Addressing the Challenges of Federated Learning

Pietro Cagnasso
Politecnico di Torino
pietro.cagnasso@studenti.polito.it

Giuseppe Galilei
Politecnico di Torino
giuseppe.galilei@studenti.polito.it

Nicolò Vergaro
Politecnico di Torino
nicolo.vergaro@studenti.polito.it

***Abstract*—This project aims to analyze the growing scenario of Federated Learning and to address some of its main challenges, which are statistical heterogeneity, systems heterogeneity, and privacy. To do so, we tested some of the most relevant algorithms: FedAVG, FedGKT, and FedDyn in the case of i.i.d. balanced, non-i.i.d. balanced, and non-i.i.d. unbalanced data distribution. Then, we moved to a privacy analysis of Federated Learning, trying the gradient attack method to recover original data from gradient information.**

I. Introduction

Federated Learning is a machine learning scenario introduced by Google in 2017. The aim is to train a model in a non-centralized way as the result of a collaboration between several devices (or clients), each of which maintains exclusive ownership of its data, granting higher privacy levels. In the standard setting for this scenario, each client trains the model on its local data, then shares only the model's parameters (or gradients) with a central server which aggregates the information received from active clients, producing a new model. Finally the resulting model's parameters are sent from the server to the clients, so that a new training round can start.

In this report, we will document our journey in Federated Learning and address some of its challenges [1]. We started by training a typical centralized model to set an upper bound for the federated scenario. Then we implemented the first-ever federated algorithm to set a baseline for comparisons with other federated algorithms. After, we addressed some real-world challenges of Federated Learning:

- Statistical heterogeneity of data, by simulating different kinds of data distributions, specifically non-i.i.d. balanced and non-i.i.d. unbalanced.
- System heterogeneity, by implementing an algorithm that allows different model complexities at the edge and on the server to work together.
- Privacy preservation, by implementing a gradient inversion attack which tries to reconstruct original private data by using gradients of the model trained at the edge and shared with the server.

Finally, we focused on a recently introduced algorithm that promises better performances than existing Federated Learning methods. We tried to evaluate and verify the claims reported in the original paper.

[1]Code available in this repository.

Related Works

FedAVG was proposed by McMahan et al. [1] in 2016 and is the first-ever introduced method in Federated Learning.

Other methods that improved FedAVG's performances have been proposed, such as FedGKT [2] and FedDyn [3], which we worked with in this project. Other proposed methods that we did not implement are SCAFFOLD [4], FedProx [5], and FairAVG [6].

Thanks to its improvements in privacy, Federated Learning can also be exploited in activities using highly sensible data like healthcare [7]. However, even if FL grants better assurances for sensible data, it is not safe from any attack, as proved by [8], [9]. Gradients sent on the network can be indeed exploited to extract information about the data that originated them.

II. Methods

A. Federated Averaging

FedAVG is a federated learning algorithm introduced by McMahan et al. in 2017 [1]. This algorithm assumes the same network architecture on the server and the client. It works iteratively in communication rounds. In the first round, some active clients are selected. Each client trains its model on its data and sends the obtained model's parameters to the server. The server averages the received parameters weighting the contribution of each client on the fraction of data it owns. The server then sends the updated parameters to clients. In the following rounds, newly selected active clients will start their training with the updated model received from the server.

The authors showed that this method is very effective; in the i.i.d. case, given an adequate number of communication rounds, it can achieve similar performance to centralized training. Moreover, they showed that this method requires a reasonable amount of time, even when training large models.

B. Group Knowledge Transfer

FedGKT is an alternative to FedAVG proposed by He et al. [2] in 2020. The main aim is to allow federated learning to happen in a scenario where clients and server do not have the same computational power. The idea is to train small CNNs on low-powered edge devices and periodically transfer knowledge to a bigger CNN on the server. The smaller network consists of just a feature extractor layer and a simple classifier. The larger one is characterized by lacking only the first layer that performs feature extraction, it will be indeed trained using as

input the features extracted from the edge-side model's first layer.

This method works as follows: each client trains his model on its own data and sends the extracted features, logits, and labels to the server. Then, the server starts training using as input the received features. It optimizes a loss computed using the received labels and logits, then sends its predicted labels and logits back to clients. Finally, each client incorporates the server's information into the loss used to train its model.

This algorithm obtains comparable results to FedAVG while having the following advantages: reduced demand for edge computation, because a smaller CNN is used; lower communication bandwidth, because feature maps are sent instead of a whole model; asynchronous training, because the server can start training as soon as it receives any input.

C. Dynamic Regularization

FedDyn is a federated learning algorithm firstly introduced by Acar et al. in 2020 [3]. Its aim is to increase the amount of computation performed by client devices in each round and reduce the overall number of communication rounds required to reach similar results compared to FedAVG.

The increased computation would be especially beneficial in the non-i.i.d. setting, where each client optimizes on a data distribution that may not be representative of the global one, causing clients to struggle to align their local objective to the global one.

FedAVG mitigates this problem by limiting the number of epochs on local devices, which prevents clients from overfitting their data and ruining the global model; this is the phenomenon of "client-drift".

Feddyn instead solves the problem by modifying the client's loss function so the optimization done at the client level is coherent with the global one. This novel loss function adds two terms to the standard loss:

- A *linear* penalty term dependant on the local state, that debiases the effect of local losses ensuring the alignment of stationary points between local and global losses.
- A *quadratic* penalty term, that ensures convergence in the long run by controlling the similarity between the local and global model.

One of FedDyn's novelties is the use of local states for both server and clients.

The algorithm works like follows: each active client executes one or more local epochs to find its local optimum, then it updates its local state (the gradient of the loss function) and sends the model to the server. The server first updates its state: a linear combination of models from active devices in current and previous communication rounds. Then it updates the global model averaging active devices' models without weighting their contribution and exploiting its local state. In the following communication rounds, active clients will start their training with the updated model computed by the server.

This algorithm uses only the hyperparameter α and has been proven to converge with a rate of $\mathcal{O}(1/T)$, where T is the number of communication rounds.

It's worth noting that the choice of using internal states does not increase the communication cost since internal states are not shared; this is an advantage against similar approaches like SCAFFOLD [4].

III. Experiments and Results

To ensure better comparability and reliability of results, we carried out all our tests within this framework:

1) Run all the experiments for 50 epochs (centralized training) or rounds (federated scenario).
2) Perform all the experiments with three different seeds (0, 128, 479) and average the results; this allowed for smoother trends and more precise accuracy.
3) Use CIFAR10 as dataset, partitioned using the sampling strategy described below.
4) Run all three typical cases of the federated scenario: i.i.d. balanced (hereafter i.i.d.), non-i.i.d. balanced, and non-i.i.d. unbalanced.
5) All experiments in the federated case were performed with 100 available clients, of which only 10% were active in each round.
6) Experiments, other than those just aiming to prove the functionality of our implementation, make use of a ResNet [10] with Batch Normalization [11] and Group Normalization [12] layers.

As a sampling strategy, we adopted the same proposed by McMahan et al. [1] which can be described as follows:

- *i.i.d.*: shuffle the dataset and then split it into equal parts between all the clients.
- *non-i.i.d.*: sort the data based on its label and split it into shards of equal size. In the balanced case we assign the same number of shards to each client, while in the unbalanced one we assign a random number to each client with each having at least one shard.

In the i.i.d. case each client is able to see all the classes. For both non-i.i.d. cases we chose to create 200 shards, to ensure that a similar average number of classes is seen by each client in both balanced and unbalanced cases. Using this sampling strategy each client sees a number of classes as reported in Table I, while Fig.7 graphically shows data distribution across devices.

TABLE I
MEAN AND STANDARD DEVIATION OF CLASSES SEEN BY EACH CLIENT IN THE THREE MAIN CASES.

Case	Classes seen
i.i.d.	10 ± 0
non-i.i.d. balanced	1.94 ± 0.24
non-i.i.d. unbalanced	1.84 ± 1.1

A. Find a Comparison Term: Centralized Training

First of all, we need a comparison term to be able to evaluate federated algorithms' performance. Since the goal of the project is to evaluate algorithms in the federated domain, a natural choice is centralized training, whose results

will be the upper bound for the various algorithms in the federated scenario. Another goal of this step is to identify some hyperparameters we can use in the federated scenario.

We trained the ResNet50 from scratch because the implementation in the framework did not provide a direct way to change the normalization layer using the pre-trained version. The hyperparameters on which we tuned our model are learning rate, weight decay, and momentum. To identify the best ranges for those hyperparameters we performed a random search using logarithmic distribution for the first two, and uniform distribution for the latter. Based on these results we selected some values (Table II) to do a complete grid search on 20 epochs.

We also tested:

- Different batch sizes: 32, 64, and 128. We ended up choosing 128, aiming to reduce the time required by the model to be trained. This choice may have disadvantaged Group Normalization since its advantage over Batch Normalization is only visible when dealing with small batch sizes.
- Both SGD and Adam as optimizers. We found the choice of the optimizer to be less relevant than all other hyperparameters. Therefore, to have results directly comparable with the ones reported in most of the papers, we preferred SGD.

The best configuration selected from the grid search turned out to be: `lr`=1e-2, `weight_decay`=1e-5, `momentum`=0.9. We trained the model for 50 epochs using this configuration and a learning rate scheduler that reduced it to one-third at epochs 20, 30, and 40. The best accuracies achieved are 92.28% and 88.18% for Batch Normalization and Group Normalization, respectively; Fig. 1 and 8.

TABLE II
HYPERPARAMETERS USED DURING GRID SEARCH TUNING

Hyperparameter	Values
`lr`	[1e-2, 1e-3]
`weight_decay`	[1e-4, 1e-5, 0]
`momentum`	[0.5, 0.7, 0.9]

B. Federated Baseline: FedAVG

The de facto standard as a federated scenario baseline is FedAVG since it is the first federated algorithm ever introduced. Executing this algorithm, we opted for only one local epoch in each client. We first tried to use the hyperparameters found by the grid search in centralized training. We noticed bad performance, especially in the non-i.i.d. case; Fig. 9.

To solve this inconvenience we tried to set up the optimizer using some of the best-performing configurations in centralized training. This step can be interpreted as a quick hyperparameter tuning. We found out that the configuration `lr`=1e-2, `weight_decay`=0, `momentum`=0.5 grants overall better trends and accuracies in i.i.d. as well as in non-i.i.d. cases. This optimizer's configuration in centralized training

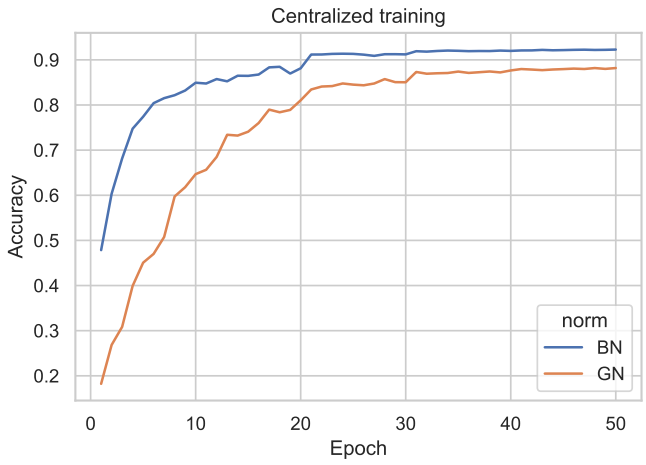

Fig. 1. Test accuracy with the best configuration found. BN = Batch Normalization, GN = Group Normalization.

lowers the accuracy by less than 2.5% from the best one; Fig. 8.

The best accuracies we got running this experiment in the three cases with both types of normalization layers are reported in Table III, accuracy trends and the comparison between the three cases are shown in Figures 2 and 10. The i.i.d. trend at round 50 is in line with the results reported in the original paper. Since it has not reached the plateau yet, this suggests that running it for more communication rounds would result in accuracies closer to the ones registered in centralized training.

TABLE III
FEDAVG ACCURACY BY SCENARIO AND NORMALIZATION LAYER.

Normalization	i.i.d.	non-i.i.d. balanced	non-i.i.d. unbalanced
BN	55.217%	19.577%	32.867%
GN	32.000%	20.557%	25.587%

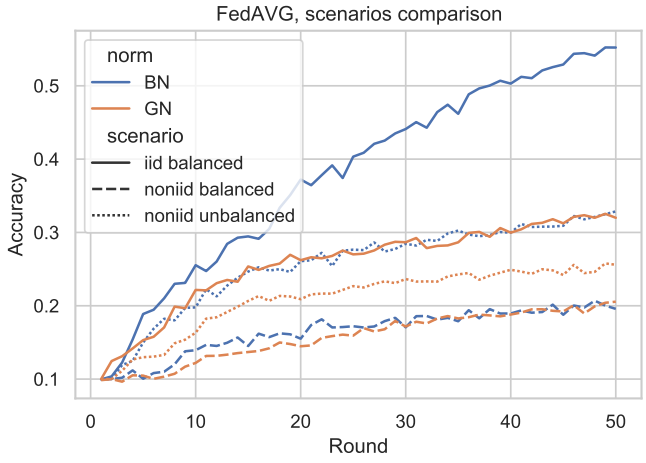

Fig. 2. Comparison between the three scenarios in which we tested FedAVG.

C. Simulate System Heterogeneity: FedGKT

To perform these experiments, we used the same hyper-parameters used to test the federated baseline. The CNNs we used in our experiments are ResNet-8 on the clients and ResNet-49 on the server. The advantage of using the lighter ResNet on the clients can also be seen from the number of parameters, in fact it has $\sim$ 2200 times less parameters than the ResNet on the server (23.5M vs 10.5k).

We run 10 epochs on the server and 1 epoch on the clients for each communication round. Following the paper's experiments, we tested two different losses scenarios. In particular, we kept the same loss on the client, that is a combination of both the CE and KD loss with parameter $\alpha = 0.5$. On the server we tried to use the CE loss alone and both the CE loss and KD loss with the same α parameter. We can see results in Table IV and Fig. 3. More extensive comparisons and respective losses are illustrated in Fig. 11 (CE loss) and Fig. 12 (CE+KD loss).

During the implementation, we faced memory issues due to limitations on the amount of available memory in both Google Colab and Kaggle. To address these issues we stored for each client the extracted features, labels and logits, while for the server we store the logits. We then remove all this information at the end of each communication round, otherwise they would accumulate and make us quickly run out of memory.

TABLE IV
FEDGKT ACCURACY BY SCENARIO, NORMALIZATION AND LOSSES

Loss	Normalization	i.i.d.	non-i.i.d. bal.	non-i.i.d. unbal.
CE	BN	76.789%	21.180%	26.392%
	GN	74.883%	31.455%	31.224%
CE+KD	BN	51.950%	28.283%	26.351%
	GN	51.053%	23.650%	26.824%

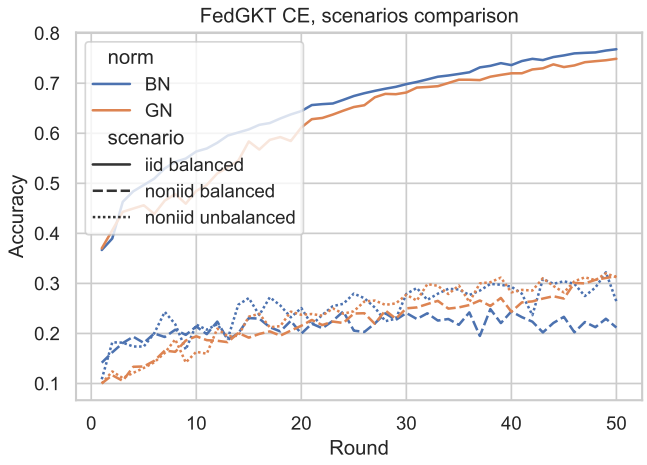

Fig. 3. Comparison between the three scenarios in which we tested FedGKT using only CE loss in the server.

D. Maintaining Privacy: Gradient Inversion Attack

Data privacy is one of the most important benefits of using federated learning, however, even if data never leaves client devices, it has been demonstrated the possibility of recovering it from the model parameters (or gradients) which are sent to the server. Several of these "Gradient Inversion" attacks exist, they have been studied and evaluated in the literature. Generally their performance depends strongly on meeting certain conditions, such as knowing the batch size and the target label of data.

We reproduced the "inverting gradients" attack introduced by Geiping et al. [8], which assumes both the conditions above. This attack is performed by a "honest but curious server" which does not interfere with the training process, but has access to gradients and uses them to reconstruct data. The gradients here are not interesting for their magnitude (which is just a signal of the state of training), but for their "high-dimensional" dimension (the direction), so the objective becomes finding images that lead to similar changes in model prediction as the original data.

To perform this experiment we trained a ResNet50 using FedAVG for 100 communication rounds, then we computed the gradients as if they were computed by a client that received the trained model from the server. We let the gradient attack run for 3000 iterations, obtaining understandable results in all four tested images from different categories as we can see in Figures 4 and 13. In the reconstructed images we can clearly understand which one are the main subjects: truck, deer, dog, and horse; and even their correct color for the large part of the image.

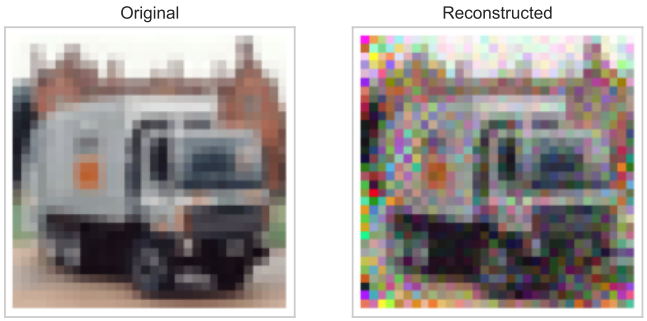

Fig. 4. Comparison between the original image and the reconstructed one. This is the result of 3000 iterations of the algorithm.

E. FedDyn

Our implementation of FedDyn makes use of 5 local epochs, which seemed to us to be the right compromise between performance and execution time. We initially tested it with the same parameters used with the other methods, however we obtained only invalid values for loss (NaN). We then tested a configuration of parameters proposed in the original paper: α=0.01 `lr`=0.1, `weight_decay`=1e-3, and `momentum`=0. This configuration allowed us to obtain valid values at least in part of the plot; however, we were not able to obtain similar

results as in the original paper, particularly in the non-i.i.d. case, as we can observe in Figures 5 and 16, Table VI.

Later, we figured that the proposed CNN in the original paper is much simpler than a ResNet50, in fact it has $\sim$ 30 times less parameters than the ResNet50 (23.5M vs 800K). We tried to perform a hyperparameter tuning on it, with the goal of finding a better set of parameters(Table V). This operation allowed us to choose a new set of parameters: α=0.1 `lr`=1e-2, `weight_decay`=5e-3, `momentum`=0.9, which led to results observable in Figure 15. By exploiting this configuration with the ResNet50, we obtained the results in Figures 5 and 17, Table VI. It can be seen that the configuration obtained by tuning on the simpler CNN was not effective on the ResNet with batch normalization layers, in fact we still obtained invalid loss results in almost all rounds. This was, however, very effective with the group normalization layers, obtaining an increase of almost 20% in i.i.d. and almost doubling our previous results in both non-i.i.d. cases.

TABLE V
HYPERPARAMETERS USED DURING GRID SEARCH TUNING FOR THE CNN

Hyperparameter	Values
α	[0.1, 1e-2, 1e-3]
`lr`	[0.1, 1e-2]
`weight_decay`	[5e-3, 1e-3, 5e-4]
`momentum`	[0, 0.5, 0.9]

TABLE VI
FEDDYN ACCURACY BY SCENARIO, NORMALIZATION AND HYPERPARAMETERS.

HP	Normalization	i.i.d.	non-i.i.d. bal.	non-i.i.d. unbal.
paper	BN	75.693%	10.033%	10.080%
	GN	35.880%	11.157%	11.227%
CNN tuned	BN	10.000%	10.001%	9.663%
	GN	53.570%	17.860%	14.897%

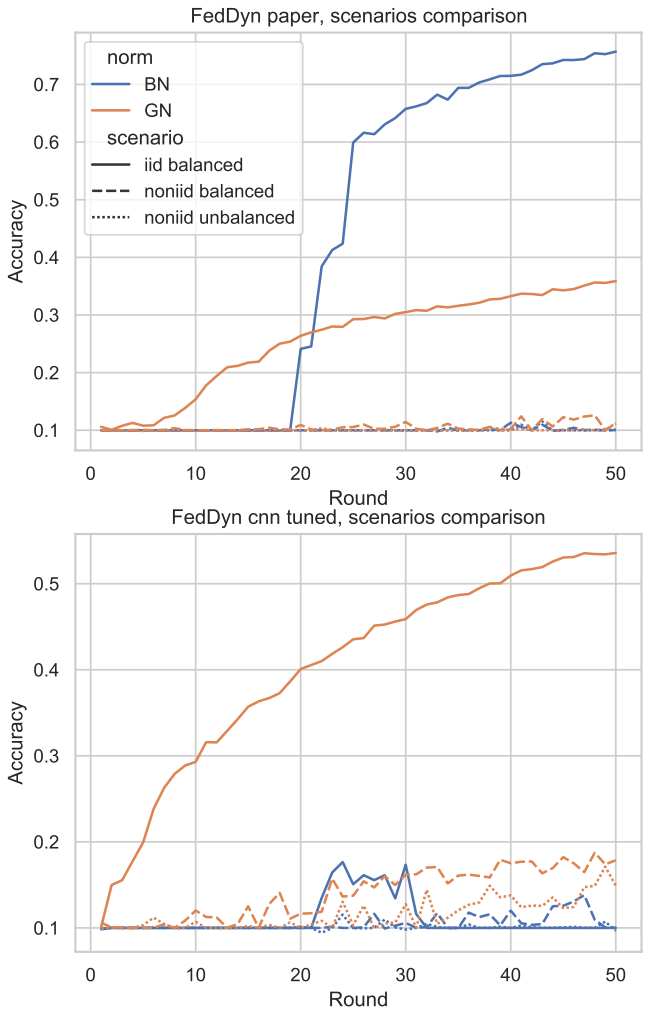

Fig. 5. Above: FedDyn using hyperparameters from the paper; Below: FedDyn after hyperparameters tuning

IV. CONSIDERATIONS

Why non-i.i.d. unbalanced results are better than balanced in FedAVG?

In the non-i.i.d unbalanced case, the results are better than the balanced counterpart. This result was already highlighted in the original paper. We can assume that this behavior is in part explained by the dataset splitting strategy proposed in the original paper: in the unbalanced setting, clients who receive more shards of data see on average more classes as well. So, since FedAVG aggregates results by a weighted average, it rewards more the contribution of clients with more data which can, on average, generalize better.

Letting FedAVG locally optimize more: closer to FedDyn?

For FedAVG experiments, we used only one local epoch. However, more local epochs could speed up learning, especially in the i.i.d. setting. On the other hand, for the non-i.i.d. setting we would risk overfitting the data we have locally and experience "client drift".

We experimented in the i.i.d. case with Batch Normalization using 5 local epochs, achieving up to 60.100% accuracy, meaning 4.883% increase compared to the result previously obtained with only one local epoch; Fig. 6.

This test is helpful to give a more fair comparison with FedDyn, by equalizing the amount of computation at the edge. We can observe that even though FedAVG improved, it remains below FedDyn by about 10% accuracy. From this, we deduce that FedDyn appears more effective at generalizing and aggregating individual clients' results.

Which loss should we use in FedGKT?

We compared cases of CE and CE+KD losses and obtained that our result aligns with the paper. The authors found that, in the server, using CE loss only performed better than CE+KD for smaller datasets, such as CIFAR-10, while CE+KD results were more helpful when dealing with more difficult datasets, such as CIFAR-100.

Some further observations about losses:

- Even if the CE loss brings improvements accuracy-wise, the loss' trends are much less "smooth" than with CE+KD loss (Fig. 11 and 12).
- We can see that, for CE loss, the non-i.i.d balanced case is the only case in which Group Normalization is better than Batch Normalization.
- The non-i.i.d balanced case is the only case in which CE+KD loss outperforms CE loss in the Batch Normalization case.

Is there a Group Normalization advantage in FedDyn?

Although we could not extensively test many parameter configurations, an interesting pattern emerged: the ResNets with Batch Normalization layers seem to have an unstable behavior with FedDyn. In fact, in at least 20 rounds, we obtained losses equal to NaN using the configuration proposed by the paper (which is the only one that reached valid results with Batch Normalization). At the same time, ResNets with Group Normalization layers seemed much more stable and did not result in any loss equal to NaN. Furthermore, using ResNets with Group Normalization allegedly allows performing tuning on simpler networks while achieving remarkable improvements on the ResNets themselves.

FedDyn: why non-i.i.d. cases systematically perform worse than results reported the paper?

In the non-i.i.d. cases, we could not obtain results comparable to those proposed in the original paper ([3]), even using the same model. To find a possible justification for this behavior, we tested the worst non-i.i.d. case sampling used in the original paper: Dirichlet(0.3). Analyzing the distribution of classes obtained by this type of sampling, we noticed that each client sees on average 8.15 classes (Fig. 14). With our sampling, instead, each client sees 4 times less classes (Fig.7). This remarkable difference allowed each client in the original paper to generalize much more effectively, thus providing that performance.

V. Conclusion

Looking at Figures 6 and 18, in which we compare all tested methods, we can conclude our journey with Federated Learning:

- In the i.i.d case, FedDyn and FedGKT obtain comparable results, much higher than all other methods. It would be interesting to perform an extensive comparative analysis on both to understand their capabilities.
- Our sampling is more difficult to work with than the Dirichlet sampling used in other papers. As explained, this is why our results are worse than the ones proposed by the authors of FedDyn. We think the community should settle on some standard sampling techniques to have comparable and also reproducible results.
- With great Resnets comes great responsibility. Heavier Resnets are obviously more powerful than simple CNNs. An interesting direction of research could be to understand if it is possible to tune hyperarameters on smaller networks and adapt the configuration to perform well on bigger networks. In this way, we could save time and obtain optimal results.
- The scenario in which we performed the gradient inversion attack is, in our opinion, very optimistic. Several conditions had to be met, such as knowing the batch size and the original labels. However, privacy is one of the founding elements of federated learning, so we think that newly proposed algorithms should at least consider the possibility of introducing security measures or perform an analysis of possible threats.

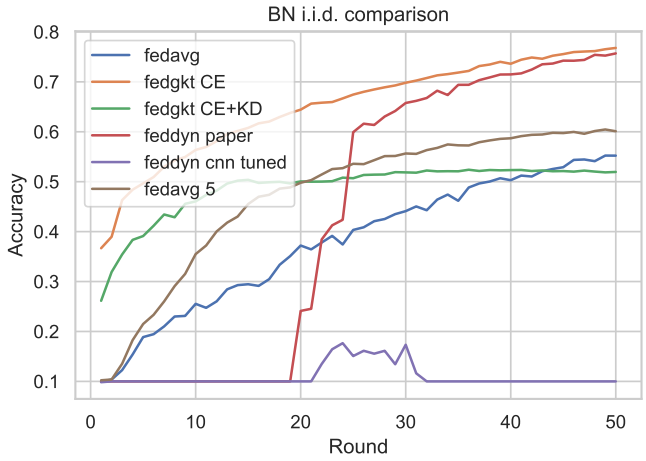

Fig. 6. Comparison between the runs using Batch Normalization in the i.i.d. case.

References

[1] H. Brendan McMahan et al. Communication-Efficient Learning of Deep Networks from Decentralized Data, Feb. 2016

[2] Chaoyang He et al. Group Knowledge Transfer: Federated Learning of Large CNNs at the Edge, Jul. 2020

[3] Durmus Alp Emre Acar et al. Federated Learning Based on Dynamic Regularization, Sep. 2020

[4] Sai Praneeth Karimireddy et al. SCAFFOLD: Stochastic Controlled Averaging for Federated Learning, Oct. 2019

[5] Tian Li et al. Federated Optimization in Heterogeneous Networks, Dec. 2018

[6] Umberto Michieli et al. Are All Users Treated Fairly in Federated Learning Systems?, Jun. 2021

[7] Praneeth Vepakomma et al. Split learning for health: Distributed deep learning without sharing raw patient data, Dec. 2018

[8] Jonas Geiping et al. Inverting Gradients - How easy is it to break privacy in federated learning?, Mar. 2020

[9] Yangsibo Huang et al. Evaluating Gradient Inversion Attacks and Defenses in Federated Learning, Dec. 2021

[10] Kaiming He et al. Deep Residual Learning for Image Recognition, Dec. 2015

[11] Sergey Ioffe et al. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, Feb. 2015

[12] Yuxin Wu et al. Group Normalization, Mar. 2018

APPENDIX

Fig. 7. Data distribution in the three cases using the sampling technique proposed in McMahan et al.

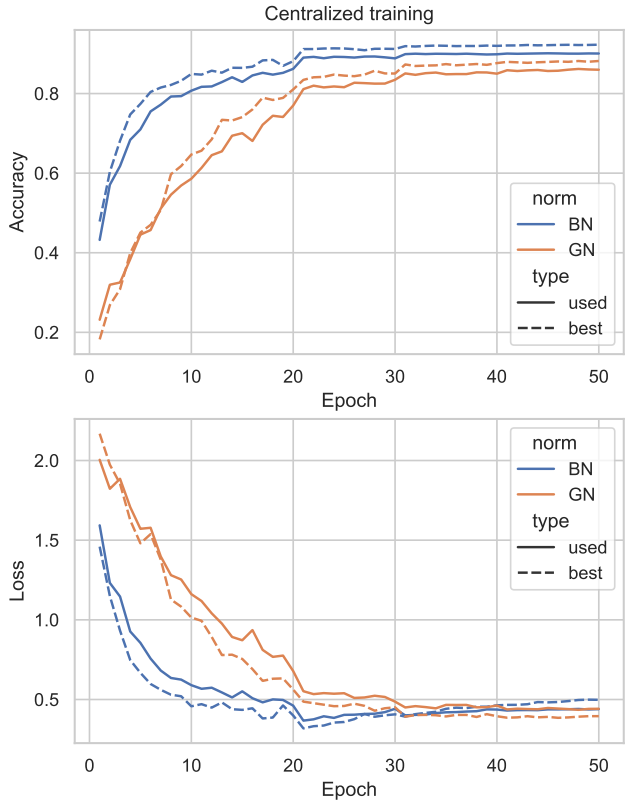

Fig. 8. Test accuracy and loss comparison between the best configuration and the one used for FedAVG in centralized training.

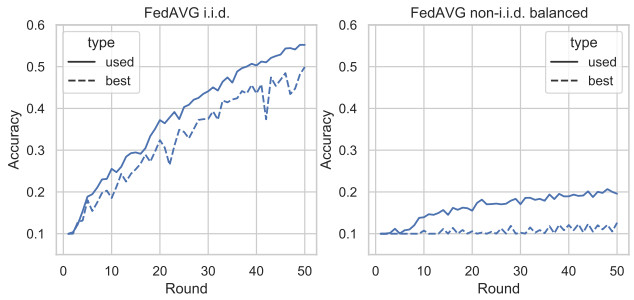

Fig. 9. Difference in accuracy between the 2 configurations. The one we switched to, shows higher accuracy and more stable trends.

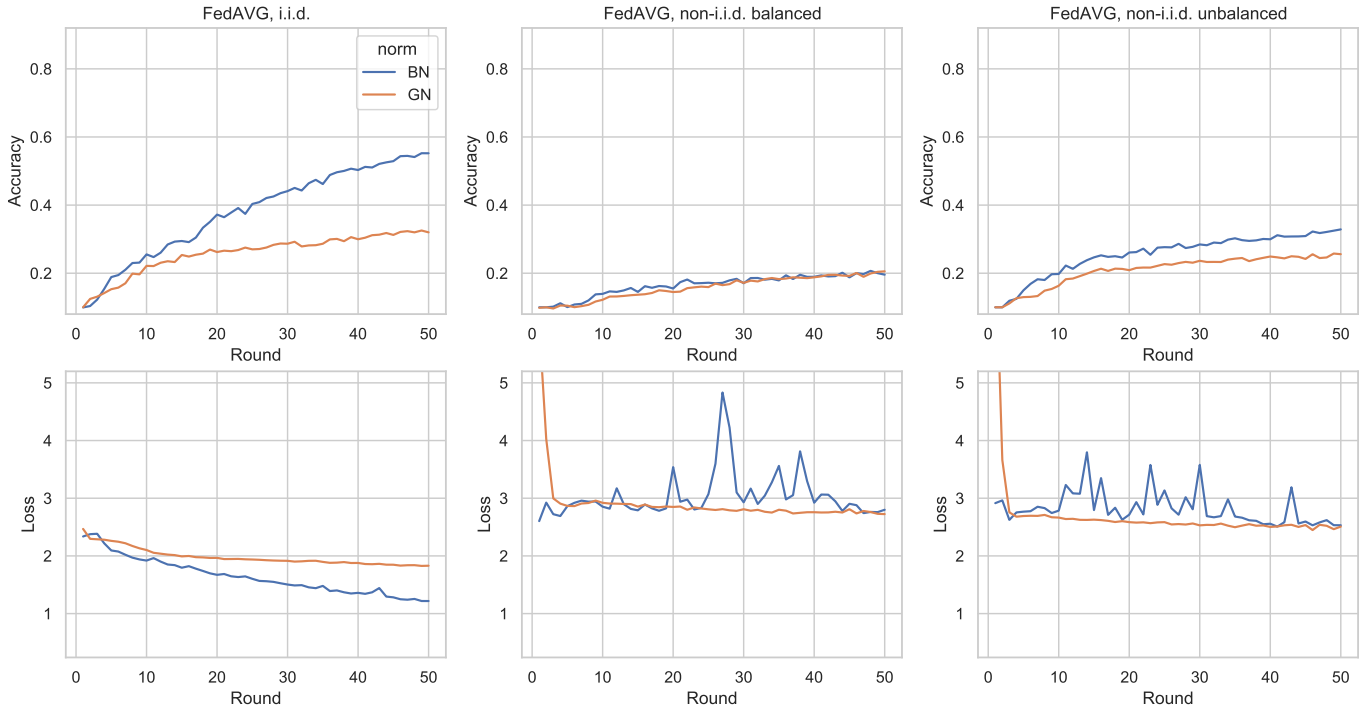

Fig. 10. Test accuracy and loss complete comparison of all the scenarios in FedAVG. The i.i.d. scenario is directly compared to the centralized training since they are the more similar.

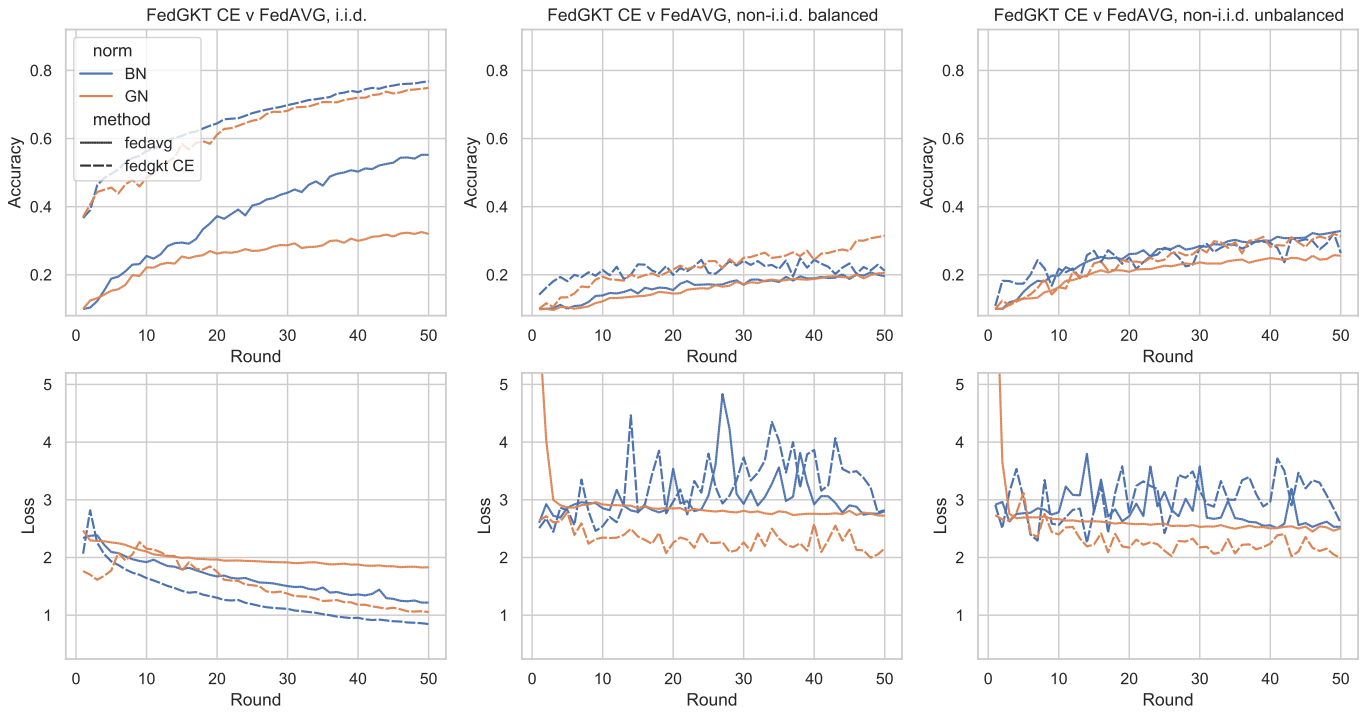

Fig. 11. Test accuracy and loss of all the scenarios in FedGKT using only CE loss in the server compared to FedAVG.

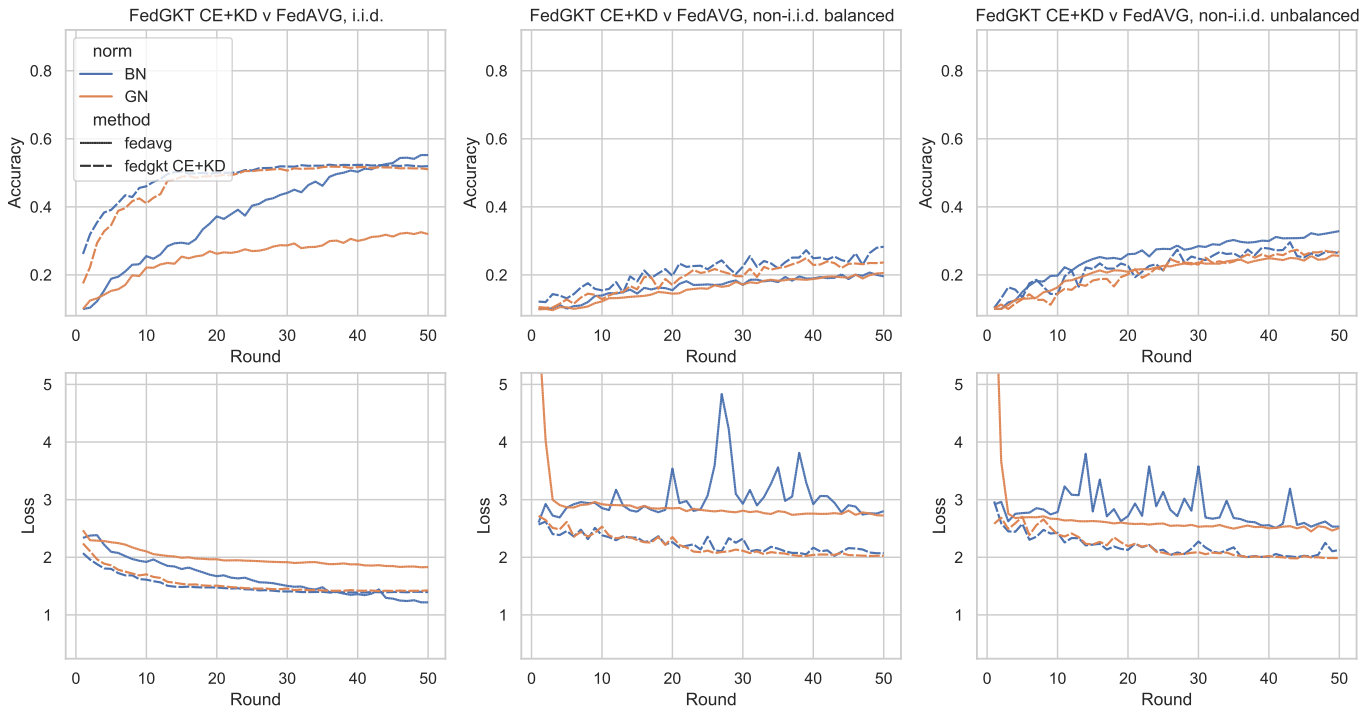

Fig. 12. Test accuracy and loss of all the scenarios in FedGKT using both CE and KD losses in the server compared to FedAVG.

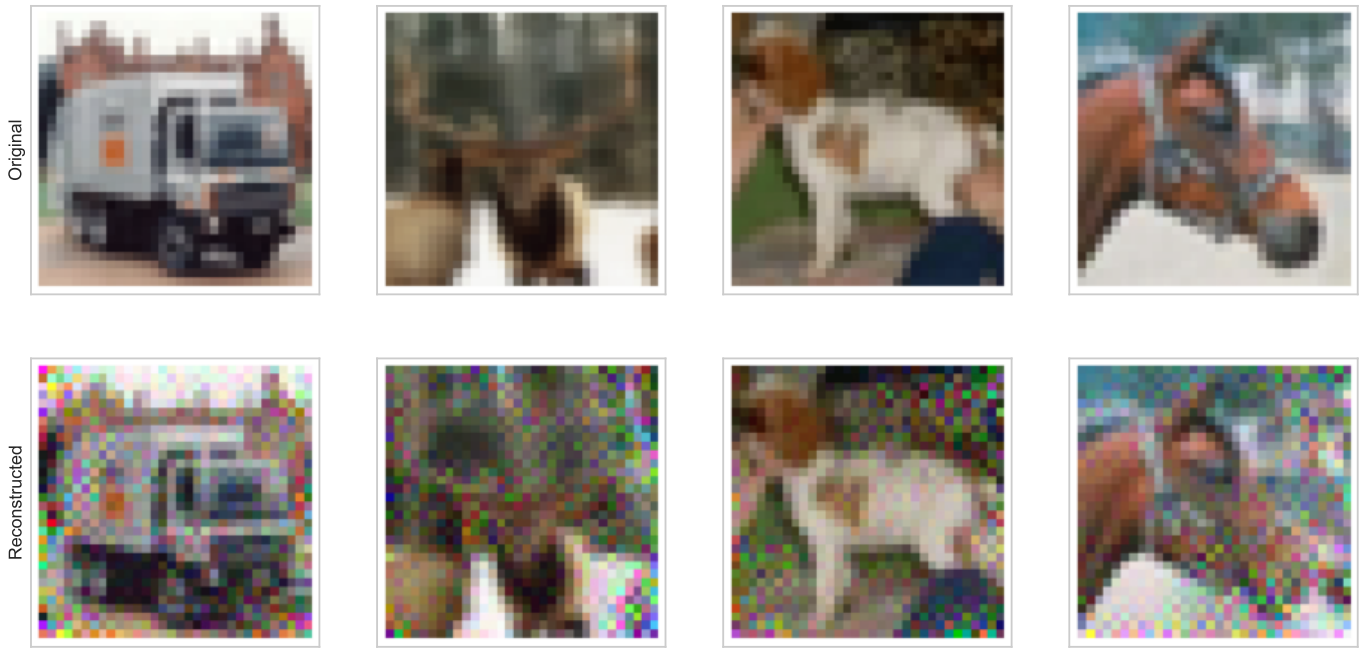

Fig. 13. Gradient attack results compared to the image that originated the gradient on which the algorithm worked. These results are the product of 3000 iterations of the algorithm.

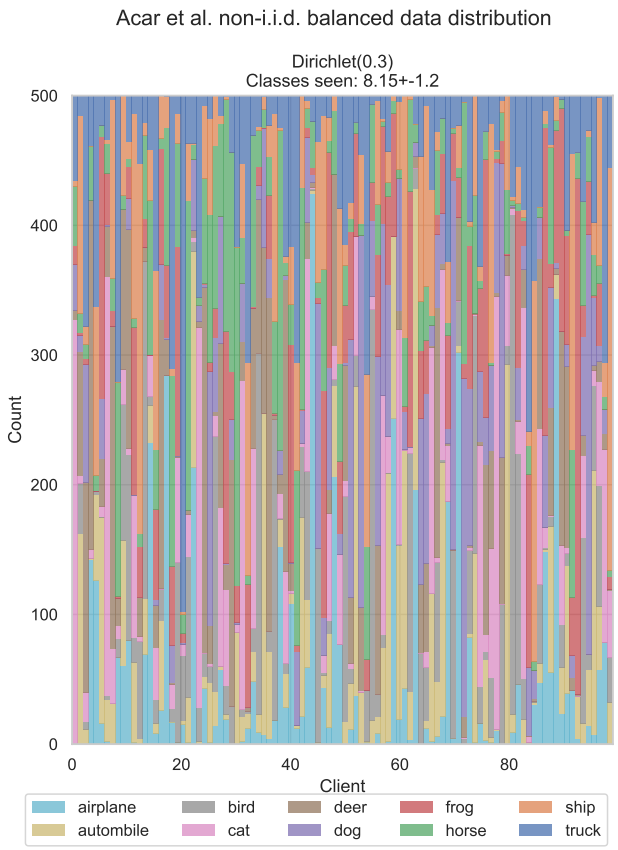

Fig. 14. Data sampling distribution used in the training step for FedDyn's original paper.

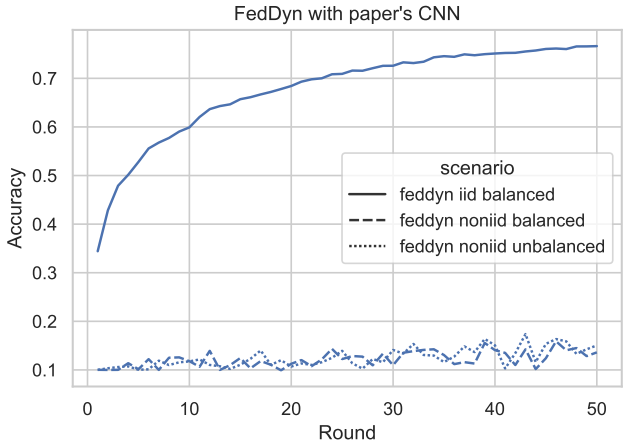

Fig. 15. FedDyn results using the same CNN proposed in the original paper.

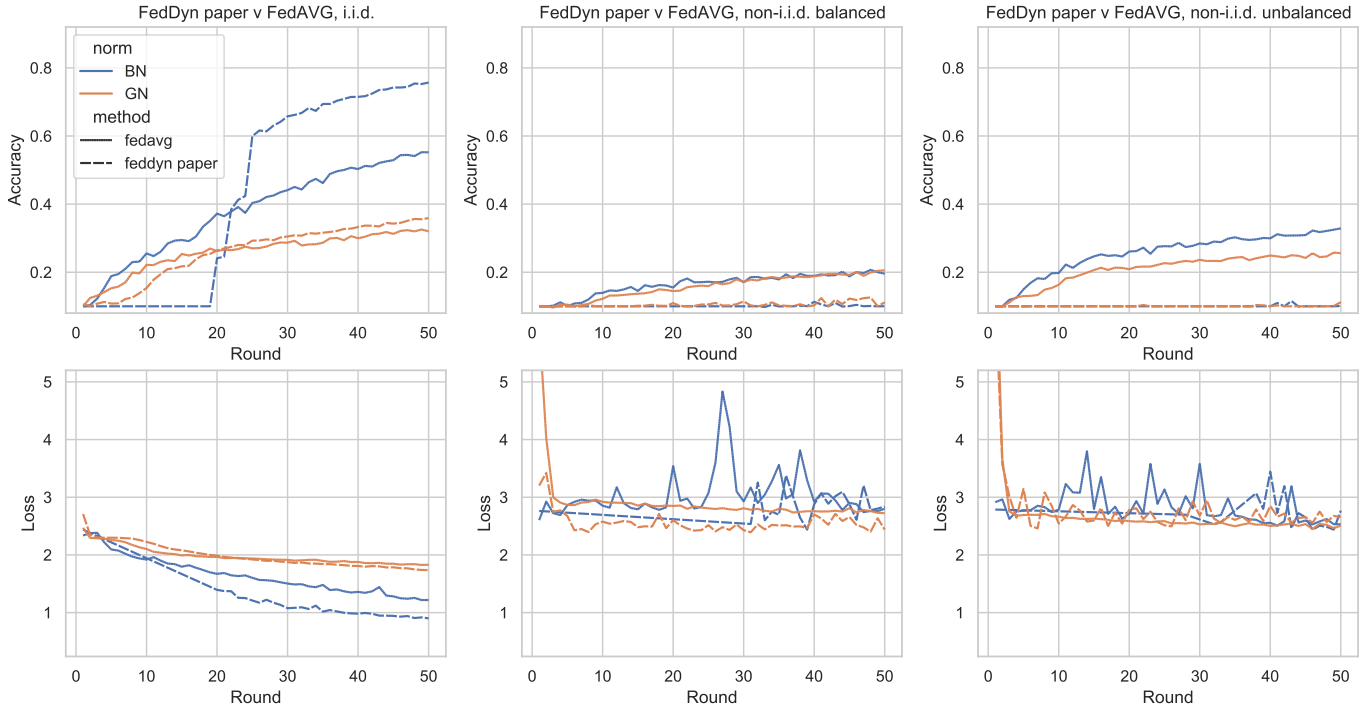

Fig. 16. Test accuracy and loss of all the scenarios in FedDyn, using the original paper's parameters, compared to FedAVG.

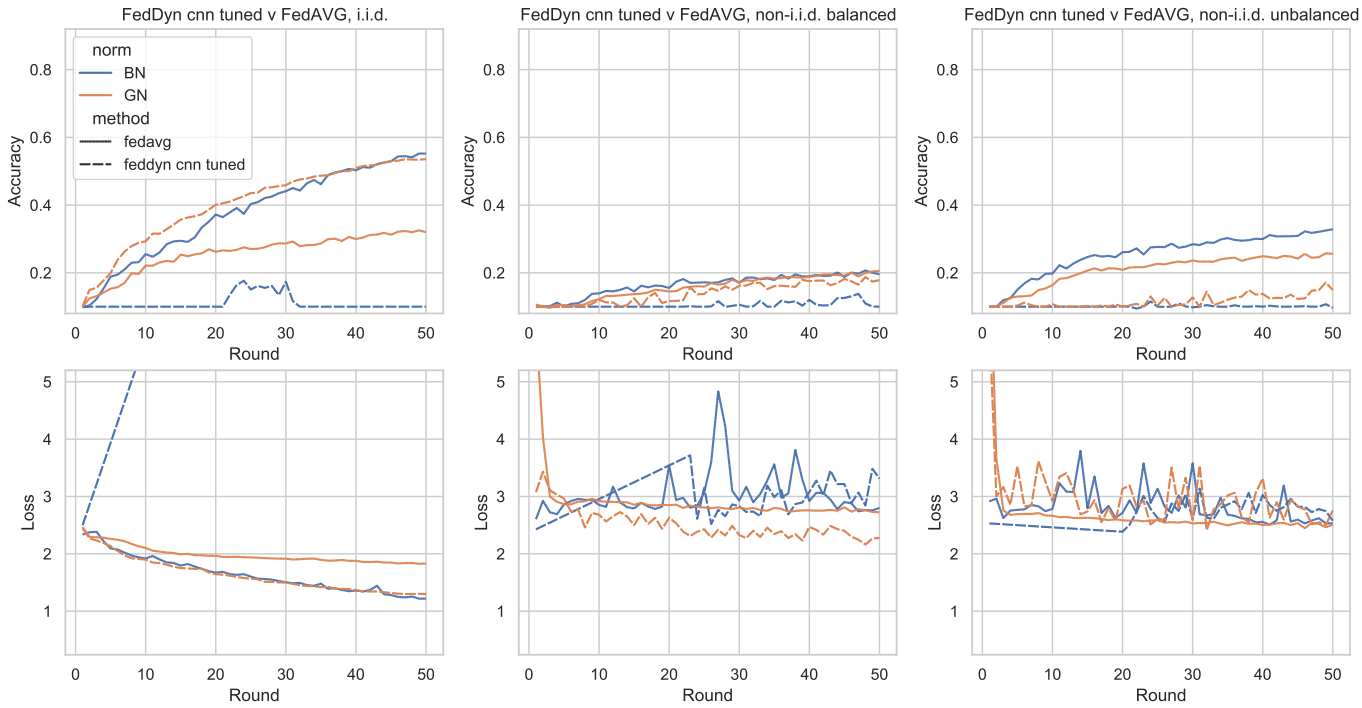

Fig. 17. Test accuracy and loss of all the scenarios in FedDyn, after hyperparameter tuning, compared to FedAVG.

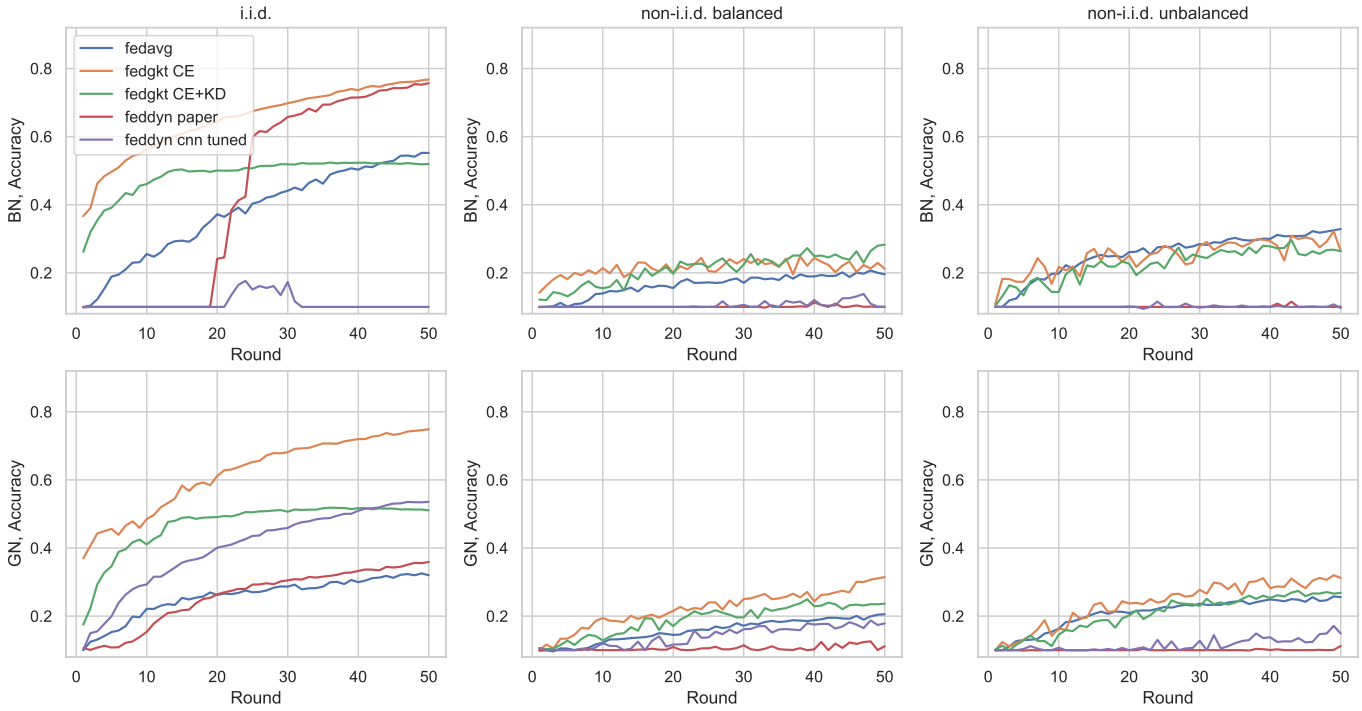

Fig. 18. Comparison of all tested methods in all scenarios with batch (above) and group (below) normalizations.