed machine learning paradigm, where the training data is kept locally at the edge devices and a global shared model is learnt by aggregating the locally computed updates through a coordinating central server. Such a setup requires continuous communication with a central server which becomes a potential bottleneck Haghighat et al. (2020). This has motivated the advancements in decentralized machine learning.

Decentralized machine learning is a branch of distributed learning which focuses on learning from data distributed across multiple agents/devices. Unlike Federated learning, these algorithms assume

that the agents are connected peer to peer without a central server. It has been demonstrated that decentralized learning algorithms Lian et al. (2017) can perform comparable to centralized algorithms on benchmark vision datasets. Lian et al. (2017) present Decentralised Parallel Stochastic Gradient Descent (D-PSGD) by combining SGD with gossip averaging algorithm Xiao & Boyd (2004). Further, the authors analytically show that the convergence rate of D-PSGD is similar to its centralized counterpart Dean et al. (2012). Balu et al. (2021) propose and analyze Decentralized Momentum Stochastic Gradient Descent (DMSGD) which introduces momentum to D-PSGD. Assran et al. (2019) introduce Stochastic Gradient Push (SGP) which extends D-PSGD to directed and time varying graphs. Tang et al. (2019); Koloskova et al. (2019) explore error-compensated compression techniques (Deep-Squeeze and CHOCO-SGD) to reduce the communication cost of P-DSGD significantly while achieving same convergence rate as centralized algorithms. Aketi et al. (2021) combined Deep-Squeeze with SGP to propose communication efficient decentralized learning over time-varying and directed graphs. Recently, Koloskova et al. (2020) proposed a unified framework for the analysis of gossip based decentralized SGD methods and provide the best known convergence guarantees.

The key assumption to achieve state-of-the-art performance by all the above mentioned decentralized algorithms is that the data is independent and identically distributed (IID) across the agents. In particular, the data is assumed to be distributed in a uniform and random manner across the agents. This assumption does not hold in most of the real-world applications as the data distributions across the agent are significantly different (non-IID) based on the user pool Hsieh et al. (2020). The effect of non-IID data in a peer-to-peer decentralized setup is a relatively under-studied problem. There are only a few works that try to bridge the performance gap between IID and non-IID data distributions for a decentralized setup. Note that, we mainly focus on a common type of non-IID data, widely used in prior works Tang et al. (2018); Hsieh et al. (2020); Lin et al. (2021); Esfandiari et al. (2021): a skewed distribution of data labels across agents. Tang et al. (2018) proposed D^2 algorithm that extends D-PSGD to non-IID data distribution. However, the algorithm was demonstrated on only a basic LENET model and is not scalable to deeper models with normalization layers. Lin et al. (2021) replace local momentum with Quasi-Global Momentum (QGM) and improve the test performance by $1 - 20\%$. But the improvement in accuracy is only $1 - 2\%$ in case of highly skewed data distribution as shown in Aketi et al. (2022). Most recently, Esfandiari et al. (2021) proposed Cross-Gradient Aggregation (CGA) and a compressed version of CGA (CompCGA), claiming state-of-the-art performance for decentralized learning algorithms over completely non-IID data. CGA aggregates *cross-gradient* information, i.e., derivatives of its model with respect to its neighbors' datasets through an additional communication round. It then updates the model using projected gradients based on quadratic programming. The cross-gradient and self-gradient terms are formally defined in Section 3. CGA and CompCGA require a very slow quadratic programming step Goldfarb & Idnani (1983) after every iteration for gradient projection which is both compute and memory intensive. This work focuses on the following question: *Can we improve the performance of decentralized learning over non-IID data with minimal compute and memory overhead?*

In this paper, we propose *Neighborhood Gradient Clustering* (*NGC*) to handle non-IID data distributions in peer-to-peer decentralized learning setups. Firstly, we classify the gradients available at each agent into three types, namely self-gradients, model-variant cross-gradients, and data-variant cross-gradients (see Section 3). The self-gradients (or local gradients) are the derivatives computed at each agent on its model parameters with respect to the local dataset. The model-variant cross-gradients are the derivatives of the received neighbors' model parameters with respect to the local dataset. These gradients are computed locally at each agent after receiving the neighbors' model parameters. Communicating the neighbors' model parameters is a necessary step in any gossip based decentralized algorithm Lian et al. (2017). The data-variant cross-gradients are the derivatives of the local model with respect to its neighbors' datasets. These gradients are obtained through an additional round of communication. We then cluster the gradients into a) *model-variant cluster* with self-gradients and model-variant cross-gradients, and b) *data-variant cluster* with self-gradients and data-variant cross-gradients. Finally, the local gradients are replaced with the weighted average of the cluster means. The main motivation behind this modification is to account for the high variation in the computed local gradients (and in turn the model parameters) across the neighbors due to the non-IID nature of the data distribution.

The proposed technique has two rounds of communication at every iteration to send model parameters and data-variant cross-gradients which incurs $2\times$ communication cost compared to tra-

ditional decentralized algorithms (D-PSGD). To reduce the communication overhead, we propose compressed version of *NGC* (*CompNGC*) by compressing the additional round of cross-gradient communication. Moreover, if the weight associated with data-variant cluster is set to 0 then *NGC* does not require an additional round of communication. We validate the performance of proposed algorithm on CIFAR10 dataset over various model architectures and graph topologies. We compare the proposed algorithm with D-PSGD, CGA, and CompCGA and show that we can achieve superior performance over non-IID data compared to the current state-of-the-art approach. We also report the order of communication, memory and compute overheads required for *NGC* and CGA as compared to D-PSGD.

Contributions: In summary, we make the following contributions.

- We propose Neighborhood Gradient Clustering (*NGC*) for decentralized learning setting that utilizes self-gradients, model-variant cross-gradients and data-variant cross-gradients to improve the learning over non-IID data distribution among agents.
- We present compressed version of Neighborhood Gradient Clustering (*CompNGC*) that reduces the additional round of cross-gradients communication by $32\times$.
- We show that the proposed method improves the performance by $1-5\%$ depending on graph topology and model architecture with significantly less compute and memory requirements compared to current state-of-the-art decentralized learning algorithm over non-IID data at iso-communication cost.
- We also show that when the weight associated with data-variant cross-gradients is set to 0, *NGC* performs $5-40\%$ better than D-PSGD without any communication overhead.

2 BACKGROUND

In this section, we provide the background on decentralized learning algorithm with peer-to-peer connections.

Algorithm 1 Decentralized Peer-to-Peer Training (*D-PSGD* with momentum)

Input: Each agent $i \in [1, n]$ initializes model weights $x_i^{(0)}$, learning rate γ, averaging rate η, mixing matrix $W = [w_{ij}]_{i,j\in[1,n]}$, and I_{ij} are elements of $n \times n$ identity matrix.

Each agent simultaneously implements the TRAIN( ) procedure
procedure TRAIN( )
  for t=$0, 1, \ldots, T-1$ **do**
    $d_i^t \sim D_i$ // sample data from training dataset.
    $g_i^t = \nabla_x f_i(d_i^t; x_i^t)$ // compute the local gradients
    $v_i^t = \beta v_i^{(t-1)} - \eta g_i^t$ // momentum step
    SENDRECEIVE(x_i^t) // share model parameters with neighbors $N(i)$.
    $\widetilde{x}_i^t = x_i^t + v_i^t$ // update the model
    $x_i^{(t+1)} = \widetilde{x}_i^t + \gamma \sum_{j\in\mathcal{N}(i)} (w_{ij} - I_{ij}) * x_j^t$ // gossip averaging step
  end
return

The main goal of the decentralized machine learning is to learn a global model using the knowledge extracted from the locally generated and stored data samples across n edge devices/agents while maintaining the privacy constraints. In particular, we solve the optimization problem of minimizing global loss function $f(x)$ distributed across n agents as given in equation. 1. Note that f_i is a local loss function (for example, cross entropy loss) defined in terms of the data sampled (d_i) from the local dataset D_i at agent i with model parameters x_i.

$$\min_{x\in\mathbb{R}^d} f(x) = \frac{1}{n}\sum_{i=1}^{n} F_i(D_i; x_i), \quad \text{and} \quad F_i(x_i, D_i) = \mathbb{E}_{d_j\in D_i}[f_i(d_j, x_i)] \;\; \forall i \tag{1}$$

This is typically achieved by combining local stochastic gradient descent Bottou (2010) with global consensus based gossip averaging Xiao & Boyd (2004). The communication topology in this peer-to-peer setup is modeled as a graph $G = ([n], E)$ with edges $\{i, j\} \in E$ if and only if agents i and j are connected by a communication link exchanging the messages directly. We represent $\mathcal{N}(i)$ as the neighbors of i including itself. It is assumed that the graph G is strongly connected with self loops i.e., there is a path from every agent to every other agent. The adjacency matrix of the graph G is referred as a mixing matrix W where w_{ij} is the weight associated with the edge $\{i, j\}$. Note that, weight 0 indicates the absence of a direct edge between the agents. We assume that the mixing matrix is doubly-stochastic and symmetric, similar to all previous works in decentralized learning. For example, in a undirected ring topology, $w_{ij} = \frac{1}{3}$ if $j \in \{i-1, i, i+1\}$. Further, the initial models and all the hyperparameters are synchronized in the beginning of the training. Algorithm. 1 describes the flow of D-PSGD with momentum. The convergence of the Algorithm. 1 assumes the data-distribution across the agents to be Independent and Identically Distributed (IID).

3 Neighborhood Gradient Clustering

We propose the *Neighborhood Gradient Clustering (NGC)* algorithm and a compressed version of *NGC* which improve the performance of decentralized learning over non-IID data distribution. *NGC* utilizes the concepts of self-gradient and cross-gradient Esfandiari et al. (2021). The following are the definitions of self-gradient and cross-gradient.

Self-Gradient: For an agent i with the local dataset D_i and model parameters x_i, the self-gradient is the gradient of the loss function f_i with respect to the model parameters x_i, evaluated on mini-batch d_i sampled from dataset D_i.

$$g_{ii}^t = \nabla_x f_i(d_i^t; x_i^t) \tag{2}$$

Cross-Gradient: For an agent i with model parameters x_i connected to neighbor j that has local dataset D_j, the cross-gradient is the gradient of the loss function f_j with respect to the model parameters x_i, evaluated on mini-batch d_j sampled from dataset D_j.

$$g_{ij}^t = \nabla_x f_j(d_j^t; x_i^t) \tag{3}$$

Note that the cross-gradient g_{ij} is computed on agent j using its local data after receiving the model parameters x_i from its neighbouring agent i and is then communicated to agent i.

3.1 The *NGC* algorithm

The flow of the Neighborhood Gradient Clustering (*NGC*) is shown in Algorithm. 2 and the form of the algorithm is similar to D-PSGD Lian et al. (2017) presented in Algorithm. 1.

The main contribution of the proposed *NGC* algorithm is the local gradient manipulation step (line 10 in Algorithm. 2). In the tth iteration of *NGC*, each agent i calculates its self-gradient g_{ii}. Then, agent i's model parameters are transmitted to all other agents (j) in its neighborhood, and the respective cross-gradients are calculated by the neighbors and transmitted back to agent i. At every iteration after the communication rounds, each agent i has access to self-gradients (g_{ii}) and two sets of cross-gradients: 1) *Model-variant cross-gradients*: The derivatives that are computed locally using its local data on the neighbors' model parameters (g_{ji}). 2) *Data-variant cross-gradients*: The derivatives (received through communication) of its model parameters on the neighbors' dataset (g_{ij}). Note that each agent i computes and transmits cross-gradients (g_{ji}) that act as model-variant cross-gradients for i and as data-variant cross-gradients for j. We then cluster the gradients into two groups namely: a) *Model-variant cluster* $\{g_{ji} \forall j \in \mathcal{N}(i)\}$ that includes self-gradients and model-variant cross-gradients, and b) *Data-variant cluster* $\{g_{ij} \forall j \in \mathcal{N}(i)\}$ that includes self-gradients and data-variant cross-gradients. The local gradients at each agent is replaced with the weighted average of the above defined cluster means as shown in Equation. 4, which assumes uniform mixing matrix ($w_{ij} = 1/m; m = |\mathcal{N}(i)|$). The mean of model-variant cluster is weighted by $(1-\alpha)$ and the mean of data-variant cluster is weighed by α where $\alpha \in [0, 1]$ is a hyper-parameter referred as *NGC* mixing weight.

$$\tilde{g}_i^t = (1-\alpha) * \underbrace{\left[\frac{1}{m} \sum_{j \in \mathcal{N}(i)} g_{ji}^t\right]}_{\textbf{(a) Model-variant cluster mean}} \quad + \quad \alpha * \underbrace{\left[\frac{1}{m} \sum_{j \in \mathcal{N}(i)} g_{ij}^t\right]}_{\textbf{(b) Data-variant cluster mean}} \tag{4}$$

Algorithm 2 Neighborhood Gradient Clustering (*NGC*)

Input: Each agent $i \in [1, n]$ initializes model weights $x_i^{(0)}$, learning rate γ, averaging rate η, mixing matrix $W = [w_{ij}]_{i,j\in[1,n]}$, *NGC* mixing weight α, and I_{ij} are elements of $n \times n$ identity matrix, $\mathcal{N}(i)$ represents neighbors of i including itself.

Each agent simultaneously implements the TRAIN( ) procedure
procedure TRAIN( )
  for t=$0, 1, \ldots, T-1$ **do**
    $d_i^t \sim D_i$ // sample data from training dataset.
    $g_{ii}^t = \nabla_x f_i(d_i^t; x_i^t)$ // compute the local self-gradients
    SENDRECEIVE(x_i^t) // shares model parameters with neighbors $\mathcal{N}(i)$.
    for each neighbor $j \in \{\mathcal{N}(i) - i\}$ **do**
      $g_{ji}^t = \nabla_x f_i(d_i^t; x_j^t)$ // compute neighbors' cross-gradients
      if $\alpha \neq 0$ **do**
        SENDRECEIVE(g_{ji}^t) // share cross-gradients between i and j
      end
    end
    $\widetilde{g}_i^t = (1-\alpha) * \sum_{j\in\mathcal{N}(i)} w_{ij} * g_{ji}^t + \alpha * \sum_{j\in\mathcal{N}(i)} w_{ij} * g_{ij}^t$ // modify local gradients
    $v_i^t = \beta v_i^{(t-1)} - \eta \widetilde{g}_i^t$ // momentum step
    $\widetilde{x}_i^t = x_i^t + v_i^t$ // update the model
    $x_i^{(t+1)} = \widetilde{x}_i^t + \gamma \sum_{j\in\mathcal{N}(i)} (w_{ij} - I_{ij}) * x_j^t$ // gossip averaging step
  end
return

The motivation for this modification is to reduce the variation of the computed local gradients across the agents. In IID settings, the local gradients should statistically resemble the cross-gradients and hence simple gossip averaging is sufficient to reach convergence. However, in the non-IID case, the local gradients across the agents are significantly different due to the variation in datasets and hence the model parameters on which the gradients are computed. The proposed algorithm reduces this variation in the local-gradients as it is equivalent to adding two bias terms ϵ and ω with weights $(1-\alpha)$ and α respectively as shown in Equation. 5.

$$
\begin{aligned}
\widetilde{g}_i^t &= (1-\alpha)*\Big[g_{ii}^t + \frac{1}{m}\sum_{j\in\mathcal{N}(i)}(g_{ji}^t - g_{ii}^t)\Big] + \alpha*\Big[g_{ii}^t + \frac{1}{m}\sum_{j\in\mathcal{N}(i)}\frac{1}{m}(g_{ij}^t - g_{ii}^t)\Big] \\
&= g_{ii}^t + (1-\alpha) * \underbrace{\Big[\frac{1}{m}\sum_{j\in\mathcal{N}(i)}(g_{ji}^t - g_{ii}^t)\Big]}_{\text{model variance bias } \epsilon_i^t} + \alpha * \underbrace{\Big[\frac{1}{m}\sum_{j\in\mathcal{N}(i)}\frac{1}{m}(g_{ij}^t - g_{ii}^t)\Big]}_{\text{data variance bias } \omega_i^t} \qquad (5) \\
\epsilon_i^t &= \frac{1}{m} * \sum_{j\in\mathcal{N}(i)} \big(\nabla_x f(d_i^t; x_j^t) - \nabla_x f(d_i^t; x_i^t)\big) \\
\omega_i^t &= \frac{1}{m} * \sum_{j\in\mathcal{N}(i)} \big(\nabla_x f(d_j^t; x_i^t) - \nabla_x f(d_i^t; x_i^t)\big) \qquad \text{(note that } f = f_i = f_j)
\end{aligned}
$$

The bias term ϵ compensates for the difference in a neighborhood's self-gradients caused due to variation in the model parameters across the neighbors. Whereas, the bias term ω compensates for the difference in a neighborhood's self-gradients caused due to variation in the data distribution across the neighbors. We hypothesis and show through our experiments that addition of these bias terms to the local gradients improves the performance of decentralized learning over non-IID data by accelerating the global convergence. Note that if we set $\alpha = 0$ in the *NGC* algorithm then it does not require an additional communication round (no communication overhead compared to D-PSGD).

3.2 THE COMPRESSED *NGC* ALGORITHM

The *NGC* algorithm at every iteration involves two-steps of communication with the neighbors: 1) communicate the model parameters, and 2) communicate the cross-gradients. This communi-

cation overhead can be a bottleneck in a resource-constrained environment. Hence we propose a compressed version of *NGC* using Error Feedback SGD (EF-SGD) Karimireddy et al. (2019) to compress gradients. We compress the error-compensated self-gradients and cross-gradients from 32 bits (full-precision) to 1 bit by using scaled signed gradients. The error between the compressed and non-compressed gradient of the current iteration (e_{ji}^t in the algorithm) is added as feedback to the gradients in the next iteration before compression. The pseudo code for *CompNGC* is shown in Algorithm. 3.

Algorithm 3 Compressed Neighborhood Gradient Clustering (*CompNGC*)

Input: Each agent $i \in [1, n]$ initializes model weights $x_i^{(0)}$, learning rate γ, averaging rate η, dimension of the gradient d, mixing matrix $W = [w_{ij}]_{i,j\in[1,n]}$, *NGC* mixing weight α, and I_{ij} are elements of $n \times n$ identity matrix.

Each agent simultaneously implements the TRAIN( ) procedure
1. **procedure** TRAIN( )
2. **for** t=$0, 1, \ldots, T-1$ **do**
3. $d_i^t \sim D_i$ // sample data from training dataset.
4. $g_{ii}^t = \nabla_x f_i(d_i^t; x_i^t)$ // compute the local self-gradients
5. $p_{ii}^t = g_{ii}^t + e_{ii}^t$ // error compensation for self-gradients
6. $\delta_{ii}^t = (||p_{ii}^t||_1/d)sgn(p_{ii})$ // compress the compensated self-gradients
7. $e_{ii}^t = p_{ii}^t - \delta_{ii}^t$ // update the error variable
8. SENDRECEIVE(x_i^t) // share model parameters with neighbors $N(i)$.
9. **for** each neighbor $j \in \{N(i) - i\}$ **do**
10. $g_{ji}^t = \nabla_x f_i(d_i^t; x_j^t)$ // compute neighbors' cross-gradients
11. $p_{ji}^t = g_{ji}^t + e_{ji}^t$ // error compensation for cross-gradients
12. $\delta_{ji}^t = (||p_{ji}^t||_1/d)sgn(p_{ji})$ // compress the compensated cross-gradients
13. $e_{ji}^t = p_{ji}^t - \delta_{ji}^t$ // update the error variable
14. **if** $\alpha \neq 0$ **do**
15. SENDRECEIVE(δ_{ji}^t) // share compressed cross-gradients between i and j.
16. **end**
17. **end**
18. $\widetilde{g}_i^t = (1-\alpha) * \sum_{j\in\mathcal{N}(i)} w_{ij} * \delta_{ji}^t + \alpha * \sum_{j\in\mathcal{N}(i)} w_{ij} * \delta_{ij}^t$ // modify local gradients
19. $v_i^t = \beta v_i^{(t-1)} - \eta \widetilde{g}_i^t$ // momentum step
20. $\widetilde{x}_i^t = x_i^t + v_i^t$ // update the model
21. $x_i^{(t+1)} = \widetilde{x}_i^t + \gamma \sum_{j\in\mathcal{N}(i)} (w_{ij} - I_{ij}) * x_j^t$ // gossip averaging step
22. **end**
23. **return**

4 EXPERIMENTS

In this section, we analyze the performance of the proposed *NGC* and *CompNGC* techniques and compare them with the baseline D-PSGD algorithm Lian et al. (2017) and state-of-the-art CGA and CompCGA methods Esfandiari et al. (2021).

4.1 EXPERIMENTAL SETUP

We present the analysis of the proposed method on CIFAR10 Krizhevsky et al. (2014) dataset with 5, 10 and 20 agents. We consider an extreme case of non-IID distribution where no two neighboring agents have same class. This is referred as complete label-wise skew or 100% label-wise non-IIDness Hsieh et al. (2020). In particular for a 10-class dataset such as CIFAR10 - each agent in a 5 nodes system has data from 2 distinct classes, each agent in a 10 nodes system has data from an unique class. For a 20 node system two agents that are maximally apart share the samples belonging to a class. We run our experiments on 5-layer CNN, modified VGG-11 and ResNet architectures (refer Appendix A.1 for details of the model architectures). We replace ReLU+BatchNorm layers with EvoNorm-S0 Liu et al. (2020) as it was shown to be better suited for decentralized learning

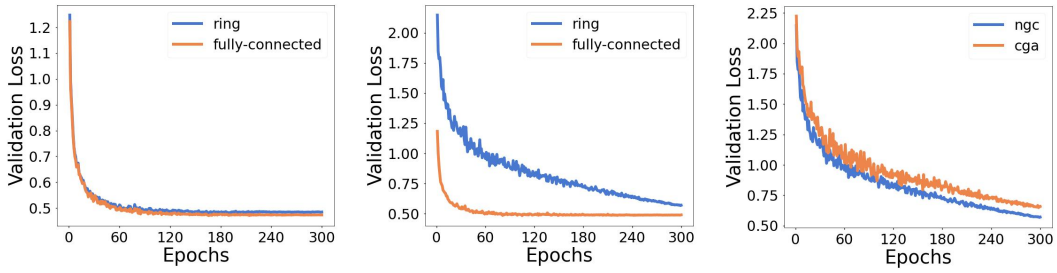
over non-IID distributions Lin et al. (2021). Initial learning-rate is tuned either to 0.1 or 0.01 based on the model architecture and optimizer. We decay the learning-rate by 10x after 50% and 75% of the training, unless mentioned otherwise. For all the experiments, we use a mini-batch size of 32 per agent, and the stopping criteria is a fixed number of epochs. We have used Nesterov momentum of 0.9 for all CGA and *NGC* experiments where as D-PSGD and *NGC* with $\alpha = 0$ has no momentum.

For the decentralized setup, we use undirected ring and undirected torus graph topologies with uniform mixing matrix. The undirected ring topology for any graph-size has 3 peers per agent including itself and each edge has a weight of $\frac{1}{3}$. The undirected torus topology with 10 agents has 4 peers per agent including itself and each edge has a weight of $\frac{1}{4}$. The undirected torus topology 20 agents has 5 peers per agent including itself and each edge has a weight of $\frac{1}{5}$. We show the performance of the proposed technique on various graph types and sizes. We report the test accuracy of the averaged model (consensus model) and the communication cost per agent for training in terms of data transferred (in Gigabytes) assuming full precision operations to be in 32 bits. The hyper-parameters for all the experiments are presented in Appendix A.2. Our PyTorch code is attached in the supplementary material.

4.2 Experimental Results

We analyze the efficiency of the proposed technique on a 5-layer CNN network and with EvoNorm-S0 as activation-normalization layer. We observe that *NGC* outperforms CGA by 1.88% on an average, for all scenarios with different number of agents and graph topologies while incurring same communication cost (shown in Table. 1). *CompNGC* performs comparable to *NGC* while reducing the communication cost by $32\times$. Finally, we validate the proposed technique on VGG-11 and ResNet architectures to assess the scalability of the method on models that are deep, parameter heavy and have residual connections. The results are summarized in Table. 2. We observe similar trends where *NGC* outperforms CGA by 2.13% and *CompNGC* outperforms CompCGA by 3% on an average. *NGC* with $\alpha = 0$ is a special case as it has same communication cost as D-PSGD algorithm but performs $\sim 20\%$ better in terms of test accuracy.

We show the convergence characteristics of the proposed algorithm over IID and Non-IID data distributions in Figure. 1a, and 1b respectively. For Non-IID distribution, we observe that there is a slight difference in convergence (as expected) with slower rate for sparser topology (ring graph) compared to denser counterpart (fully connected graph). Figure. 1c shows the comparison of the convergence characteristics of the *NGC* algorithm with the current state-of-the-art CGA algorithm. We observe that *NGC* has lower validation loss than CGA for same decentralized setup. Analysis for 10 agents is presented in Appendix A.3. The change in average validation accuracy with *NGC* mixing weight α is shown in Figure. 2. We observe that α close to 1 has the best performance as the model variance bias is taken care in the gossip averaging step. We also plot the model variance and data variance bias terms for both *NGC* and CGA techniques as shown in Figure. 3a, and 3b respectively. We observe that both the model variance and the data variance bias for *NGC* are significantly lower than CGA. This is because CGA gives more importance to self-gradients as it updates in the descent direction that is close to self-gradients and is positively correlated to data-



(a) *NGC* method on IID data. (b) *NGC* method on Non-IID data. (c) Different methods on Non-IID for ring topology.

Figure 1: Average Validation loss during training of 5 agents on CIFAR10 dataset with a 5 layer CNN network.

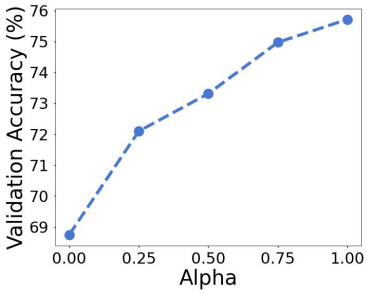
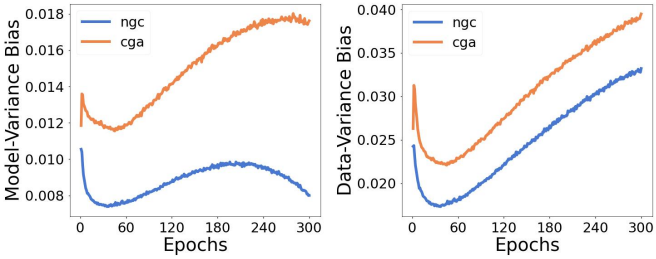


Figure 2: *NGC* mixing weight α variation for 10 agents trained on ring topology with 5 layer CNN network.



(a) Average L1 norm of ϵ.

(b) Average L1 norm of ω.

Figure 3: Average L1 norm of model variance bias and data variance bias for 5 agents trained on ring topology with 5 layer CNN architecture.

Table 1: Average test accuracy and communication cost comparison for CIFAR10 with non-IID data distribution using 5-layer CNN model architecture.

Method	Agents (n)	Ring		Torus	
		Test Acc. %	Comm. cost (GB)	Test Acc. %	Comm. cost (GB)
D-PSGD	5	49.76	17.75	-	-
	10	39.06	8.92	59.36	13.38
	20	42.52	4.50	53.56	9.01
NGC (ours) ($\alpha = 0$)	5	**82.28**	17.75	-	-
	10	**66.36**	8.92	**74.17**	13.38
	20	**47.33**	4.50	**65.85**	9.01
CGA	5	82.12	35.48	-	-
	10	73.04	17.81	75.46	26.72
	20	70.05	8.98	73.06	17.95
NGC (ours)	5	**82.62**	35.48	-	-
	10	**75.27**	17.81	**78.14**	26.72
	20	**72.50**	8.98	**75.06**	17.95
CompCGA	5	81.81	18.31	-	-
	10	70.59	9.20	75.48	13.79
	20	67.72	4.64	71.22	9.28
CompNGC (ours)	5	**82.71**	18.31	-	-
	10	**74.01**	9.20	**77.96**	13.79
	20	**72.08**	4.64	**73.35**	9.28

variant cross-gradients. In contrast, *NGC* accounts for the biases directly and gives equal importance to self-gradients and data-variant cross-gradients, thereby achieving superior performance.

The proposed *NGC* algorithm is superior in terms of memory and compute efficiency (see Table. 3), while having equal communication cost as compared to *CGA*. Since *NGC* involves weighted averaging, we do not need any additional buffer to store the cross-gradients. Weighted cross-gradients can be added to the the self-gradient buffer. CGA stores all the cross-gradients in a matrix form for quadratic programming projection of local gradient. Therefore, *NGC* has no memory overhead compared to the baseline D-PSGD algorithm, while CGA requires additional memory equivalent to the number of neighbors times model size. Moreover, the quadratic programming projection step Goldfarb & Idnani (1983) in CGA is much more expensive in terms of compute and latency as compared to weighted averaging step of cross-gradients in *NGC*. Our experiments clearly show *NGC* is superior to CGA in terms of test accuracy, memory efficiency, compute efficiency and latency.

Table 2: Average test accuracy and communication cost comparison for CIFAR10 with non-IID data using various architectures over an undirected ring topology.

Method	Agents (n)	VGG-11 Test Acc. %	VGG-11 Comm. cost (GB)	ResNet-20 Test Acc. %	ResNet-20 Comm. cost (GB)
D-PSGD	5	69.37	270.64	82.82	127.19
	10	40.33	135.86	36.71	63.84
	20	35.64	68.47	10 (diverges)	32.18
NGC (ours) ($\alpha = 0$)	5	**79.59**	270.64	**86.61**	127.19
	10	**60.08**	135.86	**62.79**	63.84
	20	**52.33**	68.47	**50.99**	32.18
CGA	5	84.67	541.05	86.97	254.27
	10	**79.96**	271.50	79.60	127.59
	20	79.20	136.72	75.83	64.25
NGC (ours)	5	**85.59**	541.05	**88.53**	254.27
	10	79.53	271.50	**84.41**	127.59
	20	**80.14**	136.72	**80.81**	64.25
CompCGA	5	83.75	279.09	86.65	131.16
	10	74.14	140.10	74.27	65.84
	20	70.57	70.61	66.00	33.18
CompNGC (ours)	5	**84.31**	279.09	**87.80**	131.16
	10	**76.90**	140.10	**78.49**	65.84
	20	**73.93**	70.61	**72.13**	33.18

Table 3: Comparison of communication, memory and compute overheads required per mini-batch compared to baseline (D-PSGD). m_s: model size for the local agent, N_i: number of neighbors, b: floating point precision of arithmetic operations (e.g. 32), QP: compute for quadratic programming projection step, FP: compute for Forward Pass. (see Appendix A.4 for more details)

Method	Comm.	Memory	Compute
CGA	$\mathcal{O}(m_s N_i)$	$\mathcal{O}(N_i m_s)$	$\mathcal{O}(3N_i FP + QP)$
CompCGA	$\mathcal{O}(\frac{m_s N_i}{b})$	$\mathcal{O}((1+\frac{1}{b})N_i m_s)$	$\mathcal{O}(3N_i FP + QP)$
NGC $\alpha = 0$ (ours)	0	0	$\mathcal{O}(3N_i FP + m_s N_i)$
NGC (ours)	$\mathcal{O}(m_s N_i)$	0	$\mathcal{O}(3N_i FP + m_s N_i)$
CompNGC (ours)	$\mathcal{O}(\frac{m_s N_i}{b})$	$\mathcal{O}(N_i m_s)$	$\mathcal{O}(3N_i FP + m_s N_i)$

5 Conclusion

Enabling decentralized training over non-IID data is key for ML applications to efficiently leverage the humongous amounts of user-generated private data. In this paper, we propose the *Neighborhood Gradient Clustering* (*NGC*) algorithm that improves decentralized learning over non-IID data distributions. Further, we present a compressed version of our algorithm (*CompNGC*) to reduce the communication overhead associated with *NGC*. We validate the performance of the proposed techniques (*NGC* and *CompNGC*) over different model architectures, graph sizes and topologies. Finally, we compare the proposed algorithms with the current state-of-the-art decentralized learning algorithm over non-IID data and show superior performance of our algorithm with significantly less compute and memory requirements.

REFERENCES

Alekh Agarwal and John C Duchi. Distributed delayed stochastic optimization. *Advances in neural information processing systems*, 24, 2011.

Sai Aparna Aketi, Amandeep Singh, and Jan Rabaey. Sparse-push: Communication-& energy-efficient decentralized distributed learning over directed & time-varying graphs with non-iid datasets. *arXiv preprint arXiv:2102.05715*, 2021.

Sai Aparna Aketi, Sangamesh Kodge, and Kaushik Roy. Low precision decentralized distributed training over iid and non-iid data. *Neural Networks*, 2022. ISSN 0893-6080. doi: https://doi.org/10.1016/j.neunet.2022.08.032.

Mahmoud Assran, Nicolas Loizou, Nicolas Ballas, and Mike Rabbat. Stochastic gradient push for distributed deep learning. In *International Conference on Machine Learning*, pp. 344–353. PMLR, 2019.

Aditya Balu, Zhanhong Jiang, Sin Yong Tan, Chinmay Hedge, Young M Lee, and Soumik Sarkar. Decentralized deep learning using momentum-accelerated consensus. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 3675–3679. IEEE, 2021.

Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pp. 177–186. Springer, 2010.

Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, et al. Large scale distributed deep networks. *Advances in neural information processing systems*, 25, 2012.

Yasaman Esfandiari, Sin Yong Tan, Zhanhong Jiang, Aditya Balu, Ethan Herron, Chinmay Hegde, and Soumik Sarkar. Cross-gradient aggregation for decentralized learning from non-iid data. In *International Conference on Machine Learning*, pp. 3036–3046. PMLR, 2021.

Donald Goldfarb and Ashok Idnani. A numerically stable dual method for solving strictly convex quadratic programs. *Mathematical programming*, 27(1):1–33, 1983.

Arya Ketabchi Haghighat, Varsha Ravichandra-Mouli, Pranamesh Chakraborty, Yasaman Esfandiari, Saeed Arabi, and Anuj Sharma. Applications of deep learning in intelligent transportation systems. *Journal of Big Data Analytics in Transportation*, 2(2):115–145, 2020.

Kevin Hsieh, Amar Phanishayee, Onur Mutlu, and Phillip Gibbons. The non-IID data quagmire of decentralized machine learning. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 4387–4398. PMLR, 13–18 Jul 2020.

Sai Praneeth Karimireddy, Quentin Rebjock, Sebastian Stich, and Martin Jaggi. Error feedback fixes signsgd and other gradient compression schemes. In *International Conference on Machine Learning*, pp. 3252–3261. PMLR, 2019.

Anastasia Koloskova, Tao Lin, Sebastian U Stich, and Martin Jaggi. Decentralized deep learning with arbitrary communication compression. *arXiv preprint arXiv:1907.09356*, 2019.

Anastasia Koloskova, Nicolas Loizou, Sadra Boreiri, Martin Jaggi, and Sebastian Stich. A unified theory of decentralized sgd with changing topology and local updates. In *International Conference on Machine Learning*, pp. 5381–5393. PMLR, 2020.

Jakub Konečnỳ, H Brendan McMahan, Daniel Ramage, and Peter Richtárik. Federated optimization: Distributed machine learning for on-device intelligence. 2016.

Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar (canadian institute for advanced research). *http://www.cs.toronto.edu/ kriz/cifar.html*, 2014.

Xiangru Lian, Ce Zhang, Huan Zhang, Cho-Jui Hsieh, Wei Zhang, and Ji Liu. Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent. *Advances in Neural Information Processing Systems*, 30, 2017.

Tao Lin, Sai Praneeth Karimireddy, Sebastian Stich, and Martin Jaggi. Quasi-global momentum: Accelerating decentralized deep learning on heterogeneous data. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 6654–6665. PMLR, 18–24 Jul 2021.

Hanxiao Liu, Andy Brock, Karen Simonyan, and Quoc Le. Evolving normalization-activation layers. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 13539–13550. Curran Associates, Inc., 2020.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Hanlin Tang, Xiangru Lian, Ming Yan, Ce Zhang, and Ji Liu. d^2: Decentralized training over decentralized data. In *International Conference on Machine Learning*, pp. 4848–4856. PMLR, 2018.

Hanlin Tang, Xiangru Lian, Shuang Qiu, Lei Yuan, Ce Zhang, Tong Zhang, and Ji Liu. Deepsqueeze: Decentralization meets error-compensated compression. *arXiv preprint arXiv:1907.07346*, 2019.

Lin Xiao and Stephen Boyd. Fast linear iterations for distributed averaging. *Systems & Control Letters*, 53(1):65–78, 2004.

A APPENDIX

A.1 NETWORK ARCHITECTURE

5 layer CNN: The 5 layer CNN consists of 4 convolutional with EvoNorm-S0 Liu et al. (2020) as activation-normalization layer, 3 max pooling layer and one linear layer. In particular, it has 2 convolutional layers with 32 filters, a max pooling layer, then 2 more convolutional layers with 64 filters each followed by another max pooling layer and a dense layer with 512 units. It has a total of $76K$ trainable parameters.

Modified VGG-11: We modify the standard VGG-11 Simonyan & Zisserman (2014) architecture by reducing the number of filters in each convolutional layer by $4\times$ and use only one dense layer with 128 units. Each convolutional layer is followed by EvoNorm-S0 as activation-normalization layer and it has $0.58M$ trainable parameters.

Modified ResNet-20: For ResNet-20, we use the standard architecture with $0.27M$ trainable parameters except that BatchNorm+ReLU layer is replaced by EvoNorm-S0.

A.2 HYPER-PARAMETERS

Hyper-parameters for Table 1: All the experiments in Table 1 have stopping criteria set to 100. We decay the learning rate by $10\times$ in a multiple steps at 50th and 75th epoch. Table 4 presents the α, β, η, and γ corresponding to the ngc mixing weight, momentum, learning-rate and gossip averaging rate for different algorithms is presented in Table 4.

Table 4: Hyper-parameters used for CIFAR10 with non-IID data distribution using 5-layer CNN model architecture.

Method	Agents (n)	Ring (α, β, η, γ)	Torus (α, β, η, γ)
D-PSGD	5	$(-, 0.0, 0.1, 1.0)$	$-$
	10	$(-, 0.0, 0.1, 1.0)$	$(-, 0.0, 0.1, 1.0)$
	20	$(-, 0.0, 0.1, 1.0)$	$(-, 0.0, 0.1, 1.0)$
NGC (ours) ($\alpha = 0$)	5	$(0.0, 0.0, 0.1, 1.0)$	$-$
	10	$(0.0, 0.0, 0.1, 1.0)$	$(0.0, 0.0, 0.1, 1.0)$
	20	$(0.0, 0.0, 0.1, 1.0)$	$(0.0, 0.0, 0.1, 1.0)$
CGA	5	$(-, 0.9, 0.01, 0.1)$	$-$
	10	$(-, 0.9, 0.01, 0.5)$	$(-, 0.9, 0.01, 0.1)$
	20	$(-, 0.9, 0.01, 0.5)$	$(-, 0.9, 0.01, 0.1)$
NGC (ours)	5	$(1.0, 0.9, 0.01, 0.1)$	$-$
	10	$(1.0, 0.9, 0.01, 0.5)$	$(1.0, 0.9, 0.01, 0.1)$
	20	$(1.0, 0.9, 0.01, 0.5)$	$(1.0, 0.9, 0.01, 0.1)$
CompCGA	5	$(-, 0.9, 0.01, 0.1)$	$-$
	10	$(-, 0.9, 0.01, 0.5)$	$(-, 0.9, 0.01, 0.1)$
	20	$(-, 0.9, 0.01, 0.5)$	$(-, 0.9, 0.01, 0.1)$
CompNGC (ours)	5	$(1.0, 0.9, 0.01, 0.1)$	$-$
	10	$(1.0, 0.9, 0.01, 0.5)$	$(1.0, 0.9, 0.01, 0.1)$
	20	$(1.0, 0.9, 0.01, 0.5)$	$(1.0, 0.9, 0.01, 0.1)$

Hyper-parameters for Table 2: All the experiments in Table 2 have stopping criteria set to 200. We decay the learning rate by $10\times$ in a multiple steps at 100th and 150th epoch. Table 4 presents the α, β, η, and γ corresponding to the ngc mixing weight, momentum, learning-rate and gossip averaging rate for different algorithms is presented in Table 5.

Hyper-parameters for Figures: The simulations for all the figures are run for 300 epochs. We scale the learning-rate by a factor or 0.981 after each epoch to obtain a smoother curve. All the experiment in Figure 1, 2, 3, 4 and 5 use 5 layer CNN network. Experiments in Figure 1, and 3 use 5 agents while the experiments in Figure 2, 4 and 5 use 10 agents. The hyper-parameters for the simulations for all the plots are mentioned in Table 6

Table 5: Hyper-parameters used for CIFAR10 with non-IID data distribution using ResNet and VGG11 model architecture

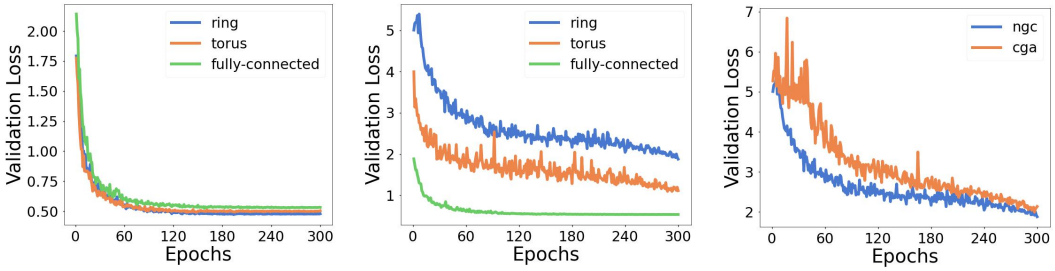
Method	Agents (n)	VGG11 (α, β, η, γ)	ResNet (α, β, η, γ)
D-PSGD	5	$(,0.0, 0.01, 1.0)$	$(-, 0.0, 0.1, 1.0)$
	10	$(,0.0, 0.01, 1.0)$	$(-, 0.0, 0.1, 1.0)$
	20	$(,0.0, 0.01, 1.0)$	$(-, 0.0, 0.1, 1.0)$
NGC (ours) ($\alpha = 0$)	5	$(0.0, 0.0, 0.01, 1.0)$	$(0.0, 0.0, 0.1, 1.0)$
	10	$(0.0, 0.0, 0.01, 1.0)$	$(0.0, 0.0, 0.1, 1.0)$
	20	$(0.0, 0.0, 0.01, 1.0)$	$(0.0, 0.0, 0.1, 1.0)$
CGA	5	$(,0.9, 0.1, 0.5)$	$(-, 0.9, 0.1, 1.0)$
	10	$(,0.9, 0.1, 0.5)$	$(-, 0.9, 0.1, 1.0)$
	20	$(,0.9, 0.1, 0.5)$	$(-, 0.9, 0.1, 1.0)$
NGC (ours)	5	$(1.0, 0.9, 0.1, 0.5)$	$(1.0, 0.9, 0.1, 1.0)$
	10	$(1.0, 0.9, 0.1, 0.5)$	$(1.0, 0.9, 0.1, 1.0)$
	20	$(1.0, 0.9, 0.1, 0.5)$	$(1.0, 0.9, 0.1, 1.0)$
CompCGA	5	$(-, 0.9, 0.01, 0.1)$	$(-, 0.9, 0.01, 0.1)$
	10	$(-, 0.9, 0.01, 0.1)$	$(-, 0.9, 0.01, 0.1)$
	20	$(-, 0.9, 0.01, 0.1)$	$(-, 0.9, 0.01, 0.1)$
CompNGC (ours)	5	$(1.0, 0.9, 0.01, 0.1)$	$(1.0, 0.9, 0.01, 0.1)$
	10	$(1.0, 0.9, 0.01, 0.1)$	$(1.0, 0.9, 0.01, 0.1)$
	20	$(1.0, 0.9, 0.01, 0.1)$	$(1.0, 0.9, 0.01, 0.1)$

Table 6: Hyper-parameters used for Figures 1, 2, 3, 4 and 5 on CIFAR10 using 5 layer CNN.

Figure	α	β	η	γ
1a (skew=0) NGC (ours)	1.0	0.9	0.01	1.0
1b (skew=1) NGC (ours)	1.0	0.9	0.01	0.1
1c and 3 (skew=1) ring	1.0	0.9	0.01	0.1
2 (skew=1) NGC (ours) ring	0.0	0.9	0.01	1.0
	0.25			1.0
	0.5			0.5
	0.75			0.5
	1.0			0.25
4a (skew=0) NGC (ours)	1.0	0.9	0.1	1.0
4b (skew=1) NGC (ours)	1.0	0.9	0.1	0.5
4c (skew=1) ring	1.0	0.9	0.1	0.5

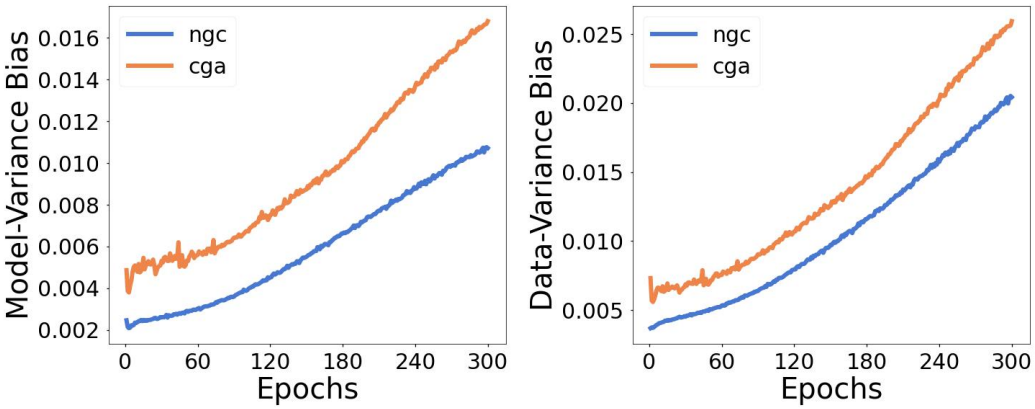
A.3 Analysis for 10 Agents

We show the convergence characteristics of the proposed *NGC* algorithm over IID and Non-IID data distributions in Figure. 4a, and 4b respectively. For Non-IID distribution, we observe that there is a slight difference in convergence (as expected) with slower rate for sparser topology (ring graph) compared to denser counterpart (fully connected graph). Figure. 4c shows the comparison of the convergence characteristics of the *NGC* technique with the current state-of-the-art CGA algorithm. We observe that *NGC* has lower validation loss than CGA for same decentralized setup. We also plot the model variance and data variance bias terms for both *NGC* and CGA techniques as shown in Figure. 5a, and 5b respectively. We observe that both model variance and data variance bias for *NGC* are significantly lower than CGA.



(a) *NGC* method on IID data. (b) *NGC* method on Non-IID data. (c) Different methods on Non-IID for ring topology.

Figure 4: Average Validation loss during training of 10 agents on CIFAR10 dataset with a 5 layer CNN network.



(a) Average L1 norm of ϵ. (b) Average L1 norm of ω.

Figure 5: Average L1 norm of model variance bias and data variance bias for 10 agents trained on ring topology with 5 layer CNN architecture.

A.4 Resource Comparison

The communication cost, memory overhead and compute overhead for various decentralized algorithms are shown in Table. 3. The D-PSGD algorithm requires each agent to communicate model parameters of size m_s with all the N_i neighbors for the gossip averaging step and hence has a communication cost of $\mathcal{O}(m_s N_i)$. In case of *NGC* and CGA, there is an additional communication round for sharing data-variant cross gradients apart from sharing model parameters for gossip averaging step. So, both these techniques incur a communication cost of $\mathcal{O}(2m_s N_i)$ and therefor an overhead of $\mathcal{O}(m_s N_i)$ compared to D-PSGD. *CompNGC* compresses the additional round of communication involved with *NGC* from b bits to 1 bit. This reduces the communication overhead from $\mathcal{O}(m_s N_i)$ to $\mathcal{O}(\frac{m_s N_i}{b})$.

CGA algorithm stores all the received data-variant cross-gradients in the form of a matrix for quadratic projection step. Hence, CGA has a memory overhead of $\mathcal{O}(m_s N_i)$ compared to D-PSGD. *NGC* does not require any additional memory as it averages the received data-variant cross-gradients into self-gradient buffer. The compressed version of *NGC* requires an additional memory of $\mathcal{O}(m_s N_i)$ to store the error variables e_{ji} (refer Algorith. 3). CompCGA also needs to store error variables along with the projection matrix of compressed gradients. Therefore, CompCGA has a memory overhead of $\mathcal{O}(m_s N_i + \frac{m_s N_i}{b})$. Note that memory overhead depends on the type of graph topology and model architecture but not on the size of the graph. The memory overhead for different model architectures trained on undirected ring topology is shown in Table. 7

CGA algorithm involves quadratic programming projection step Goldfarb & Idnani (1983) to update the local gradients. Quadratic programming solver (quadprog) uses Goldfarb/Idnani dual algorithm.

Table 7: Memory overheads for various methods trained on different model architectures over undirected ring topology with 2 neighbors per agent.

Architecture	CGA (MB)	*NGC* (MB)	CompCGA (MB)	*CompNGC* (MB)
5 layer CNN	0.58	0	0.58	0.60
VGG-11	4.42	0	4.42	4.56
ResNet20	2.28	0	2.28	2.15

CGA uses quadratic programming to solve the following (Equation 6 -see Equation 5a in Esfandiari et al. (2021)) optimization problem in an iterative manner:

$$\begin{aligned} \text{min}_u \quad & \frac{1}{2}u^T G G^T u + g^T G^T u \\ \text{s.t.} \quad & u \geq 0 \end{aligned} \tag{6}$$

where, G is the matrix containing cross-gradients, g is the self-gradient and the optimal gradient direction g^* in terms of the optimal solution of the above equation u^* is $g^* = G^T u^* + g$. The above optimization takes multiple iterations which results in compute and time complexity to be of polynomial(degree$\geq$ 2) order. In contrast, NGC involves simple averaging step that requires $O(m_s N_i)$ addition operations.