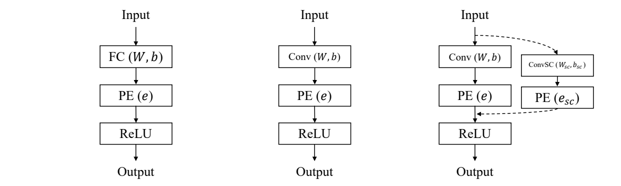
Federated Learning with Position-Aware Neurons - Supplementary

Xin-Chun Li[1], Yi-Chu Xu[1], Shaoming Song[2], Bingshuai Li[2], Yinchuan Li[2], Yunfeng Shao[2], De-Chuan Zhan[1]
[1]State Key Laboratory for Novel Software Technology, Nanjing University
[2]Huawei Noah's Ark Lab
{lixc, xuyc}@lamda.nju.edu.cn, zhandc@nju.edu.cn
{shaoming.song, libingshuai, liyinchuan, shaoyunfeng}@huawei.com

1. Dataset Details

The utilized datasets include Mnist [14], FeMnist [2], SVHN [19], GTSRB [23], Cifar10/100 [13], and Cinic10 [5]. We detail these datasets as follows.

- **Mnist** [14] is a digit recognition dataset that contains 10 digits to classify. The raw set contains 60,000 samples for training and 10,000 samples for evaluation. The image size is 28×28.
- **SVHN** [19] is the Street View House Number dataset which contains 10 numbers to classify. The raw set contains 73,257 samples for training and 26,032 samples for evaluation. The image size is 32×32.
- **GTSRB** [23] is the German Traffic Recognition Benchmark with 43 traffic signs. The raw set contains 39,209 samples for training and 12,630 samples for evaluation. We resize the images to 32×32.
- **Cifar10** and **Cifar100** [13] are subsets of the Tiny Images dataset and respectively have 10/100 classes to classify. They consist of 50,000 training images and 10,000 test images. The image size is 32×32.
- **Cinic10** [5] is a combination of Cifar10 and ImageNet [6], which contains 10 classes. It contains 90,000 samples for training, validation, and test, respectively. We do not use the validation set. The image size is 32×32.
- **FeMnist** [2] is built by partitioning the data in Extended MNIST [4] based on the writer of the digit/character. There are 62 digits and characters in all. The total number of training samples is 805,263. There are 3,550 users, and each user owns 226.8 samples on average. We only use 10% users (i.e., 355 users). For each user, we take 20% of the samples to construct the global test set. We resize the images to 28×28.



	MLP	VGG	ResNet
	$W \in R^{d' \times d}, b \in R^{d'}, e \in R^{d'}$	$W \in R^{C' \times C \times k_w \times k_h}$, $b \in R^{C'}, e \in R^{C'}$	$W \in R^{C' \times C \times k_w \times k_h}, b \in R^{C'}, e \in R^{C'}$ $W_{sc} \in R^{C' \times C \times 1 \times 1}, b \in R^{C'}, e \in R^{C'}$
PAN_+	ReLU$(Wx + b + e)$	ReLU(Conv$(x; W, b) + e$)	ReLU(Conv$(x; W, b) + e +$ Conv$(x; W_{sc}, b_{sc}) + e_{sc}$)
$PAN_\circ$	ReLU$((Wx + b) \circ e)$	ReLU(Conv$(x; W, b) \circ e$)	ReLU(Conv$(x; W, b) \circ e +$ Conv$(x; W_{sc}, b_{sc}) \circ e_{sc}$)

Figure 1. Network architectures with PANs. "PE" denotes position encoding; "SC" denotes shortcut. For ResNet, we only show one convolution layer in the basic block and omit the BatchNorm layers for simplification.

For centralized training, we correspondingly use the training set and test set for the first six datasets. For FeMnist, we centralize users' training samples as the training set. For decentralized training (i.e., FL), we split the training set of the first six datasets according to Dirichlet distributions as done in previous FL works [9, 17, 24]. Specifically, we split the training set onto K clients and each client's label distribution is generated from Dirichlet(α). While for FeMnist, we directly take the 355 users as clients. Some of these datasets are utilized in previous FL works. For example, Cifar10/Cifar100/Cinic10 are recommended by FedML [7], and FeMnist is recommended by LEAF [2].

2. Network Details

We utilize MLP, VGG [21], ResNet [8] in this paper. We detail their architectures as follows:

- **MLP** denotes a multiple layer perceptron with four layers containing input and output layers. For Mnist and FeMnist, the input size is $28 \times 28 = 784$. MLP has the architecture: FC1(784, 1024), ReLU(), FC2(1024,

Algorithm 1 Shuffle Process

```
1: Input: parameters $\{W_l, b_l\}_{l=1}^{L}$; shuffle probability $P_{\text{sf}}$
2: Generate-Permutation-Matrix: $\{\Pi_l\}_{l=1}^{L-1}$, $\Pi_{\{0,L\}} = I$
3: for each layer $l = 1, 2, \dots, L$ do
4:     $W_l \leftarrow \Pi_l W_l \Pi_{l-1}^T$, $b_l \leftarrow \Pi_l b_l$
5: end for
```

Generate-Permutation-Matrix

```
1: Input: number of neurons $J$; shuffle probability $P_{\text{sf}}$
2: Initialize: $\Pi = I^{J\times J}$
3: for $j = 1, 2, \dots, J$ do
4:     sample $i$ from Range$(j+1, J)$
5:     if $p \sim$ Uniform$(0,1) \leq P_{\text{sf}}$ then Swap$(\Pi_j, \Pi_i)$
6: end for
```

Algorithm 2 Shuffle Process in FL

```
1: Input: shuffle probability $P_{\text{sf}}$; expected shuffle times
   $N_{\text{sf}}$; number of local epochs $E$; batch size $B$; number
   of local samples $\{N_k\}_{k=1}^K$
2: for each client $k \in S_t$ do
3:     Calculate the number of local update steps: $r_k =$
       $E * N_k/B$
4:     for each local step in $[r_k]$ do
5:         if $p \sim$ Uniform$(0,1) \leq N_{\text{sf}}/r_k$ run the Shuf-
           fleProcess with shuffle probability $P_{\text{sf}}$
6:     end for
7: end for
```

1024), ReLU(), FC3(1024, 1024), ReLU(), FC4(1024, C). C denotes the number of classes.

- **VGG** contains a series of networks with various layers. The paper of VGG [21] presents VGG11, VGG13, VGG16, and VGG19. We follow their architectures and report the configuration of VGG11 as an example: 64, M, 128, M, 256, 256, M, 512, 512, M, 512, 512, M. "M" denotes the max-pooling layer. VGG11 contains 8 convolution blocks and three fully-connected layers in [21]. However, we only use one fully-connected layer for classification in this paper. VGG9 is commonly utilized in previous FL works [17, 24], whose configuration is: 32, 64, M, 128, 128, M, 256, 256, M. We keep all the fully-connected layers in VGG9 for a fair comparison with other works. The three fully-connected layers in VGG9 are: FC(4096, 512), ReLU(), FC(512, 512), ReLU(), FC(512, C). We name the ith convolution layer in VGG as "Conv$i$". We do not use BatchNorm [11] in VGG by default.

- **ResNet** introduces residual connections to plain neural networks. We take the Cifar versions used in the paper [8], i.e., ResNet20 with the basic block. We set the initial channel as 64 (i.e., the output channel of the first convolution layer), and take nine continual basic blocks with 64, 64, 64, 128, 128, 128, 256, 256, 256 channels, respectively. We add a fully-connected layer for classification. We use BatchNorm [11] in ResNet20 and add it before ReLU activation.

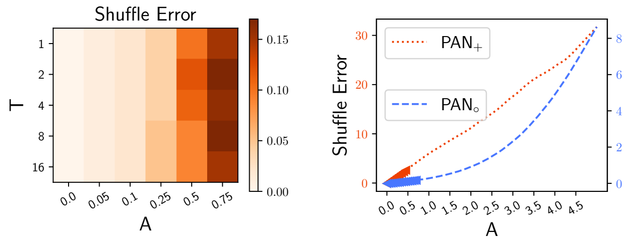


Figure 2. **Left**: shuffle error with various T and A (PAN$_\circ$). **Right**: the difference between PAN$_+$ and PAN$_\circ$ (T=1). (MLP)

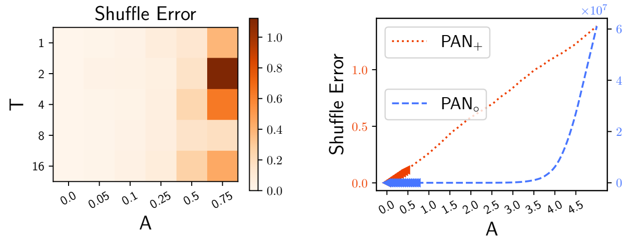


Figure 3. **Left**: shuffle error with various T and A (PAN$_\circ$). **Right**: the difference between PAN$_+$ and PAN$_\circ$ (T=1). (ResNet20)

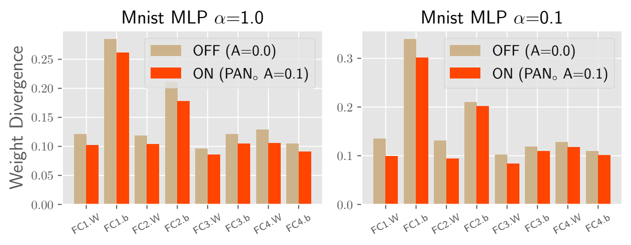


Figure 4. Weight divergence with PANs off/on. (E = 20, MLP on Mnist.)

For these networks with PANs, we plot the demos in Fig. 1. We add PE before the ReLU activation layer and after the BatchNorm layer. We show the formulations of additive PANs and multiplicative PANs in the table of Fig. 1.

3. Hyper-parameter Details

For both centralized training and decentralized training (i.e., FL), we take a constant learning rate without scheduling, although some works have pointed out decaying the learning rate will help in FL [3]. We take SGD with momentum 0.9 as the optimizer by default if without more declaration. For MLP and VGG networks, we set the learning

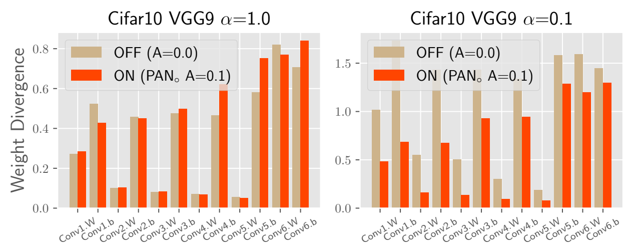


Figure 5. Weight divergence with PANs off/on. ($E = 5$, VGG9 on Cifar10.)

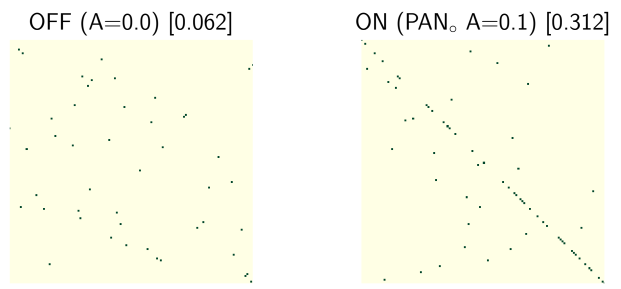


Figure 6. Optimal assignment matrix with PANs off/on, left vs. right. ($\alpha = 1.0$, $E = 20$, VGG9 Conv6 on Cifar10.)

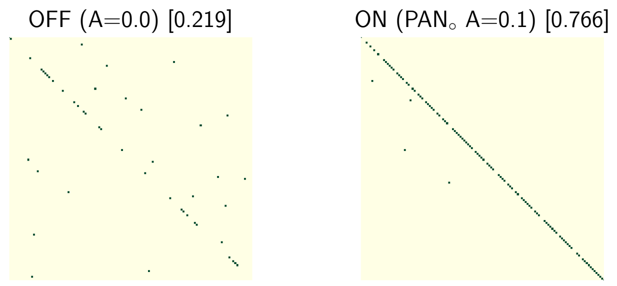


Figure 7. Optimal assignment matrix with PANs off/on, left vs. right. ($\alpha = 1.0$, $E = 20$, MLP FC3 on Mnist.)

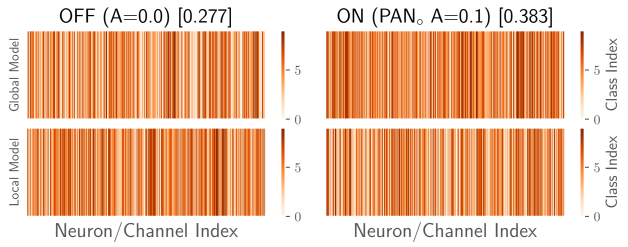


Figure 8. Preference vectors with PANs off/on, left vs. right. ($\alpha = 1.0$, VGG9 Conv5 on Cifar10.)

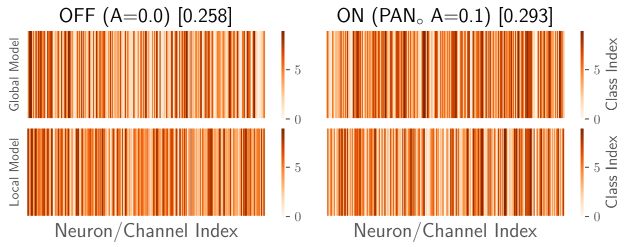


Figure 9. Preference vectors with PANs off/on, left vs. right. ($\alpha = 1.0$, VGG9 Conv4 on Cifar10.)

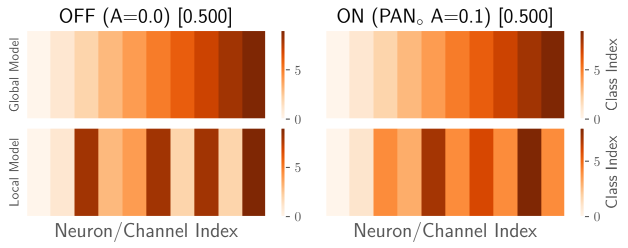


Figure 10. Preference vectors with PANs off/on, left vs. right. ($\alpha = 1.0$, MLP FC4 on Mnist.)

rate as 0.05; for ResNet, we use 0.1. We respectively use a warm start with 100 training steps and 10 training steps for centralized training and decentralized training (during local training). We use batch size 10 for FeMnist and 64 for other datasets.

We use FedAvg [18], FedProx [16], FedOpt [20], Scaffold [12], and MOON [15] as base FL algorithms. For all of these algorithms, we take H communication rounds, and select $R * 100.0\%$ clients during each round. Each client updates the global model on their private data for E epochs. For FedProx, the regularization coefficient of the proximal term is tuned in $\{1e-4, 1e-3\}$ and the best one is reported. For FedOpt, we take SGD with momentum 0.9 as the global optimizer, and tune the global learning rate in $\{0.1, 0.5, 0.9\}$, which is similar to FedAvgM [10]. We also try using Adam as the global optimizer and find the performances are not stable. For Scaffold, we use the implementation from the online page [1]. For MOON, we set the coefficient of the contrastive loss as 1.0, which is recommended by the authors. We then replace the normal neurons with the proposed PANs to improve these algorithms. We keep $T = 1$ by default and tune hyper-parameters from: PAN$_+$ with $A = 0.05$, PAN$_\circ$ with $A = 0.05$, PAN$_\circ$ with $A = 0.1$.

[1] https://github.com/ramshi236/Accelerated-Federated-Learning-Over-MAC-in-Heterogeneous-Networks

	FeMnist MLP	GTSRB VGG9	SVHN VGG9	Cifar10 VGG11	Cifar100 ResNet20	Cinic10 ResNet20
SGD + Momentum=0.9 (LR in $\{0.05,0.1\}$)	53.39	86.96	89.93	84.57	70.82	82.76
Adam (LR=3e-4)	54.25	90.84	91.13	87.13	67.22	81.99

Table 1. The performances of centralized training with corresponding networks (without PANs), i.e., the **upper bound of decentralized training** (FL).

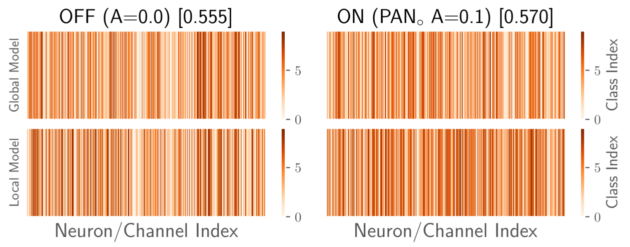


Figure 11. Preference vectors with PANs off/on, left vs. right. ($\alpha = 1.0$, MLP FC3 on Mnist.)

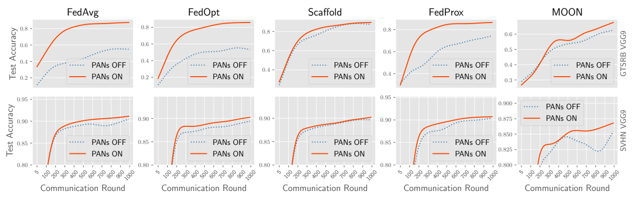


Figure 12. Comparison results on non-i.i.d. data (α=0.1). Rows show datasets and columns show FL algorithms. PANs could universally improve these algorithms.

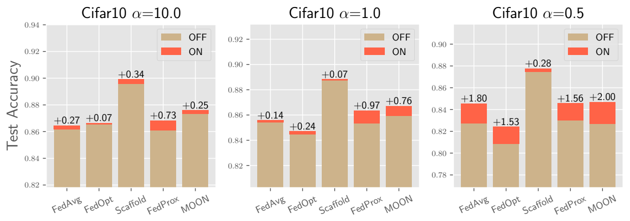


Figure 13. Comparisons under various levels of non-i.i.d. data on Cifar10. Smaller α implies more non-i.i.d. data.

4. Experimental Details

4.1. Shuffle Test and Shuffle Test in FL

We propose a procedure to measure the degree of permutation invariance of a certain neural network, that is, how large the shuffle error is after shuffling the neurons. The shuffle process is shown in Alg. 1, where P_{sf} controls the disorder level of the constructed permutation matrices. Some additional descriptions are: (1) the permutation matrix (PM) should be randomly generated and we don't need

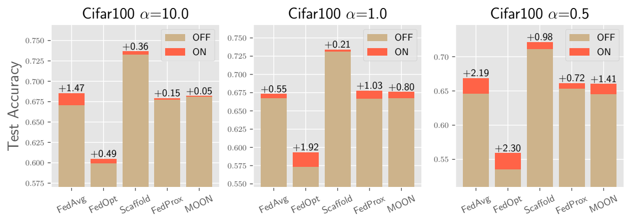


Figure 14. Comparisons under various levels of non-i.i.d. data on Cifar100. Smaller α implies more non-i.i.d. data.

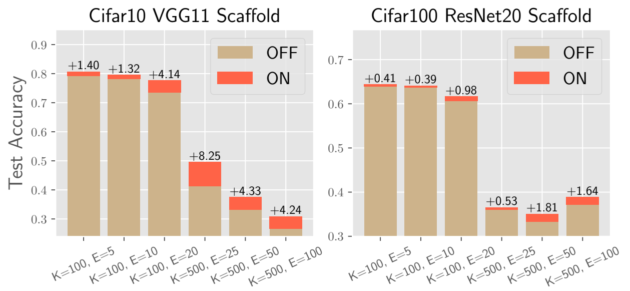


Figure 15. Comparisons under different FL scenes (K, E) based on Scaffold.

to solve it; (2) PMs are introduced just to verify the property of PANs that they can disable the permutation invariance of neural networks, which is not used in our FedPAN algorithm; (3) the computational complexity is $\mathcal{O}(J)$, requiring at most J swaps, which is very efficient to implement during simulation.

We introduce the shuffle test in the body of this paper. Specifically, we manually shuffle the network and study the output change, i.e., the shuffle error defined in the body. A hyper-parameter P_{sf} is used to control the disorder of permutation. Given a P_{sf}, we could generate a permutation matrix Π, then we calculate how many neurons are not shuffled via computing "R_{kept}=np.mean(np.diag(Π))". We use the functions provided in the Numpy [2] package. This is calculated and its correspondence to P_{sf} is shown in the body. The shuffle process is also applied to FL. Specifically, we present the Pseudo-Code in Alg. 2. Easily, the model will be shuffled for N_{sf} times during local training in

[2]https://numpy.org/

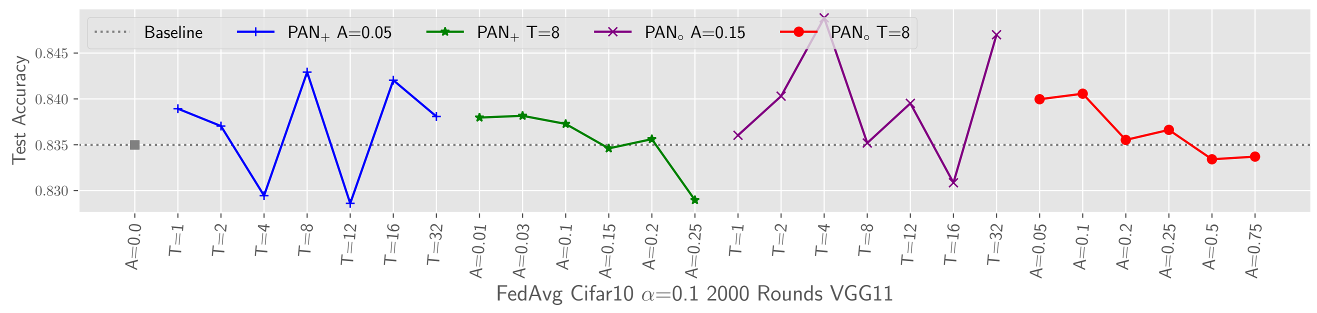


Figure 16. Hyper-parameter analysis on Cifar10 with VGG11.

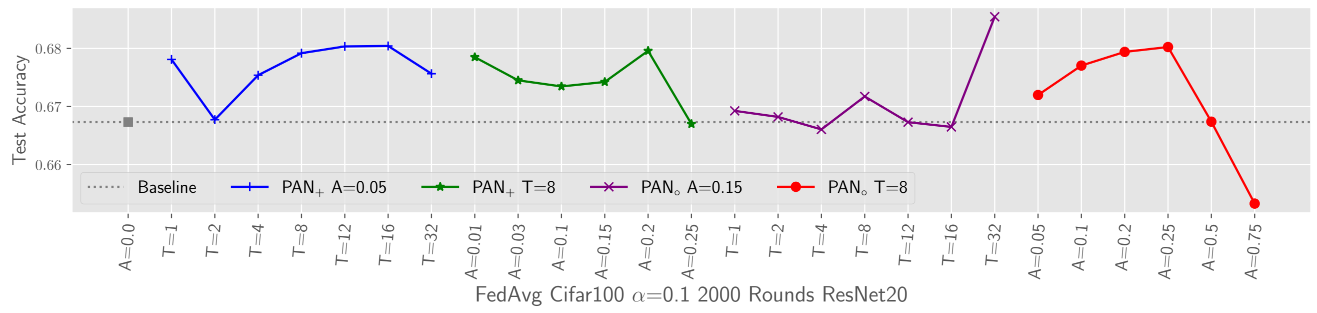


Figure 17. Hyper-parameter analysis on Cifar100 with ResNet20.

expectation. Hence, we calculate the corresponding R_{kept} as the diagonal ones after several accumulative permutation, i.e., "R_{kept}=np.mean(np.diag($\Pi_{r_k} \cdots \Pi_2\Pi_1$))", where Π_1, Π_2, and Π_{r_k} denote the generated permutation matrices in each local update step. We simulate the process for a single layer 10 times and calculate the averaged R_{kept}. We keep $P_{\text{sf}} = 0.1$ and show the relations of R_{kept} and N_{sf} in the body.

5. Additional Experimental Results

Shuffle Error on Random Data: We investigate the shuffle error via taking the random data as input in the body, where we only present the results based on VGG13. We report similar results on MLP and ResNet20, which are shown in Fig. 2 and Fig. 3. Multiplicative PANs with a larger A make the network more sensitive to neuron permutation.

Weight Divergence: Our proposed PANs could decrease the weight divergence during FL. Specifically, we split the training data onto $K = 10$ clients with $\alpha \in \{1.0, 0.1\}$ and select all clients in each round, i.e., $R = 1.0$. We take $H = 20$ communication rounds and then calculate the local gradient variance as an approximation. We vary the number of local epochs $E \in \{5, 20\}$. We only report the results on Mnist with $E = 5$ in the body. Additional results of Mnist with $E = 20$ (Fig. 4) and Cifar10 with $E = 5$ (Fig. 5) further verify that PANs could decrease the local gradient variance.

Matching via Optimal Assignment: We first train a global model via FL for $H = 20$ communication rounds, where the scene contains 10 clients with $\alpha = 1.0$. Then, we randomly sample a local client and update the global model for E epochs. Our goal is to search for a matrix to match the neurons of the global model and the updated one, i.e., the local model of this client. We then use 500 test samples to obtain the neuron's activations as their representations. Hence, the optimal assignment problem could be solved and the assignment matrix is a permutation matrix. The results on various layers of VGG9 and MLP are shown in Fig. 6 and Fig. 7. *Notably, the calculated matching ratio, i.e., the number in "[]", is only an approximated value which represents how much neurons are shuffled. The absolute value (e.g., 0.062) does not represent the actual permutation during training.*

Visualizing Neurons via Preference Vectors: Similarly, more of the visualization results via preference vectors of neurons are provided in Fig. 8, Fig. 9, Fig. 10, and Fig. 11. Notably, there are only 10 neurons in Fig. 10 because FC4 is the output layer with 10 classes. Using PANs could encourage neurons at the same position contribute to the same classes as much as possible.

Universal Application of PANs: We report the results of applying PANs to popular FL algorithms on FeMnist, Cifar10, Cifar100, and Cinic10 in the body. We show the results on SVHN and GTSRB in Fig. 12. Training on GTSRB is not stable, and some algorithms will converge slower,

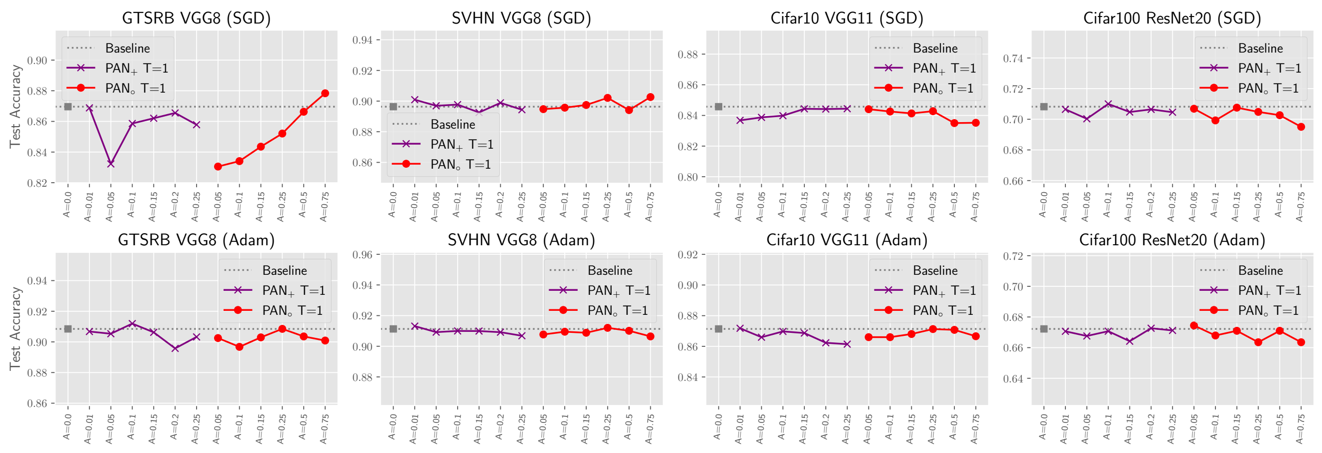


Figure 18. Performances of centralized training with PANs. The two parts respectively show the results of additive PANs and multiplicative PANs.

e.g., FedAvg and FedOpt. This could be improved with the additional effort of tuning learning rates, while we omit this in this paper. Comparison results on Cifar10 and Cifar100 under various levels of non-i.i.d. data are shown in Fig. 13 and Fig. 14. The improvements under various scenes based on Scaffold are shown in Fig. 15. These additional results further verify the universal application of PANs to improve the performance of FL.

Hyper-parameter Analysis: We present the performances of various A with PAN$_\circ$ when $T = 1$ in the body and point out that setting $A = 0.1$ is a good choice. Here, we present a more comprehensive analysis with both additive and multiplicative PANs. The used FL scene is: $K = 100$, $\alpha = 0.1$, $H = 2000$, $R = 0.1$, $E = 5$. We plot the results on Cifar10 with VGG11 and Cifar100 with ResNet20 in Fig 16 and Fig. 17. The leftmost point shows the baseline of the performance. The four parts in different colors show the results with various T or A, while the other one is fixed. For example, the first part shows the performances with $T \in \{1, 2, 4, 8, 12, 16, 32\}$ in PAN$_+$, while A is fixed to 0.05. Clearly, with fixed T, a larger A leads to degradation (the green and the red part). Setting A around 0.1 for PAN$_\circ$ is recommended. The results on Cifar100 are more invariant to T, although the performances fluctuate a lot on Cifar10. Many of these hyper-parameters could surpass the baseline.

6. More Studies

6.1. Centralized Training

We report the test accuracies of centralized training on FeMnist, GTSRB, SVHN, Cifar10, Cifar100, and Cinic10. The utilized networks are correspondingly MLP, VGG9, VGG9, VGG11, ResNet20, and ResNet20. The numbers of training epochs are respectively 30, 20, 30, 30, 100, and

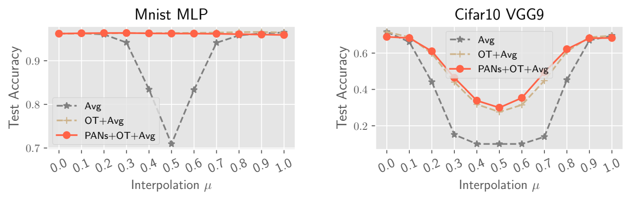


Figure 19. Model fusion of MLP on Mnist (Left) and VGG9 on Cifar10 (Right) with direct parameter averaging, optimal transport, and PANs. The x-axis shows the interpolation coefficient.

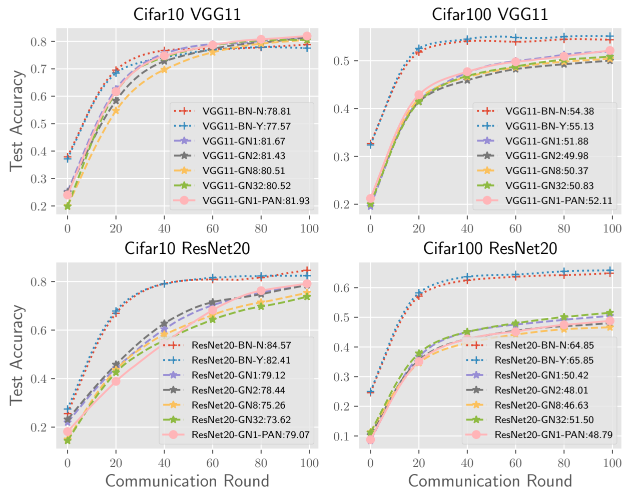


Figure 20. Comparisons of different normalization techniques in ConvNet. The top is based on VGG11 and the bottom is based on ResNet20. We use datasets Cifar10 and Cifar100.

100. We utilize both SGD with momentum 0.9 and Adam as the optimizer. For SGD, we use 0.05 as the learning rate for MLP and VGG, while 0.1 for ResNet20. For Adam, we

use 0.0003 for all networks. The performances are listed in Tab. 1. We then add PANs to some datasets and find that the performances degrade slightly. We vary the hyper-parameter A in PANs while keeping $T = 1$. The results are shown in Fig. 18. Using PANs could harm the training process slightly, and commonly, a larger A could make the results worse. Although we try utilizing the adaptive optimizer (i.e., Adam), the results of utilizing PANs do not improve. Advanced optimizers should be proposed to mitigate the degradation, which is left for future work.

6.2. Optimal Transport for Model Fusion

FL should send down the global model to local clients as the initialization during each communication round. If not, coordinate-based parameter averaging will become worse. The work [22] studies model fusion with different initializations, and utilizes optimal transport [1] to align model parameters. We split Mnist and Cifar10 into two parts uniformly. We train independent models on these two sets correspondingly. The obtained models after training 20 epochs are denoted as θ_A and θ_B. Then, an interpolation is evaluated, i.e., $(1 - \mu)\theta_A + \mu\theta_B$, $\mu \in [0, 1]$. Directly averaging these two models will perform poorly, which is shown in Fig 19 (the line with legend "Avg"). If we align the models via optimal transport and then interpolate the aligned models, the results become better (the line with legend "OT+Avg" in Fig 19). We further add PANs during model training and the performances could be slightly improved (the line with legend "PANs+OT+Avg" in Fig 19). This shows that PANs may still be helpful with different initializations.

6.3. BatchNorm vs. GroupNorm

We then investigate the normalization techniques in deep neural networks. Previous FL works point out that GroupNorm may be more applicable to FL with non-iid data [9]. Specifically, BatchNorm calculates the mean and variance of a data batch, which is relevant to local training data. Hence, the statistical information in BatchNorm will diverge a lot across clients. One solution is aggregating the statistical information during FL, i.e., averaging the "running mean" and "running variance" in BatchNorm. We denote this as "BN-Y". In contrast, we use "BY-N" to represent the method that "running mean" and "running variance" are not aggregated. We also vary the number of groups in GroupNorm, i.e., $\{1, 2, 8, 32\}$, which are denoted as "GN1", "GN2", "GN8", and "GN32". We list the convergence curves on Cifar10 and Cifar100 in Fig. 20. We use VGG11 and ResNet20 as the backbone. The numbers in the legends denote the final test accuracies. GroupNorm only improves the performances of Cifar10 with VGG11. Additionally, setting the number of groups as 1 is better. We also apply PANs to networks with "GN1" and find the performance does not improve. The combination of PANs with various normalization techniques is also interesting, which is also left for future work.

6.4. Personalization in FL

Finally, we present some possible varieties of PANs for personalization in FL. In the body of this paper, we take the same position encodings among clients and implicitly make neurons combined with their positions. However, if we take different position encodings or partially shared position encodings among clients, we could let similar clients contribute more. Some clients own individual positions, which could be utilized for personalization. These ideas are also left for future work.

References

[1] Martial Agueh and Guillaume Carlier. Barycenters in the wasserstein space. *SIMA*, 43(2):904–924, 2011. 7

[2] Sebastian Caldas, Peter Wu, Tian Li, Jakub Konecný, H. Brendan McMahan, Virginia Smith, and Ameet Talwalkar. LEAF: A benchmark for federated settings. *CoRR*, abs/1812.01097, 2018. 1

[3] Giulia Fanti Charlie Hou, Kiran Thekumparampil and Sewoong Oh. Multistage stepsize schedule in federated learning: Bridging theory and practice. In *ICML Workshop*, 2021. 2

[4] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and André van Schaik. EMNIST: an extension of MNIST to handwritten letters. *CoRR*, abs/1702.05373, 2017. 1

[5] Luke Nicholas Darlow, Elliot J. Crowley, Antreas Antoniou, and Amos J. Storkey. CINIC-10 is not imagenet or CIFAR-10. *CoRR*, abs/1810.03505, 2018. 1

[6] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, pages 248–255, 2009. 1

[7] Chaoyang He, Songze Li, Jinhyun So, Mi Zhang, Hongyi Wang, Xiaoyang Wang, Praneeth Vepakomma, Abhishek Singh, Hang Qiu, Li Shen, Peilin Zhao, Yan Kang, Yang Liu, Ramesh Raskar, Qiang Yang, Murali Annavaram, and Salman Avestimehr. Fedml: A research library and benchmark for federated machine learning. *CoRR*, abs/2007.13518, 2020. 1

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016. 1, 2

[9] Kevin Hsieh, Amar Phanishayee, Onur Mutlu, and Phillip B. Gibbons. The non-iid data quagmire of decentralized machine learning. In *ICML*, pages 4387–4398, 2020. 1, 7

[10] Tzu-Ming Harry Hsu, Hang Qi, and Matthew Brown. Measuring the effects of non-identical data distribution for federated visual classification. *CoRR*, abs/1909.06335, 2019. 3

[11] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, pages 448–456, 2015. 2

[12] Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank J. Reddi, Sebastian U. Stich, and Ananda Theertha Suresh. SCAFFOLD: stochastic controlled averaging for federated learning. In *ICML*, pages 5132–5143, 2020. 3

[13] Alex Krizhevsky. Learning multiple layers of features from tiny images. 2012. 1

[14] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 1

[15] Qinbin Li, Bingsheng He, and Dawn Song. Model-contrastive federated learning. In *CVPR*, pages 10713–10722, 2021. 3

[16] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. In *MLSys*, 2020. 3

[17] Tao Lin, Lingjing Kong, Sebastian U. Stich, and Martin Jaggi. Ensemble distillation for robust model fusion in federated learning. In *NeurIPS*, 2020. 1, 2

[18] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *AISTATS*, pages 1273–1282, 2017. 3

[19] Yuval Netzer, Tiejie Wang, Adam Coates, A. Bissacco, Bo Wu, and A. Ng. Reading digits in natural images with unsupervised feature learning. 2011. 1

[20] Sashank J. Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečný, Sanjiv Kumar, and Hugh Brendan McMahan. Adaptive federated optimization. In *ICLR*, 2021. 3

[21] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015. 1, 2

[22] Sidak Pal Singh and Martin Jaggi. Model fusion via optimal transport. In *NeurIPS*, 2020. 7

[23] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. The german traffic sign recognition benchmark: A multi-class classification competition. In *IJCNN*, pages 1453–1460, 2011. 1

[24] Hongyi Wang, Mikhail Yurochkin, Yuekai Sun, Dimitris S. Papailiopoulos, and Yasaman Khazaeni. Federated learning with matched averaging. In *ICLR*, 2020. 1, 2