Decentralized ML: Experiments on Topology and Privacy

Devrim Celik, Alec Flowers, Nina Mainusch
CS-439 - Optimization for Machine Learning - EPFL, Switzerland

***Abstract*—Decentralized machine learning is used to address problems such as data locality, ownership, privacy and high communication costs. We contribute an empirical evaluation of the impact of topology and differential privacy on model convergence and error in a decentralized learning system.**

I. Introduction

Centralized forms of machine learning have been thoroughly explored and are the current state of the art but have significant problems dealing with issues such as data locality, ownership, privacy and high communication costs. Decentralized machine learning addresses these problems by transferring control from a master server to each individual edge device. This allows data to reside at the source, protecting privacy, and lowering communication costs by shifting the medium from the data itself to a compressed representation. While decentralized networks do enhance privacy there are still problems since sensitive data can be learned from something as simple as a shared gradient. This motivates our exploration of adding differential privacy to the communication to ensure maximum privacy.

A. Centralized, Federated & Decentralized machine learning

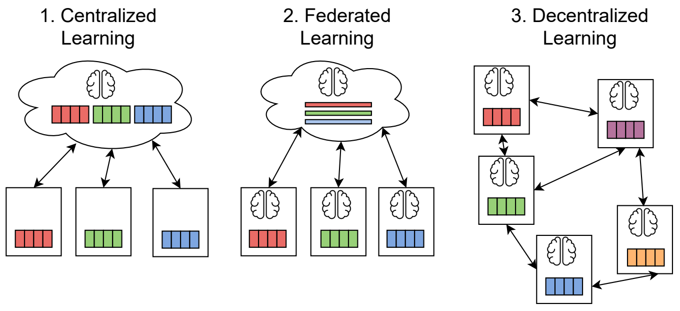

Fig. 1. Three types of organizational structures described graphically, where device units are depicted with rectangles, central units are depicted as clouds and arrows indicate communication between the units. Each brain represents a machine learning model that is trained by the corresponding unit.

In a traditional centralized machine learning setup, as visualized in Figure 1.1, data is moved from remote devices to a central system where models can be classically trained using stochastic gradient descent (SGD) which has runtime $O(1/\sqrt{T})$ if the target function is convex [1]. In federated learning, shown in Figure 1.2, each device collaboratively trains a global machine learning model and is organized by a central server. Data is not shared and the devices send updates to a server which aggregates these and in turn updates the shared models across the devices. The most well known optimization algorithm is federated averaging (parallel SGD/local SGD) with convergence $O(\frac{HM}{T} + \frac{\sigma}{\sqrt{TKM}})$[1] [2], [3]. Decentralized learning, depicted in Figure 1.3, does away with a central server, and replaces it with peer to peer communication between devices. There is no longer a global model such as in federated learning, as each device has its own unique model, however the process can be designed so that all the models converge. The main optimization algorithms for decentralized learning are gossip type (consensus) algorithms which are designed for computation and information exchange in an arbitrarily connected network of nodes [4].

B. Topology

In decentralized learning it has been shown that the convergence rate typically depends on the network topology. The network topology can be modeled as a graph $G = (V, E)$ with $\forall v \in V$ representing a device and $\forall \{i, j\} \in E$ representing a communication channel between two devices i and j. In particular the spectral gap $\delta \geq 0$ of the normalized adjacency matrix (W) of the communication topology graph is important in convergence.

Definition 1 (Spectral Gap) Assume $W \in [0,1]^{n\times n}$ is a symmetric doubly stochastic matrix with eigenvalues $1 = |\lambda_1(W)| > |\lambda_2(W)| \leq ... \leq |\lambda_n(W)|$ the spectral gap is

$$\delta = 1 - |\lambda_2(W)| \in (0,1] \tag{1}$$

The reason for this is because of the spectral gaps relation to connectivity of the graph. For example a fully connected graph has a $\delta = O(1)$ where a ring graph with node degree 2 has $\delta = O(1/n^2)$ on n nodes [5].

TABLE I
Spectral gaps for the normalized, doubly stochastic adjacency matrix for our graphs with 4, 16, and 32 nodes.

	4	16	32
FullyConnectedGraph	.666	.933	.968
CycleGraph	1.0	.076	.019
RingOfCliques	1.0	.104	.028
Torus2D	1.0	.195	.051

C. Differential Privacy

Differential privacy (DP) is a method of publishing certain pieces of data in a privacy-preserving fashion. The fact that in DP the practitioners can quantitatively control the amount of privacy that will be leaked differentiates it from the concept of anonymization or the generation of synthetic data. Let us

[1]M devices, loss functions are H-smooth, stochastic gradients with variance at most σ^2, T iterations, K local updates

represent the DP (random) algorithm as a function $\mathcal{A}$ and our original dataset as D, thus $\mathcal{A}(D)$ is the result of applying the DP mechanism to the original data. The algorithm $\mathcal{A}$ is said to provide (ϵ, δ)-differential privacy if, for all datasets D_1 and D_2 that differ on a single element (i.e., the data of one person), and all subsets S of image$(\mathcal{A})$:

$$P[\mathcal{A}(D_1) \in S] \leq exp(\epsilon) P[\mathcal{A}(D_2) \in S] + \delta, \tag{2}$$

where the probability is taken in respect to the randomness used by mechanism $\mathcal{A}$. A common dilemma in privacy-preserving technologiesis that the lower we set ϵ (the privacy budget), the "more random" $\mathcal{A}$ has to be, resulting in data that possesses less utility, but at the same time leaking less information.

Coming back to the project at hand, training machine learning models often requires large, representative datasets, which may be crowdsourced and contain sensitive information. The models should not expose private information in these datasets, which is why we utilized the `opacus` library within python for training the models. Opacus alters the standard learning procedure by adding three extra steps in the backpropagation step. Instead of simply calculating the gradients and using them to update the corresponding parameters, `opacus` is based on DP-SGD algorithm, which we wrote out in appendix B [6].

D. Related Work

We use the D-PSGD algorithm where they provide a theoretical analysis to show that decentralized can outperform centralized algorithms for distributed stochastic gradient descent [7]. Specifically in 1 we use the idea put forth in Lian et. al. Algorithm 1 to compute a neighborhood weighted average after the local update steps. Our algorithm runs in a loop but the updates occur so that it mimics a parallel decentralized algorithm.

In [8] they look at the contradiction between theory which says that worker communication topology should have a strong impact on the number of epochs needed to converge and experiments which show the opposite conclusion. Neglia et. al. note that in [9] and [10] they calculate the number of iterations T to approximate the minimum objective function by the desired error ϵ is

$$T \in O(\frac{1}{\epsilon^2 \delta}) \tag{3}$$

and conclude theoretically that a more connected network topology leads to faster convergence. However, their experiments show that when a dataset is statistically similar the topology has little influence on the number of epochs needed to converge.

II. Experiments

A. Data

To test the performance of the different architectures, we consider the task of predicting the class of a digit in the `MNIST` dataset [11]. The training and test set consist of 60000 and 10000 samples, respectively, and we normalized the input data. The data is partitioned between the nodes either *randomly* or *uniformly*, i.e. each node gets a random amount of data or exactly the same amount. Each node receives a unique partition of the data. Furthermore, we have an *iid* or *non-iid* setting, where in the former, the classes are distributed randomly between each node, but in the latter, every node only gets data from certain classes.

B. Implementation

The fundamental building block of our implementation is the `Node` class. Each `Node` represents one of the agents within the decentralized network. Using it, we are able to define the data as well as all learning related parameters individually for each node. The artificial neural networks within each node is built with `PyTorch`. It is a convolutional network consisting of 2 convolutional and 2 fully connected layers, where we apply max-pooling and batch normalisation after each convolutional layer. For each layer we use the `SeLU` activation function. The last fully connected layer is equipped with 10 activations (i.e. one per class) and the negative log-likelihood loss. Each `Node` possesses methods for training and testing the network and for sharing and receiving weights, such that all the weights of each individual's `PyTorch` model can be transmitted from one to the other.

For implementing the differential privacy mechanism in our nodes, we employed the `opacus` library, which is written as an extension on top of the standard `PyTorch` library, allowing us to define the previously discussed ϵ and δ parameter.

One step higher in the hierarchy is the `DecentralizedNetwork` class, which unifies all nodes in a complete decentralized network. The connectivity matrix between the nodes can be specified via the `graph_type` argument; we implemented a fully connected graph, a binomial graph, a ring of cliques, a circulant graph, a cycle graph and a two-dimensional torus, where all of them are based on the `networkx` python library [12]. The way in which weights are shared between the `Nodes` is based on a synchronous approach, i.e., after each batch of iterations all nodes perform one optimization step, share their weights with all of the neighbours and take the arithmetic mean over their own and the received weights.

C. Experimental setup

For our experiments we decided to compare four different topologies: the fully connected, ring, circle of cliques and torus graph, where we let the number of nodes vary between [4, 16, 32]. We ran the tests for all data combinations of [iid, non-iid] and [uniform, random] and furthermore once in a non-private and once in a private setting with parameters $\epsilon = 1.1$ and $\delta = 1e - 6$. In regard to the learning rate, for each setup we tried $\gamma \in [0.0001,\ 0.001,\ 0.01,\ 0.1,\ 0.2]$ and settled for the one which led to the best convergence, respectively.

In every epoch, each sample is used for training exactly once by a node. In the *iid uniform* setting with $[4,\ 16,\ 32]$ nodes, each node gets $[15000,\ 3750,\ 1875]$ samples, respectively, in

Algorithm 1: Decentralized Stochastic Gradient Descent based on (D-PSGD) [7]

```
initial weights $\omega_{0,n}$, step size $\gamma$, weight matrix $W$,
  data matrix $X$
for $e \in [Epochs]$ do
    for $b \in [Batches]$ do
        for $n \in [Nodes]$ do
            Compute a local stochastic gradient on
              local data $X_{e_b,n} \rightarrow \nabla F_{e_b,n}(\omega_{e_b,n}; X_{e_b,n})$
            Update the local optimization weights
              $\omega_{e_b+\frac{1}{2},n} \leftarrow \omega_{e_b,n} - \gamma \nabla F_{e_b,n}(\omega_{e_b,n}; X_{e_b,n})$
        end
        Communicate weights between neighboring
          nodes
        Compute the neighborhood weighted average
          and update weights for every node
          $\omega_{e_b+1,n} = \sum_{j=1}^{|nodes|} W_{n,j}\omega_{e_b,j}$
    end
end
```

all other settings the nodes differ in their available amount of samples. When a node has fewer samples than its neighbours and it has used all its batches in an epoch, it stops sharing its weights but still updates its model with the weight that the other nodes share with it, until the epoch is completed.

Our results are summarized in Table II, III and IV. For each setup we mention the learning rate γ, and for each graph we list three convergence statistics. We say that a configuration converged in x epochs if the difference of the training accuracy in epoch i to epoch $i+1$ does not change more than $\epsilon = 0.0001$. For this epoch, we state the training loss and test accuracy. If a model does not meet this condition in the available 50 epochs we denote it with '> 50'.

III. Discussion

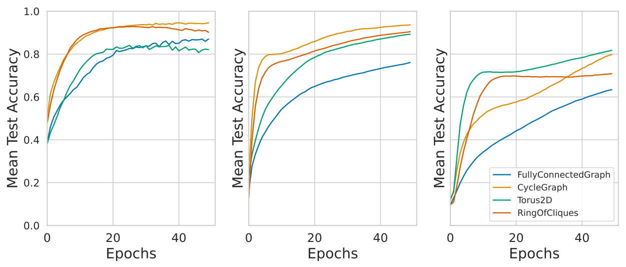

Fig. 2. Convergence plots from the *non-iid uniform* and no differential privacy setup. The number of nodes per plot is 4, 16, and 32, respectively.

In our *iid uniform* setup we expected to see similar results as in [13], where they noticed that convergence of average accuracy tends to happen faster for sparser networks. Our results corroborate these results and we see that no matter the topology the average node converges to a similar accuracy in the end. See Figure 4 in appendix C. Jiang et. al. then state that sparser topology may be faster for convergence but that topology density is more stable and therefore crucial for a decentralized learning paradigm. We interrogate this claim by introducing two new paradigms, a *non-iid* data setting and a privacy-conscious setting. In the *non-iid* data setting we give each node only 3 out of 10 classes from MNIST and expect sparser topologies to struggle with learning while denser topologies that are averaging more weights to be able to incorporate the other class information faster. We expected the cycle graphs with node degree 2 to learn the worst as amongst 3 nodes they only cover at most 9 out of the 10 class labels. As can be seen in Figure 2 in the *non-iid* setting the fully connected graph with 16 and 32 nodes test curve lags behind the other sparser graphs. The cycle graph also does quite well even being the top performing topology in the 16 node scenario. We postulate that potentially our *non-iid* scenario was not difficult enough which means that even in *non-iid* scenarios sparsity may be desired if the data is somewhat shuffled. For further experiments there is room to make the *non-iid* setting much more difficult by either restricting the number of classes or trying a more difficult task such as CIFAR-10 or

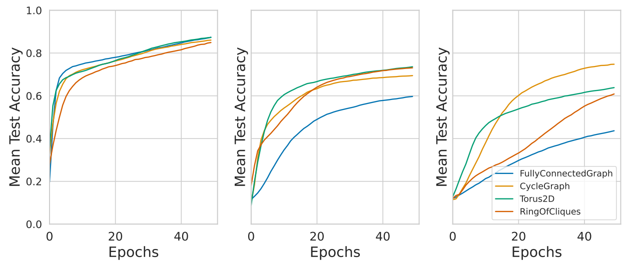

Fig. 3. Convergence plots from the *non-iid uniform* and differential privacy setup. The number of nodes per plot is 4, 16, and 32, respectively.

In the privacy setting our expectation was that it would converge slower and not reach the same level of final accuracy because of the introduction of noise. This is exactly what happened and can clearly be seen in the difference between Figure 2 and Figure 3. Comparing the final accuracy's from Table II the introduction of noise seems to hinder learning more in the more difficult setting of *non-iid random* and when we increase the number of nodes.

IV. Conclusion

We built a framework that would allow us to simulate different scenarios in decentralized machine learning. The main factors we adjusted were the type of topology, the number of nodes, the iid-ness of the data, and adding differential privacy. Our experiments show that sparsity, as measured by the spectral gap of the topology, performs unexpectedly well even in difficult settings while more densely connected graph learn more slowly as the number of nodes increases. This contradicts previous theory but falls in line with other experimental results.

References

[1] B. Gartner and M. Jaggi, “Optimization for machine learning, lecture notes,” 2021.

[2] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, “Communication-Efficient Learning of Deep Networks from Decentralized Data,” in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, A. Singh and J. Zhu, Eds., vol. 54. Fort Lauderdale, FL, USA: PMLR, 20–22 Apr 2017, pp. 1273–1282. [Online]. Available: http://proceedings.mlr.press/v54/mcmahan17a.html

[3] P. Kairouz, H. Brendan McMahan, B. Avent, A. Bellet, M. Bennis, A. Nitin Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings, R. G. D, H. Eichner, S. El Rouayheb, D. Evans, J. Gardner, Z. Garrett, A. Gascón, B. Ghazi, P. B. Gibbons, M. Gruteser, Z. Harchaoui, C. He, L. He, Z. Huo, B. Hutchinson, J. Hsu, M. Jaggi, T. Javidi, G. Joshi, M. Khodak, J. Konečn, A. Korolova, F. Koushanfar, S. Koyejo, T. Lepoint, Y. Liu, P. Mittal, M. Mohri, R. Nock, R. Pagh, H. Qi, D. Ramage, R. Raskar, M. Raykova, D. Song, W. Song, S. U. Stich, Z. Sun, A. Theertha Suresh, F. Tramèr, P. Vepakomma, J. Wang, L. Xiong, Z. Xu, Q. Yang, F. X. Yu, H. Yu, and S. Zhao, “Advances and Open Problems in Federated Learning,” Tech. Rep., 2021.

[4] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, “Randomized Gossip Algorithms,” *IEEE TRANSACTIONS ON INFORMATION THEORY*, vol. 52, no. 6, 2006.

[5] A. Koloskova, S. U. Stich, and M. Jaggi, “Decentralized Stochastic Optimization and Gossip Algorithms with Compressed Communication,” *36th International Conference on Machine Learning, ICML 2019*, vol. 2019-June, pp. 6088–6111, feb 2019. [Online]. Available: http://arxiv.org/abs/1902.00340

[6] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, “Deep learning with differential privacy,” *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, Oct 2016. [Online]. Available: http://dx.doi.org/10.1145/2976749.2978318

[7] X. Lian, C. Zhang, H. Zhang, C.-J. Hsieh, W. Zhang, and J. Liu, “Can Decentralized Algorithms Outperform Centralized Algorithms? A Case Study for Decentralized Parallel Stochastic Gradient Descent,” Tech. Rep.

[8] G. Neglia, C. Xu, D. Towsley, and G. Calbi, “Decentralized gradient methods: does topology matter?” feb 2020. [Online]. Available: http://arxiv.org/abs/2002.12688

[9] A. Nedić and A. Ozdaglar, “Distributed subgradient methods for multi-agent optimization,” *IEEE Transactions on Automatic Control*, vol. 54, no. 1, pp. 48–61, 2009.

[10] J. Duchi, A. Agarwal, and M. Wainwright, “Dual Averaging for Distributed Optimization: Convergence Analysis and Network Scaling,” *IEEE Transactions on Automatic Control*, vol. 57, no. 3, pp. 592–606, may 2010. [Online]. Available: http://arxiv.org/abs/1005.2012http://dx.doi.org/10.1109/TAC.2011.2161027

[11] Y. LeCun and C. Cortes, “MNIST handwritten digit database,” 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[12] A. Hagberg, P. Swart, and D. S Chult, “Exploring network structure, dynamics, and function using networkx,” Los Alamos National Lab.(LANL), Los Alamos, NM (United States), Tech. Rep., 2008.

[13] Z. Jiang, A. Balu, C. Hegde, and S. Sarkar, “Collaborative Deep Learning in Fixed Topology Networks,” *Advances in Neural Information Processing Systems*, vol. 2017-December, pp. 5905–5915, jun 2017. [Online]. Available: http://arxiv.org/abs/1706.07880

Appendix A
Code

To see our implementation and the code that produced our tables and plots, visit our *GitHub Repository*.

Appendix B
Algorithm

We state the DP-SGD algorithm on which the `opacus` library is based on.

Algorithm 2: DP-SGD

```
Input: Examples $\{x_1, ..., x_N\}$, loss function
    $\mathcal{L}(\theta)\frac{1}{N}\sum_i \mathcal{L}(\theta, x_i)$. Parameters: learning rate
    $\eta_t$, noise scale $\sigma$, group size $L$, gradient norm
    bound $C$.
Initialize $\theta_0$ randomly.;
for $t \in [T]$ do
    Take a random sample $L_t$ with sampling
     probability $\frac{L}{N}$;
    Compute gradient: For each $i \in L_t$, compute
     $g_t(x_i) \leftarrow \nabla_{\theta_t}\mathcal{L}(\theta_t, x_i)$;
    Clip gradient: $\bar{g}_t(x_i) \leftarrow \frac{g_t(x_i)}{\max(1, \frac{||g_t(x_i)||_2}{C})}$;
    Add noise: $\tilde{g}_t \leftarrow \frac{1}{L}(\sum_i \bar{g}_t(x_t)) + \mathcal{N}(0, \sigma^2 C^2 I))$;
    Descent: $\theta_{t+1} \leftarrow \theta_t - \eta_t \tilde{g}_t$
end
Output: $\theta_T$ and compute the overall privacy cost
     $(\epsilon, \delta)$ using a privacy accounting method
```

Appendix C
Figures

In our report we present the convergence plots for the *non-iid uniform* setup. For completeness we will also list the plots for the *iid uniform*, *iid random* and *non-iid random* setup.

A. *iid uniform setup*

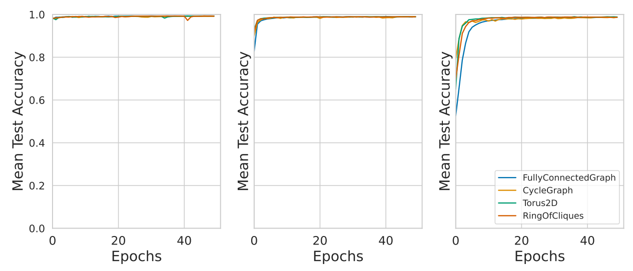

Fig. 4. Convergence plots from the *iid uniform* and no differential privacy setup. The number of nodes per plot is 4, 16, and 32, respectively.

For the *iid uniform* setup we can see in Figure 4 that for all graphs we achieve a perfect mean test accuracy over all devices after only a couple of epochs. Only for the graphs with 32 nodes, convergence takes sightly longer. If adding differential privacy as depicted in Figure 5, we are not able to achieve a perfect test accuracy, but for 4 and 16 nodes we still manage to

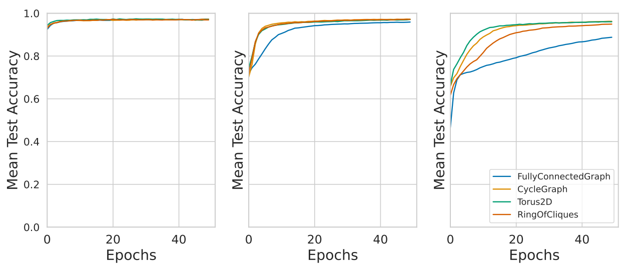

Fig. 5. Convergence plots from the *iid uniform* and differential privacy setup. The number of nodes per plot is 4, 16, and 32, respectively.

converge. For 32 nodes on the other hand convergence happens slower, and especially the fully connected graph struggles to achieve a test accuracy as good as the others.

B. *iid random setup*

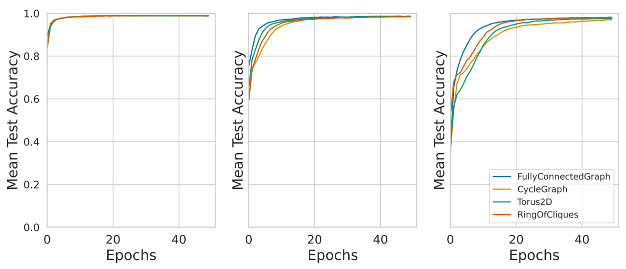

Fig. 6. Convergence plots from the *iid random* and no differential privacy setup. The number of nodes per plot is 4, 16, and 32, respectively.

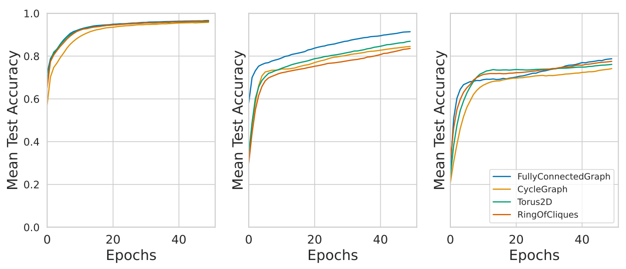

Fig. 7. Convergence plots from the *iid random* and differential privacy setup. The number of nodes per plot is 4, 16, and 32, respectively.

In the iid random setup, where each device only gets data from a certain class, we see a similar but little slower convergence behaviour compared to the *iid uniform* setup. This time the fully connected graph performs as good or even better than the other topologies. In the differential private configuration with 32 nodes the networks start struggling to achieve a top test accuracy and stagnate before 80%.

C. *non-iid random setup*

The *non-iid random* setup is the hardest one for our decentralized models to learn. Not only does each device only get data from a certain class, but also a random amount of

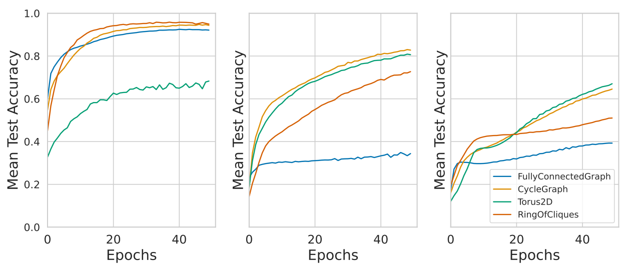

Fig. 8. Convergence plots from the *non-iid random* and no differential privacy setup. The number of nodes per plot is 4, 16, and 32, respectively.

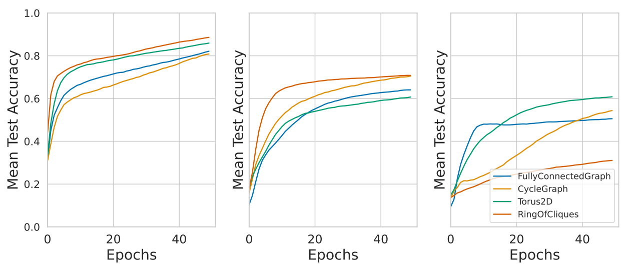

Fig. 9. Convergence plots from the *non-iid random* and differential privacy setup. The number of nodes per plot is 4, 16, and 32, respectively.

data assigned. If the node ends up having few samples for a label, this will be underrepresented and the network will have a hard time achieving a good test performance. This can be one explanation for why in Figure 8 and 9 certain graphs exhibit a poor test performance. Also at the end of the available 50 epochs, some models still seem to be improving and would need more training epochs.

Appendix D
Tables

Corresponding to the setups and figures from appendix C, we show the convergence behaviour of our models in Table II, III and IV, where we list all the results for the models with 4, 16, and 32 nodes, respectively.

TABLE II
4 NODES

Setup	γ	**Graphs**	**Epoch**	**Train Loss**	**Test Acc.**
IID Uniform	0.01	FullyConnectedGraph	23	0.0002	0.9864
		CycleGraph	26	0.0013	0.9848
		Torus2D	7	0.0143	0.9845
		RingOfCliques	7	0.0148	0.9844
IID Uniform Private	0.01	FullyConnectedGraph	15	0.1520	0.9509
		CycleGraph	19	0.1531	0.9579
		Torus2D	9	0.1403	0.9572
		RingOfCliques	30	0.1557	0.9556
IID Random	0.0001	FullyConnectedGraph	13	0.0595	0.9779
		CycleGraph	38	0.0621	0.9837
		Torus2D	15	0.0617	0.9810
		RingOfCliques	8	0.0830	0.9712
IID Random Private	0.0001	FullyConnectedGraph	>50	0.1457	0.9537
		CycleGraph	36	0.1803	0.9361
		Torus2D	>50	0.1453	0.9469
		RingOfCliques	13	0.2455	0.9001
Non-IID Uniform	0.0001	FullyConnectedGraph	29	0.0901	0.7229
		CycleGraph	27	0.2853	0.9257
		Torus2D	>50	0.1283	0.9327
		RingOfCliques	>50	0.1400	0.9443
Non-IID Uniform Private	0.0001	FullyConnectedGraph	>50	0.4352	0.8417
		CycleGraph	>50	0.8369	0.8376
		Torus2D	>50	0.4450	0.8413
		RingOfCliques	>50	0.7288	0.8308
Non-IID Random	0.0001	FullyConnectedGraph	>50	0.0730	0.9225
		CycleGraph	20	0.1564	0.8665
		Torus2D	16	0.1390	0.5032
		RingOfCliques	32	0.0622	0.9197
Non-IID Random Private	0.0001	FullyConnectedGraph	16	1.4456	0.6528
		CycleGraph	>50	0.6959	0.7704
		Torus2D	9	1.2229	0.6922
		RingOfCliques	>50	0.3553	0.8464

TABLE III
16 NODES

Setup	γ	**Graphs**	**Epoch**	**Train Loss**	**Test Acc.**
IID Uniform	0.01	FullyConnectedGraph	8	0.0628	0.9840
		CycleGraph	15	0.0156	0.9842
		Torus2D	20	0.0097	0.9842
		RingOfCliques	9	0.0307	0.9904
IID Uniform Private	0.01	FullyConnectedGraph	26	0.1849	0.9339
		CycleGraph	15	0.1413	0.9532
		Torus2D	41	0.1315	0.9744
		RingOfCliques	14	0.1571	0.9583
IID Random	0.0001	FullyConnectedGraph	24	0.0795	0.9795
		CycleGraph	30	0.0942	0.9775
		Torus2D	10	0.3016	0.9421
		RingOfCliques	18	0.1559	0.9777
IID Random Private	0.0001	FullyConnectedGraph	>50	0.2699	0.8851
		CycleGraph	24	1.1281	0.7429
		Torus2D	>50	0.4870	0.8490
		RingOfCliques	14	1.5346	0.7073
Non-IID Uniform	0.0001	FullyConnectedGraph	34	1.9198	0.6763
		CycleGraph	>50	0.6539	0.9192
		Torus2D	47	1.0500	0.8441
		RingOfCliques	18	1.4788	0.7902
Non-IID Uniform Private	0.0001	FullyConnectedGraph	>50	2.0603	0.5462
		CycleGraph	>50	1.8268	0.6672
		Torus2D	44	1.9188	0.7086
		RingOfCliques	45	1.6598	0.6988
Non-IID Random	0.0001	FullyConnectedGraph	40	1.4010	0.3601
		CycleGraph	>50	0.3703	0.8278
		Torus2D	27	1.0256	0.7018
		RingOfCliques	3	2.2882	0.3376
Non-IID Random Private	0.0001	FullyConnectedGraph	>50	2.0750	0.6549
		CycleGraph	>50	1.5378	0.7234
		Torus2D	26	1.4966	0.5668
		RingOfCliques	27	1.9804	0.6530

TABLE IV
32 NODES

Setup	γ	**Graphs**	**Epoch**	**Train Loss**	**Test Acc.**
IID Uniform	0.01	FullyConnectedGraph	13	0.1109	0.9855
		CycleGraph	45	0.0024	0.9896
		Torus2D	24	0.0123	0.9914
		RingOfCliques	17	0.0316	1.0000
IID Uniform Private	0.01	FullyConnectedGraph	>50	0.3892	0.9062
		CycleGraph	42	0.1660	0.9587
		Torus2D	37	0.1718	0.9561
		RingOfCliques	>50	0.2026	0.9535
IID Random	0.0001	FullyConnectedGraph	20	0.1668	0.9699
		CycleGraph	28	0.3504	0.9449
		Torus2D	26	0.2075	0.9725
		RingOfCliques	21	0.1560	0.9810
IID Random Private	0.0001	FullyConnectedGraph	10	2.0063	0.6856
		CycleGraph	43	1.5771	0.7426
		Torus2D	>50	1.1874	0.7786
		RingOfCliques	29	1.3188	0.7374
Non-IID Uniform	0.0001	FullyConnectedGraph	>50	2.2148	0.6272
		CycleGraph	>50	1.3124	0.8504
		Torus2D	18	2.0856	0.7269
		RingOfCliques	18	2.1649	0.6778
Non-IID Uniform Private	0.0001	FullyConnectedGraph	>50	2.2620	0.4382
		CycleGraph	>50	2.0726	0.7578
		Torus2D	41	2.1279	0.5871
		RingOfCliques	8	2.2980	0.2347
Non-IID Random	0.0001	FullyConnectedGraph	4	2.3635	0.3233
		CycleGraph	>50	1.2416	0.8311
		Torus2D	47	1.1472	0.7857
		RingOfCliques	>50	1.7693	0.3880
Non-IID Random Private	0.0001	FullyConnectedGraph	>50	1.7446	0.5298
		CycleGraph	6	2.2925	0.2671
		Torus2D	23	2.2218	0.6097
		RingOfCliques	>50	2.1935	0.3787