Neighborhood Gradient Clustering: An Efficient Decentralized Learning Method for Non-IID Data

Sai Aparna Aketi [*1] Sangamesh Kodge [*1] Kaushik Roy [1]

Abstract

Decentralized learning algorithms enable the training of deep learning models over large distributed datasets, without the need for a central server. In practical scenarios, the distributed datasets can have significantly different data distributions across the agents. In this paper, we propose *Neighborhood Gradient Clustering (NGC)*, a novel decentralized learning algorithm to improve decentralized learning over non-IID data. Specifically, the proposed method replaces the local gradients of the model with the weighted mean of self-gradients, model-variant cross-gradients, and data-variant cross-gradients. Model-variant cross-gradients are derivatives of the received neighbors' model parameters with respect to the local dataset - computed locally. Data-variant cross-gradients are derivatives of the local model with respect to its neighbors' datasets - received through communication. We demonstrate the efficiency of *NGC* over non-IID data sampled from various vision datasets. Our experiments demonstrate that the proposed method either remains competitive or outperforms (by up to 6%) the existing state-of-the-art (SoTA) with significantly less compute and memory requirements.

1. Introduction

Decentralized machine learning is a branch of distributed learning which focuses on learning from data distributed across multiple agents/devices. Unlike Federated learning (Konečný et al., 2016), 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 comparably 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.

The key assumption to achieve state-of-the-art performance by all the above-mentioned algorithms is that the data is independent and identically distributed (IID) across the agents. This assumption does not hold in most of the real-world applications as the data distributions across the agents are significantly different (non-IID) based on the user pool (Hsieh et al., 2020). There are only a few works that try to bridge the performance gap between IID and non-IID data 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; 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. 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 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 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 in peer-to-peer de-

[*]Equal contribution [1]Electrical and Computer Engineering, Purdue University, West Lafayette, Indiana, USA. Correspondence to: Sai Aparna Aketi <saketi@purdue.edu>.

Federated Learning and Analytics in Practice Workshop at the 40^{th} *International Conference on Machine Learning*, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

centralized 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 data-variant cross-gradients are obtained through an additional round of communication. Finally, the local gradients are replaced with the weighted average of the self- and cross-gradients. 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 traditional decentralized algorithms (D-PSGD). To reduce the communication overhead, we propose the compressed version of *NGC* (*CompNGC*) by compressing the additional round of cross-gradient communication. We validate the performance of the proposed algorithm on various datasets, models, and graphs and achieve superior performance over non-IID data compared to the current state-of-the-art approach.

Contributions: In summary, we make the following contributions. 1) We propose Neighborhood Gradient Clustering (*NGC*) for a decentralized learning setting that utilizes self-gradients, model-variant cross-gradients, and data-variant cross-gradients to improve the learning over non-IID data among agents. 2) We present compressed version of Neighborhood Gradient Clustering (*CompNGC*) that reduces the additional round of cross-gradients communication by $32\times$. 3) Our experiments show that the proposed method outperforms the SoTA algorithm by upto 6% (1.62% on average) with significantly less compute and memory requirements at iso-communication cost. 4) We also show that when the weight associated with data-variant cross-gradients is set to 0, *NGC* performs $2-34\%$ better than D-PSGD without any communication overhead.

2. Background

The main goal of 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 privacy constraints. In particular, we solve the optimization problem of minimizing global loss function $\mathcal{F}(x)$ distributed across N agents as given in equation. 1.

$$\min_{x\in\mathbb{R}^d} \mathcal{F}(x) = \frac{1}{N}\sum_{i=1}^{N} f_i(x), \qquad (1)$$
$$and \;\; f_i(x) = \mathbb{E}_{d^i\sim D^i}[F_i(x; d^i)] \;\; \forall i$$

This is typically achieved by combining stochastic gradient descent (Bottou, 2010) with global consensus-based gossip averaging (Xiao & Boyd, 2004). The communication topology in this 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 to 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 at the beginning of the training. Algorithm. 2 in the appendix describes the flow of D-PSGD with momentum. The convergence of the Algorithm. 2 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. *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.

Algorithm 1 Neighborhood Gradient Clustering (*NGC*)

Input: Each agent $i \in [1, N]$ initializes model weights $x^i_{(0)}$, step size η, momentum coefficient β, averaging rate γ, 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 TRAIN( ) procedure

```
1. procedure TRAIN( )
2.   for k=$0, 1, \ldots, K-1$ do
3.     $g^{ii}_k = \nabla_x f_i(d^i_k; x^i_k)$   where   $d^i_k \sim D^i$
4.     SENDRECEIVE($x^i_k$)
5.     for each neighbor $j \in \{\mathcal{N}(i) - i\}$ do
6.       $g^{ji}_k = \nabla_x f_i(d^i_k; x^j_k)$
7.       if $\alpha \neq 0$ : SENDRECEIVE($g^{ji}_k$)
8.     end
9.     $\widetilde{g}^i_k = \sum_{j\in\mathcal{N}(i)}[(1-\alpha)w_{ji}g^{ji}_k + \alpha w_{ij}g^{ij}_k]$
10.    $v^i_k = \beta v^i_{(k-1)} - \eta\widetilde{g}^i_k$
11.    $\widetilde{x}^i_k = x^i_k + v^i_k$
12.    $x^i_{(k+1)} = \widetilde{x}^i_k + \gamma\sum_{j\in\mathcal{N}(i)}(w_{ij} - I_{ij})x^j_k$
13.  end
14. return
```

Self-Gradient $g^{ii}_k = \nabla_x F_i(x^i_k; d^i_k)$*:* 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.
Cross-Gradient $g_k^{ij} = \nabla_x F_j(x_k^i; d_k^j)$*:* 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.

The NGC algorithm: The flow of the Neighborhood Gradient Clustering (*NGC*) is shown in Algorithm. 1 and the form of the algorithm is similar to D-PSGD (Lian et al., 2017). The main contribution of the proposed *NGC* algorithm is the local gradient manipulation step (line 9 in Algorithm. 1). In the k^{th} 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, 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 are replaced with the weighted average of the above-defined cluster means. The mean of the model-variant cluster is weighted by $(1-\alpha)$ and the mean of the data-variant cluster is weighed by α where $\alpha \in [0,1]$ is a hyper-parameter referred to as *NGC* mixing weight.

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. 2 (assume ($w_{ij} = 1/m; m = |\mathcal{N}(i)|$)).

$$\begin{aligned}\widetilde{g}_k^i = g_k^{ii} &+ (1-\alpha) * \underbrace{\Big[\frac{1}{m}\sum_{j\in\mathcal{N}(i)} (g_k^{ji} - g_k^{ii})\Big]}_{\text{model variance bias } \epsilon_k^i} \\ &+ \alpha * \underbrace{\Big[\frac{1}{m}\sum_{j\in\mathcal{N}(i)} \frac{1}{m}(g_k^{ij} - g_k^{ii})\Big]}_{\text{data variance bias } \omega_k^i} \\ \epsilon_k^i &= \frac{1}{m} * \sum_{j\in\mathcal{N}(i)} \big(\nabla_x F(d_k^i; x_k^j) - \nabla_x F(d_k^i; x_k^i)\big) \\ \omega_k^i &= \frac{1}{m} * \sum_{j\in\mathcal{N}(i)} \big(\nabla_x F(d_k^j; x_k^i) - \nabla_x F(d_k^i; x_k^i)\big)\end{aligned} \tag{2}$$

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 hypothesize and show through our experiments that the addition of these bias terms to the local gradients improves the performance of decentralized learning over non-IID data by accelerating 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).

The Compressed NGC Algorithm: The *NGC* algorithm at every iteration involves two rounds of communication with the neighbors: 1) communicate the model parameters, and 2) communicate the cross-gradients. This communication 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. The pseudo-code for *CompNGC* is shown in Algorithm. 3 in the Appendix.

4. Experiments

In this section, we present the analysis of the following –

Table 1. Test accuracy comparisons for 5-layer CNN trained on non-IID CIFAR-10 over various graphs.

Method	Agents	Ring	Torus
D-PSGD	5	76.00 ± 1.44	-
	10	47.68 ± 3.20	55.34 ± 6.32
	20	44.85 ± 1.94	50.12 ± 1.91
NGC (ours) ($\alpha = 0$)	5	**82.20 ± 0.34**	-
	10	**67.43 ± 1.15**	**73.84 ± 0.33**
	20	**58.80 ± 1.30**	**64.55 ± 1.16**
CGA	5	82.20 ± 0.43	-
	10	72.96 ± 0.40	76.04 ± 0.62
	20	69.88 ± 0.84	73.21 ± 0.27
NGC (ours)	5	**83.36 ± 0.65**	-
	10	**75.34 ± 0.30**	**78.53 ± 0.56**
	20	**73.36 ± 0.88**	**75.11 ± 0.07**
CompCGA	5	82.00 ± 0.25	-
	10	71.41 ± 0.94	75.95 ± 0.41
	20	68.15 ± 0.79	71.71 ± 0.54
CompNGC (ours)	5	**82.91 ± 0.21**	-
	10	**74.36 ± 0.42**	**77.82 ± 0.20**
	20	**71.46 ± 0.85**	**73.62 ± 0.74**

(a) Datasets (Appendix A.3): CIFAR-10, CIFAR-100, Fashion MNIST and Imagenette (Husain, 2018). (b) Model architectures (Appendix A.4): 5-layer CNN, ResNet-20, LeNet-5, and MobileNet-V2. (c) Topologies: Ring and Torus. (d) Number of agents: varying from 5 to 20. Note

Table 2. Test accuracy comparisons for various datasets with non-IID sampling trained over undirected ring topology.

Method	Agents	Fashion MNIST (LeNet-5)	CIFAR-10 (ResNet-20)	CIFAR-100 (ResNet-20)	Imagenette (MobileNet-V2)
D-PSGD	5	86.43 ± 0.14	82.13 ± 0.84	44.66 ± 5.23	47.09 ± 9.20
	10	75.49 ± 0.32	31.66 ± 6.01	19.03 ± 13.27	32.81 ± 2.18
NGC (ours)	5	**88.49 ± 0.18**	**85.88 ± 0.58**	**55.96 ± 0.95**	**60.15 ± 2.17**
($\alpha = 0$)	10	**82.85 ± 0.24**	**66.02 ± 2.86**	**35.34 ± 0.32**	**36.13 ± 1.97**
CGA	5	90.03 ± 0.39	87.52 ± 0.50	56.43 ± 2.39	72.82 ± 1.25
	10	**87.61 ± 0.30**	79.98 ± 1.23	53.61 ± 1.07	61.97 ± 0.58
NGC (ours)	5	**90.61 ± 0.18**	**88.52 ± 0.19**	**56.50 ± 3.23**	**74.49 ± 0.93**
	10	87.24 ± 0.23	**84.02 ± 0.44**	**53.77 ± 0.15**	**64.06 ± 1.11**
CompCGA	5	90.45 ± 0.34	86.73 ± 0.34	55.74 ± 0.33	72.76 ± 0.44
	10	81.62 ± 0.37	73.63 ± 0.55	38.84 ± 0.54	59.92 ± 0.72
CompNGC (ours)	5	**90.48 ± 0.19**	**87.56 ± 0.34**	**57.51 ± 0.48**	**72.91 ± 1.06**
	10	**83.38 ± 0.39**	**78.50 ± 0.98**	**43.07 ± 0.32**	**61.91 ± 2.10**

that we use low-resolution (32×32) images of the Imagenette dataset for the experiments in Table. 2. We consider an extreme case of label-wise non-IID data where no two neighboring agents have the same class. We report the test accuracy of the consensus model averaged over three randomly chosen seeds. The hyperparameters for all the experiments are present in Appendix. A.7. We compare the proposed method with iso-communication baselines. The experiments on *NGC* ($\alpha = 0$) are compared with D-PSGD, *NGC* with CGA, and *CompNGC* with CompCGA. More details on the experimental setup and communication costs can be found in Appendix A.1 and A.6 respectively.

Results: We evaluate variants of *NGC* and *CompNGC* and compare them with respective baselines in Table. 1, for training 5-layer CNN on CIFAR-10 over various graph sizes and topologies. Further, we demonstrate the generalizability of *NGC* by evaluating it on various image datasets such as Fashion MNIST, and Imagenette and on challenging datasets CIFAR-100 in Table. 2. We observe that *NGC* with $\alpha = 0$ consistently outperforms D-PSGD in all cases with a significant performance gain varying from $2 - 34\%$ with an average improvement of 13.23%. The experiments show the superiority of *NGC* over CGA with a maximum improvement of 6.14% (1.62% on average) and *CompNGC* over CompCGA with a maximum gain of 6.28% (2.14% on average). Finally, through this exhaustive set of experiments, we demonstrate that the weighted averaging of self- and cross-gradients can be served as a simple plugin to boost the performance of decentralized learning over non-IID data.

Hardware Benefits: The proposed *NGC* algorithm is superior in terms of memory and compute efficiency while having equal communication cost as compared to *CGA* (refer Table. 3 and Appendix. A.5). Since *NGC* involves a simple weighted averaging operation, additional memory to store the cross-gradients is not required. CGA stores all the cross-gradients in a matrix form for quadratic programming projection of the local gradient. Moreover, the quadratic programming projection step (Goldfarb & Idnani, 1983) in CGA is much more expensive in compute and latency than the weighted averaging step of cross-gradients in *NGC*. Our experiments clearly show that *NGC* is superior to CGA in terms of test accuracy, memory efficiency, compute efficiency, and latency.

Table 3. Comparison of communication, memory, and compute overheads per mini-batch compared to D-PSGD. m_s: model size, N_i: number of neighbors, b: floating point precision, Q: compute for Quadratic Programming, B: compute for Backward Pass.

Method	Comm.	Memory	Compute
CGA	$\mathcal{O}(m_s N_i)$	$\mathcal{O}(N_i m_s)$	$\mathcal{O}(N_i B + Q)$
CompCGA	$\mathcal{O}(\frac{m_s N_i}{b})$	$\mathcal{O}(N_i m_s)$	$\mathcal{O}(N_i B + Q)$
NGC $\alpha = 0$	0	0	$\mathcal{O}(N_i B + m_s N_i)$
NGC	$\mathcal{O}(m_s N_i)$	0	$\mathcal{O}(N_i B + m_s N_i)$
CompNGC	$\mathcal{O}(\frac{m_s N_i}{b})$	$\mathcal{O}(N_i m_s)$	$\mathcal{O}(N_i B + m_s N_i)$

5. Conclusion

We propose the *Neighborhood Gradient Clustering* (*NGC*) that improves decentralized learning over non-IID data. Further, we present a compressed version of our algorithm (*CompNGC*) to reduce the communication overhead associated with *NGC*. We empirically validate the performance of proposed techniques over different models, datasets, graph sizes, and topologies. Finally, comparisons with the current state-of-the-art decentralized algorithm over non-IID data show the superior performance of *NGC* with significantly less compute and memory requirements setting the new state-of-the-art for decentralized learning over non-IID data.

Acknowledgements

This work was supported in part by, the Center for Brain-inspired Computing (C-BRIC), a DARPA-sponsored JUMP center, the Semiconductor Research Corporation (SRC), the National Science Foundation, the DoD Vannevar Bush Fellowship, and DARPA ShELL.

References

Aketi, S. A., Kodge, S., and Roy, K. 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.

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

Balu, A., Jiang, Z., Tan, S. Y., Hedge, C., Lee, Y. M., and Sarkar, S. 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.

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

Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., Ranzato, M., Senior, A., Tucker, P., Yang, K., et al. Large scale distributed deep networks. *Advances in neural information processing systems*, 25, 2012.

Esfandiari, Y., Tan, S. Y., Jiang, Z., Balu, A., Herron, E., Hegde, C., and Sarkar, S. Cross-gradient aggregation for decentralized learning from non-iid data. In *International Conference on Machine Learning*, pp. 3036–3046. PMLR, 2021.

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

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Hsieh, K., Phanishayee, A., Mutlu, O., and Gibbons, P. 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.

Husain, H. Imagenette - a subset of 10 easily classified classes from the imagenet dataset. *https://github.com/fastai/imagenette*, 2018.

Karimireddy, S. P., Rebjock, Q., Stich, S., and Jaggi, M. Error feedback fixes signsgd and other gradient compression schemes. In *International Conference on Machine Learning*, pp. 3252–3261. PMLR, 2019.

Konečnỳ, J., McMahan, H. B., Ramage, D., and Richtárik, P. Federated optimization: Distributed machine learning for on-device intelligence. 2016.

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

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Lian, X., Zhang, C., Zhang, H., Hsieh, C.-J., Zhang, W., and Liu, J. Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent. *Advances in Neural Information Processing Systems*, 30, 2017.

Lin, T., Karimireddy, S. P., Stich, S., and Jaggi, M. 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.

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

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520, 2018.

Tang, H., Lian, X., Yan, M., Zhang, C., and Liu, J. d^2: Decentralized training over decentralized data. In *International Conference on Machine Learning*, pp. 4848–4856. PMLR, 2018.

Xiao, H., Rasul, K., and Vollgraf, R. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

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

A. Appendix

A.1. Decentralized Learning Setup

The traditional decentralized learning algorithm (d-psgd) is described as Algorithm. 2. For the decentralized setup, we use an undirected ring and undirected torus graph topologies with a 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 have 5 peers per agent including itself and each edge has a weight of $\frac{1}{5}$. We consider an extreme case of non-IID distribution where no two neighboring agents have the same class. This is referred to as complete label-wise skew or 100% label-wise non-IIDness (Hsieh et al., 2020). In particular, for a 10-class dataset such as CIFAR-10 - each agent in a 5-agent system has data from 2 distinct classes, and each agent in a 10 agents system has data from a unique class. For a 20 agent system two agents that are maximally apart share the samples belonging to a class.

Algorithm 2 Decentralized Peer-to-Peer Training (*D-PSGD* with momentum)

Input: Each agent $i \in [1, N]$ initializes model weights $x^i_{(0)}$, step size η, averaging rate γ, 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
1. procedure TRAIN( )
2.   for k=$0, 1, \dots, K-1$ do
3.     $d^i_k \sim D^i$                                              // sample data from training dataset.
4.     $g^i_k = \nabla_x f_i(d^i_k; x^k_i)$                          // compute the local gradients
5.     $v^i_k = \beta v^i_{(k-1)} - \eta g^i_k$                      // momentum step
6.     $\widetilde{x}^i_k = x^i_k + v^i_k$                           // update the model
7.     SENDRECEIVE($\widetilde{x}^i_k$)                              // share model parameters with neighbors $N(i)$.
8.     $x^i_{(k+1)} = \widetilde{x}^i_k + \gamma \sum_{j\in\mathcal{N}(i)} (w_{ij} - I_{ij}) * \widetilde{x}^j_k$   // gossip averaging step
9.   end
10. return
```

A.2. CompNGC Algorithm

The section presents the pseudocode for Compressed *NGC* in Algorithm 3

A.3. Datasets

In this section, we give a brief description of the datasets used in our experiments. We use a diverse set of datasets each originating from a different distribution of images to show the generalizability of the proposed techniques.

CIFAR-10: CIFAR-10 (Krizhevsky et al., 2014) is an image classification dataset with 10 classes. The image samples are colored (3 input channels) and have a resolution of 32×32. There are $50,000$ training samples with 5000 samples per class and $10,000$ test samples with 1000 samples per class.

CIFAR-100: CIFAR-100 (Krizhevsky et al., 2014) is an image classification dataset with 100 classes. The image samples are colored (3 input channels) and have a resolution of 32×32. There are $50,000$ training samples with 500 samples per class and $10,000$ test samples with 100 samples per class. CIFAR-100 classification is a harder task compared to CIFAR-10 as it has 100 classes with very less samples per class to learn from.

Fashion MNIST: Fashion MNIST (Xiao et al., 2017) is an image classification dataset with 10 classes. The image samples are in greyscale (1 input channel) and have a resolution of 28×28. There are $60,000$ training samples with 6000 samples per class and $10,000$ test samples with 1000 samples per class.

Imagenette: Imagenette (Husain, 2018) is a 10-class subset of the ImageNet dataset. The image samples are colored (3 input channels) and have a resolution of 224×224. There are 9469 training samples with roughly 950 samples per class and 3925 test samples. We conduct our experiments on a resized low resolution of 32×32

Algorithm 3 Compressed Neighborhood Gradient Clustering (*CompNGC*)

Input: Each agent $i \in [1, N]$ initializes model weights $x^i_{(0)}$, step size η, averaging rate γ, 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 k=$0, 1, \dots, K-1$ do
3.     $d^i_k \sim D^i$                                   // sample data from training dataset
4.     $g^{ii}_k = \nabla_x f_i(d^i_k; x^i_k)$                 // compute the local self-gradients
5.     $p^{ii}_k = g^{ii}_k + e^{ii}_k$                          // error compensation for self-gradients
6.     $\delta^{ii}_k = (||p^{ii}_k||_1/d)sgn(p^{ii})$           // compress the compensated self-gradients
7.     $e^{ii}_{k+1} = p^{ii}_k - \delta^{ii}_k$                   // update the error variable
8.     SENDRECEIVE($x^i_k$)                             // share model parameters with neighbors $N(i)$
9.     for each neighbor $j \in \{N(i) - i\}$ do
10.      $g^{ji}_k = \nabla_x f_i(d^i_k; x^j_k)$               // compute neighbors' cross-gradients
11.      $p^{ji}_k = g^{ji}_k + e^{ji}_k$                        // error compensation for cross-gradients
12.      $\delta^{ji}_k = (||p^{ji}_k||_1/d)sgn(p^{ji})$         // compress the compensated cross-gradients
13.      $e^{ji}_{k+1} = p^{ji}_k - \delta^{ji}_k$                 // update the error variable
14.      if $\alpha \neq 0$ do
15.        SENDRECEIVE($\delta^{ji}_k$)                  // share compressed cross-gradients between $i$ and $j$
16.      end
17.    end
18.    $\widetilde{g}^i_k = (1-\alpha) * \sum_{j\in\mathcal{N}(i)} w_{ji} * \delta^{ji}_k + \alpha * \sum_{j\in\mathcal{N}(i)} w_{ij} * \delta^{ij}_k$   // modify local gradients
19.    $v^i_k = \beta v^i_{(k-1)} - \eta\widetilde{g}^i_k$           // momentum step
20.    $\widetilde{x}^i_k = x^i_k + v^i_k$                       // update the model
21.    $x^i_{(k+1)} = \widetilde{x}^i_k + \gamma\sum_{j\in\mathcal{N}(i)}(w_{ij} - I_{ij}) * x^j_k$   // gossip averaging step
22.  end
23. return
```

A.4. Network Architecture

We replace ReLU+BatchNorm layers of all the model architectures 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).

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 layers, 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.

ResNet-20: For ResNet-20 (He et al., 2016), we use the standard architecture with $0.27M$ trainable parameters except that BatchNorm+ReLU layers are replaced by EvoNorm-S0.

LeNet-5: For LeNet-5 (LeCun et al., 1998), we use the standard architecture with $61,706$ trainable parameters.

MobileNet-V2: We use the the standard MobileNet-V2 (Sandler et al., 2018) architecture used for CIFAR dataset with $2.3M$ parameters except that BatchNorm+ReLU layers are replaced by EvoNorm-S0.

A.5. 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 $\mathcal{O}(m_sN_i)$. In the case of *NGC* and CGA, there is an additional communication round for sharing data-variant cross gradients apart from sharing model parameters for the gossip averaging step. So, both these techniques incur a communication cost of $\mathcal{O}(2m_sN_i)$ and therefore an overhead of $\mathcal{O}(m_sN_i)$ 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 the 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 the 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. 4

Table 4. Memory overheads for various methods trained on different model architectures with CIFAR-10 dataset 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
ResNet-20	2.28	0	2.28	2.15

Table 5. Communication costs per agent in GBs for experiments in Table 1

Method	Agents	Ring	Torus
D-PSGD	5	17.75	-
and	10	8.92	13.38
NGC $\alpha = 0$	20	4.50	6.60
CGA	5	35.48	-
and	10	17.81	26.72
NGC	20	8.98	17.95
CompCGA	5	18.31	-
and	10	9.20	13.79
CompNGC	20	4.64	9.28

The computation of the cross-gradients (in both CGA and NGC algorithms) requires N_i forward and backward passes through the deep learning model at each agent. This is reflected as $\mathcal{O}(3N_i FP)$ in the compute overhead in Table. 3. We assume that the computing effort required for the backward pass is twice that of the forward pass. CGA algorithm involves quadratic programming projection step (Goldfarb & Idnani, 1983) to update the local gradients. Quadratic programming solver (quadprog) uses Goldfarb/Idnani dual algorithm. CGA uses quadratic programming to solve the following (Equation 3 -see Equation 5a in (Esfandiari et al., 2021)) optimization problem in an iterative manner:

$$\begin{aligned} \min_u \quad & \frac{1}{2} u^T G G^T u + g^T G^T u \\ \text{s.t.} \quad & u \geq 0 \end{aligned} \tag{3}$$

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, resulting in compute and time complexity of polynomial(degree$\geq$ 2) order. In contrast, NGC involves a simple averaging step that requires $O(m_s N_i)$ addition operations.

A.6. Communication Cost:

In this section we present the communication cost per agent in terms of Gigabytes of data transferred during the entire training process (refer Tables. 5 and 6. The D-PSGD and *NGC* with $\alpha = 0$ have the lowest communication cost ($1\times$). We emphasize that *NGC* with $\alpha = 0$ outperforms D-PSGD in decentralized learning over label-wise non-IID data for the same communication cost. *NGC* and CGA have $2\times$ communication overhead compared to D-PSGD where as *CompNGC* and CompCGA have $1.03\times$ communication overhead compared to D-PSGD. The compressed version of *NGC* and CGA compresses the second round of cross-gradient communication to 1 bit. We assume the full-precision cross-gradients to be of 32-bit precision and hence the *CompNGC* reduces the communication cost by $32\times$ compared to *NGC*.

Table 6. Communication costs per agent in GBs for experiments in Table 2

Method	Agents	Fashion MNIST (LeNet-5)	CIFAR-10 (ResNet-20)	CIFAR-100 (ResNet-20)	Imagenette (MobileNet-V2)
D-PSGD and	5	17.25	127.19	103.74	103.12
NGC $\alpha = 0$	10	8.61	63.84	51.89	51.60
CGA and	5	34.50	254.27	207.47	206.23
NGC (ours)	10	17.23	127.59	103.79	103.19
CompCGA and	5	17.79	131.16	106.98	106.34
CompNGC (ours)	10	8.88	65.84	53.52	53.21

A.7. Hyper-parameters

All the experiments were run for three randomly chosen seeds. We decay the step size by 10x after 50% and 75% of the training, unless mentioned otherwise.

Table 7. Hyper-parameters used for CIFAR-10 with non-IID data using 5-layer CNN model architecture presented in Table 1

Method	Agents (n)	Ring $(\alpha, \beta, \eta, \gamma)$	Torus $(\alpha, \beta, \eta, \gamma)$
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 CIFAR-10 on 5 layer CNN: All the experiments that involve 5layer CNN model (Table. 1) have stopping criteria set to 100 epochs. We decay the step size by $10\times$ in multiple steps at 50^{th} and 75^{th} epoch. Table 7 presents the α, β, η, and γ corresponding to the *NGC* mixing weight, momentum, step size, and gossip averaging rate. For all the experiments, we use a mini-batch size of 32 per agent. The stopping criteria is a fixed number of epochs. We have used Nesterov momentum of 0.9 for all CGA and *NGC* experiments whereas D-PSGD and *NGC* with $\alpha = 0$ have no momentum.

Hyper-parameters for CIFAR-10 on ResNet-20: All the experiments for the CIFAR-10 dataset trained on ResNet-20 architectures (Table. 2) have stopping criteria set to 200 epochs. We decay the step size by $10\times$ in multiple steps at 100^{th} and 150^{th} epoch. Table 8 presents the α, β, η, and γ corresponding to the ngc mixing weight, momentum, step size, and gossip averaging rate. For all the experiments, we use a mini-batch size of 32 per agent.

Table 8. Hyper-parameters used for CIFAR-10 with non-IID data using ResNet architecture presented in Table 2

Method	Agents	$(\alpha, \beta, \eta, \gamma)$
	5	$(-, 0.0, 0.1, 1.0)$
D-PSGD	10	$(-, 0.0, 0.1, 1.0)$
NGC (ours)	5	$(0.0, 0.0, 0.1, 1.0)$
$(\alpha = 0)$	10	$(0.0, 0.0, 0.1, 1.0)$
	5	$(-, 0.9, 0.1, 1.0)$
CGA	10	$(-, 0.9, 0.1, 1.0)$
	5	$(1.0, 0.9, 0.1, 1.0)$
NGC (ours)	10	$(1.0, 0.9, 0.1, 1.0)$
	5	$(-, 0.9, 0.01, 0.1)$
CompCGA	10	$(-, 0.9, 0.01, 0.1)$
	5	$(1.0, 0.9, 0.01, 0.1)$
CompNGC (ours)	10	$(1.0, 0.9, 0.01, 0.1)$

Hyper-parameters used for CIFAR-100, Fashion-MNIST and Imagenette: All the experiments in Table. 2 except for CIFAR-10 have stopping criteria set to 100 epochs. We decay the step size by $10\times$ in multiple steps at 50^{th} and 75^{th} epoch. Table 9 presents the α, β, η, and γ corresponding to the ngc mixing weight, momentum, step size, and gossip averaging rate. For all the experiments related to Fashion MNIST and Imagenette (low resolution of (32×32)), we use a mini-batch size of 32 per agent. For all the experiments related to CIFAR-100, we use a mini-batch size of 20 per agent.

Table 9. Hyper-parameters used for Table. 2

	Agents	Fashion MNIST	CIFAR-100	Imagenette
Method	(n)	$(\alpha, \beta, \eta, \gamma)$	$(\alpha, \beta, \eta, \gamma)$	$(\alpha, \beta, \eta, \gamma)$
D-PSGD	5	$(-, 0.0, 0.01, 1.0)$	$(-, 0.0, 0.1, 1.0)$	$(-, 0.0, 0.1, 1.0)$
	10	$(-, 0.0, 0.01, 1.0)$	$(-, 0.0, 0.1, 1.0)$	$(-, 0.0, 0.1, 1.0)$
NGC (ours)	5	$(0.0, 0.0, 0.01, 1.0)$	$(0.0, 0.0, 0.1, 1.0)$	$(-, 0.0, 0.1, 1.0)$
$(\alpha = 0)$	10	$(0.0, 0.0, 0.01, 1.0)$	$(0.0, 0.0, 0.1, 1.0)$	$(-, 0.0, 0.1, 1.0)$
CGA	5	$(-, 0.9, 0.01, 1.0)$	$(-, 0.9, 0.1, 1.0)$	$(0.0, 0.0, 0.01, 0.5)$
	10	$(-, 0.9, 0.01, 1.0)$	$(-, 0.9, 0.1, 0.5)$	$(0.0, 0.0, 0.01, 0.5)$
NGC (ours)	5	$(1.0, 0.9, 0.01, 1.0)$	$(1.0, 0.9, 0.1, 1.0)$	$(0.0, 0.0, 0.01, 0.5)$
	10	$(1.0, 0.9, 0.01, 1.0)$	$(1.0, 0.9, 0.1, 0.5)$	$(0.0, 0.0, 0.01, 0.5)$
CompCGA	5	$(-, 0.9, 0.01, 0.1)$	$(-, 0.9, 0.01, 0.1)$	$(0.0, 0.0, 0.01, 0.1)$
	10	$(-, 0.9, 0.01, 0.1)$	$(-, 0.9, 0.01, 0.1)$	$(0.0, 0.0, 0.01, 0.5)$
CompNGC (ours)	5	$(1.0, 0.9, 0.01, 0.1)$	$(1.0, 0.9, 0.01, 0.1)$	$(0.0, 0.0, 0.01, 0.1)$
	10	$(1.0, 0.9, 0.01, 0.1)$	$(1.0, 0.9, 0.01, 0.1)$	$(0.0, 0.0, 0.01, 0.5)$