# Federated Learning with Position-Aware Neurons

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

## Abstract

*Federated Learning (FL) fuses collaborative models from local nodes without centralizing users' data. The permutation invariance property of neural networks and the non-i.i.d. data across clients make the locally updated parameters imprecisely aligned, disabling the coordinate-based parameter averaging. Traditional neurons do not explicitly consider position information. Hence, we propose Position-Aware Neurons (PANs) as an alternative, fusing position-related values (i.e., position encodings) into neuron outputs. PANs couple themselves to their positions and minimize the possibility of dislocation, even updating on heterogeneous data. We turn on/off PANs to disable/enable the permutation invariance property of neural networks. PANs are tightly coupled with positions when applied to FL, making parameters across clients pre-aligned and facilitating coordinate-based parameter averaging. PANs are algorithm-agnostic and could universally improve existing FL algorithms. Furthermore, "FL with PANs" is simple to implement and computationally friendly.*

## 1. Introduction

Federated Learning (FL) [13, 42] generates a global model via collaborating with isolated clients for privacy protection and efficient distributed training, generally following the parameter server architecture [6, 21]. Clients update models on their devices using private data, and the server periodically averages these models for multiple communication rounds [27]. The whole process does not transmit users' data and meets the basic privacy requirements.

Represented by FedAvg [27], many FL algorithms aggregate local parameters via a simple coordinate-based averaging [22–25] These algorithms have two kinds of drawbacks. First, as traditional neurons are unaware of their positions, neural networks have the permutation invariance property, implying that hidden neurons could be dislocated during training without affecting the local performances. Second, the samples across clients are non-independent and identically distributed (non-i.i.d.) [11], which could exacerbate the permutation of neural networks during local training, making local models misaligned and leading to weight divergence [47]. These reasons degrade the performance of coordinate-based parameter averaging.

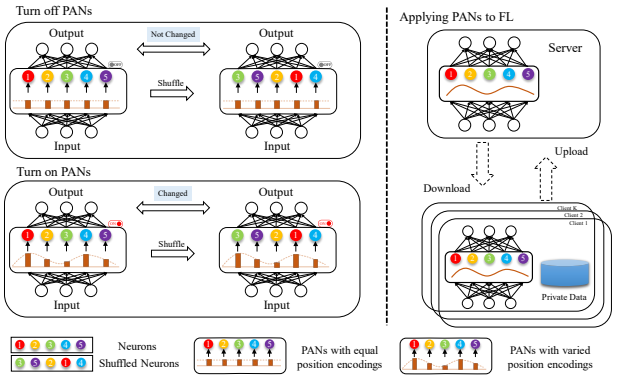


Figure 1. **Left**: Position-Aware Neurons (PANs). We fuse equal/varied position encodings to neurons' outputs, PANs are turned off/on, and the shuffled networks make the same/different predictions, i.e., the permutation invariance property is enabled/disabled. **Right**: applying PANs to FL. Neurons are coupled with their positions for pre-alignment.

Recently, a series of works utilize various matching techniques to align neurons, such as Bayesian nonparametric learning [38, 44, 45] and optimal transport [2, 33]. First, these methods are too complex to implement. Second, they solve the misalignment problem after finishing local updates and hence belong to post-processing strategies that need additional computation budgets. Fed$^2$ [43] pioneers a novel aspect via designing feature-oriented model structures following a pre-aligned manner. However, it has to carefully customize the network architecture and only stays

at the group level of pre-alignment. By contrast, we explore a more straightforward and general technique to pre-align neurons during local training procedures.

Our work mainly focuses on solving the non-i.i.d. challenge in FL, more specifically, seeking solutions via limiting the permutation invariance property of neural networks. We first summarize the above analysis: *the permutation invariance property of neural networks leads to neuron misalignment across local models. The more heterogeneous the data, the more serious the misalignment is*. Hence, our motivation is intuitive: *could we design a switch to control the permutation invariance property of neuron networks?* We propose **Position-Aware Neurons (PANs)** as the solution, which couple neurons with their positions. Specifically, for each neuron (channel for ConvNet [10, 17, 32]), we add or multiply a position-related value (i.e., position encoding) to its output. We introduce a hyper-parameter to turn on/off the PANs, and correspondingly, to disable/enable the permutation invariance property of neural networks. PANs bind neurons in their positions, implicitly pre-aligning neurons across clients even faced with non-i.i.d. data. From another aspect, PANs could keep some consistent ingredients in the forward and backward pass across local models, which could reduce the weight divergence. Overall, appropriate PANs facilitate the coordinate-based parameter averaging in FL. Replacing traditional neurons with PANs is simple to implement and computationally friendly, which is universal to various FL algorithms. Contributions can be briefed as: *(1) proposing PANs to disable/enable the permutation invariance property of deep networks; (2) applying PANs to FL, which binds neurons in positions and pre-aligns parameters for better coordinate-wise parameter averaging.*

## 2. Related Works

**FL with Non-I.I.D. Data:** Existing works solve the non-i.i.d. data problem in FL from various aspects. [47] points out the weight divergence phenomenon in FL and use shared data to decrease the divergence. FedProx [23] takes a proximal term during local training as regularization. FedOpt [30] considers updating the global model via momentum or adaptive optimizers (e.g., Adam [15], Yogi [46]) instead of simple parameter averaging. Scaffold [14] introduces control variates to rectify the local update directions and mitigates the influences of client drift. MOON [22] utilizes model contrastive learning to reduce the distance between local and global models. Some other works utilize similar techniques including dynamic regularization [1], ensemble distillation [3, 26], etc. We take several representative FL algorithms and use PANs to improve them.

**FL with Permutation Invariance Property:** The permutation invariance of neuron networks could lead to neuron misalignment. PFNM [45] matches local nodes' parameters via Beta-Bernoulli process [35] and Indian Buffet Process [9], formulating an optimal assignment problem and solving it via Hungarian algorithm [18]. SPAHM [44] applies the same procedure to aggregate Gaussian topic models, hidden Markov models, and so on. FedMA [38] points out PFNM does not apply to large-scale networks and proposes a layer-wise matching method. [33] utilizes optimal transport [2] to fuse models with different initializations. These methods are all post-processing ones that need additional computation costs. Fed$^2$ is recently proposed to align features during local training via separating features into different groups. However, it needs to carefully design the architectures. Differently, we take a more fine-grained alignment of neurons rather than network groups, and we will show our method is more general.

**Position Encoding:** Position encoding is popular in sequence learning architectures, e.g., ConvS2S [8] and transformer [36], etc. These architectures take position encodings to consider the order information. Relative position encoding [31] is more applicable to sequences with various lengths. Some other studies are devoted to interpreting what position encodings learn [37, 39]. Another interesting work is applying position encodings instead of zero-padding to GAN [41] as spatial inductive bias. Differently, we resort to position encodings to bind neurons in their positions in FL. Furthermore, these works only consider position encodings at the input layer, while we couple them with neurons.

## 3. Position-Aware Neurons

In this section, we investigate the permutation invariance of neural networks and introduce PANs to control it.

### 3.1. Permutation Invariance Property

Assume an MLP network has $L+1$ layers (containing input and output layer), and each layer contains $J_l$ neurons, where $l \in \{0, 1, \cdots, L\}$ is the layer index. $J_0$ and $J_L$ are input and output dimensions. We denote the parameters of each layer as the weight matrix $W_l \in \mathcal{R}^{J_l \times J_{l-1}}$ and the bias vector $b_l \in \mathcal{R}^{J_l}$, $l \in \{1, 2, \cdots, L\}$. The input layer does not have parameters. We use $h_l \in \mathcal{R}^{J_l}$ as the activations of the $l$th layer. We have $h_l = f_l(W_l h_{l-1} + b_l)$, where $f_l(\cdot)$ is the element-wise activation function, e.g., ReLU [28]. $f_L(x) = x$ denotes no activation function in the output layer. Sometimes, we use $y = v^T f(Wx + b)$ to represent a network with only one hidden layer and the output dimension is one (called as MLP0), where $x \in \mathcal{R}^{J_0}$, $W \in \mathcal{R}^{J \times J_0}$, $b \in \mathcal{R}^J$, $v \in \mathcal{R}^J$. We use $\Pi \in \{0, 1\}^{J \times J}$ as a permutation matrix that satisfies $\sum_j \Pi_{\cdot,j} = 1$ and $\sum_j \Pi_{j,\cdot} = 1$. Easily, we have some properties: $\Pi^T \Pi = I$, $\Pi a + \Pi b = \Pi(a + b)$, $\Pi a \odot \Pi b = \Pi(a \odot b)$, where $I$ is the identity matrix and $\odot$ denotes Hadamard product. If $f(\cdot)$ is an element-wise function, $f(\Pi x) = \Pi f(x)$.

For MLP0, we have $y = (\Pi v)^T f(\Pi W x + \Pi b) = v^T f(Wx + b)$, implying that if we permute the parameters

properly, the output of a certain neural network does not change, i.e., the **permutation invariance property**. Extending it to MLP, the layer-wise permutation process is

$$h_l = f_l(\Pi_l W_l \Pi_{l-1}^T h_{l-1} + \Pi_l b_l), \tag{1}$$

where $\Pi_0 = I$ and $\Pi_L = I$, meaning that the input and output layers are not shuffled. For ConvNet [17, 32], we take convolution kernels as basic units. The convolution parameters could be denoted as $W_l \in \mathcal{R}^{C_l \times w_l \times h_l \times C_{l-1}}$, where the four dimensions denote the number of output/input channels ($C_l$, $C_{l-1}$) and the kernel size ($w_l$, $h_l$). The permutation could be similarly applied as $\Pi_l W_l \Pi_{l-1}^T$. For ResNet [10], we use $h_l = f_l(\Pi_l W_l \Pi_{l-1}^T h_l) + \Pi_l M_l \Pi_{l-1}^T h_l$ to permute all parameters in a basic block including the shortcut (if shortcut is not used, $M_l = I$).

### 3.2. Position-Aware Neurons

The essential reason for the permutation invariance of neural networks is that neurons have nothing to do with their positions. Hence, an intuitive improvement is fusing position-related values (position encodings) to neurons. We propose **Position-Aware Neurons (PANs)**, adding or multiplying position encodings to neurons' outputs, i.e.,

$$\text{PAN}_+ : \ h_l = f_l(W_l h_{l-1} + b_l + \underline{e_l}), \tag{2}$$

$$\text{PAN}_\circ : \ h_l = f_l((W_l h_{l-1} + b_l) \odot \underline{e_l}), \tag{3}$$

where $e_l$ denotes position encodings that are only related to positions and not learnable. We use "$\text{PAN}_+$" and "$\text{PAN}_\circ$" to represent additive and multiplicative PANs, respectively. We use sinusoidal functions to generate $e_l$ as commonly used in previous position encoding works [36], i.e.,

$$\text{PAN}_+ : \ e_{l,j} = A \sin(2\pi T j / J) \in [-A, A], \tag{4}$$

$$\text{PAN}_\circ : \ e_{l,j} = 1 + A \sin(2\pi T j / J) \in [1 - A, 1 + A], \tag{5}$$

where $T$ and $A$ respectively denotes the period and amplitude of position encodings, and $j \in \{0, 1, \cdots, J-1\}$ is the position index of a neuron. *For ConvNet, we assign position encodings for each channel, and $j$ is the channel index. Notably, if we take $T \to 0$ or $A = 0$, PANs degenerate into normal neurons. In practice, we only apply PANs to the hidden layers, while the input and output layers remain unchanged, i.e., $l \in \{1, 2, \cdots, L-1\}$ for $e_l$.* With PANs, the permutation process in Eq. 1 could be reformulated as

$$\text{PAN}_+ : \ h_{l,\text{sf}} = f_l(\Pi_l W_l \Pi_{l-1}^T h_{l-1,\text{sf}} + \Pi_l b_l + \underline{e_l}), \tag{6}$$

$$\text{PAN}_\circ : \ h_{l,\text{sf}} = f_l((\Pi_l W_l \Pi_{l-1}^T h_{l-1,\text{sf}} + \Pi_l b_l) \odot \underline{e_l}), \tag{7}$$

where the subscript "sf" denotes "shuffled" (or permuted). To measure the output change after shuffling, we define the *shuffle error* as:

$$\text{Err}(A, T, \{\Pi_l\}_{l=0}^L) = \|h_{L,\text{sf}} - h_L\| / J_L, \tag{8}$$

and this error on MLP0 without considering bias (i.e., $y = v^T f(Wx + e)$) is

$$\begin{aligned} \text{PAN}_+ : \ &\text{Err}(A, T, \Pi) \\ &= |y_{\text{sf}} - y| \\ &= |(\Pi v)^T f(\Pi W x + e) - v^T f(Wx + e)| \\ &= |(\Pi v)^T f(\Pi W x + e) - (\Pi v)^T f(\Pi W x + \Pi e)| \\ &\approx |(\Pi e - e)^T \frac{\partial y_{\text{sf}}}{\partial e}|, \end{aligned} \tag{9}$$

where we take $y_{\text{sf}} = (\Pi v)^T f(\Pi W x + e)$ as the function of $e$ and take Taylor expansion as an approximation. Obviously, shuffle error is closely related to the strength of permutation, i.e., $\Pi - I$. For example, if $\Pi = I$, the network is not shuffled and the outputs are kept unchanged. Then, if we take equal values as position encodings, i.e., $e_j = e_i, \forall i, j$, the output also does not change because $\Pi e = e$. This can be obtained via taking $\alpha = 0$ or $T \to 0$. If we take a larger $T$ (e.g., 1) and larger $\alpha$ (e.g., 0.05), Err is generally non-zero because $\Pi e \neq e$. The error of multiplicative PANs is similar. *We abstract PANs as a switch: if we take equal/varied position encodings, PANs are turned off/on, and hence the network keeps/loses the permutation invariance property (i.e., the same/different outputs after permutation).* As illustrated at the left of Fig. 1, the five neurons of a certain hidden layer are shuffled while the position encodings they are going to add/multiply are not shuffled, and the outputs will change with PANs turned on.

Furthermore, are there any essential differences between additive and multiplicative PANs, and how much influence do they have on the shuffle error? In Eq. 9, the shuffle error is partially determined by $\partial y_{\text{sf}} / \partial e$, and we extent this gradient to MLP with multiple layers. We assume all layers have the same number of neurons (i.e., $J_l = J, \forall l$) and take the same position encodings (i.e., $e_l = e \in \mathcal{R}^J, \forall l$). We denote $s_{l,\text{sf}} = \Pi_l W_l \Pi_{l-1}^T h_{l-1,\text{sf}} + \Pi_l b_l$ and obtain the recursive gradient expressions:

$$\text{PAN}_+ : \ \frac{\partial h_{l,\text{sf}}}{\partial e} = \text{D}(f_l') \left( \frac{\partial s_{l,\text{sf}}}{\partial h_{l-1,\text{sf}}} \frac{\partial h_{l-1,\text{sf}}}{\partial e} + I \right), \tag{10}$$

$$\text{PAN}_\circ : \ \frac{\partial h_{l,\text{sf}}}{\partial e} = \text{D}(f_l') \Bigg( \frac{\partial s_{l,\text{sf}}}{\partial h_{l-1,\text{sf}}} \frac{\partial h_{l-1,\text{sf}}}{\partial e} \odot [e]^J + \text{D}(s_{l,\text{sf}}) \Bigg), \tag{11}$$

where $\text{D}(\cdot)$ transforms a vector to a diagonal matrix and $[\cdot]^J$ repeats a vector $J$ times to obtain a matrix. $f_l'$ denotes the gradient of activation functions, whose element is 0 or 1 in ReLU. If we expand Eq. 10 and Eq. 11 correspondingly, we will find that the gradient $\frac{\partial h_{L,\text{sf}}}{\partial e}$ of additive PANs does not explicitly rely on $e$. However, for the multiplicative one, $\frac{\partial h_{l,\text{sf}}}{\partial e}$ is relevant to $\frac{\partial h_{l-1,\text{sf}}}{\partial e}$ and $[e]^J$, which could lead to

a polynomial term $A^{L-1}$ (resulted from $[e]^J \odot \cdots \odot [e]^J$, informally). Hence, we conclude: *taking PANs as a switch could control the permutation invariance property of neural networks. The designed multiplicative PANs will make this switch more sensitive.*

## 4. FL with PANs

In this section, we briefly introduce FedAvg [27] and analyze the effects of PANs when applied to FL.

### 4.1. FedAvg

Suppose we have a server and $K$ clients with various data distributions. FedAvg first initializes a global model $\theta_0$ on the server. Then, a small fraction (i.e. $R \in [0, 1]$) of clients $S_t$ download the global model and update it on their local data for $E$ epochs, and then upload the updated model $\theta_0^{(k)}$ to the server. Then, the server takes a coordinate-based parameter averaging, i.e., $\theta_1 \leftarrow \frac{1}{|S_t|}\sum_{k\in S_t}\theta_0^{(k)}$. Next, $\theta_1$ will be sent down for a new communication round. This will be repeated for $H$ communication rounds. Because the parameters could be misaligned during local training, some works [38, 44, 45] are devoted to finding the correspondences between clients' uploaded neurons for better aggregation. For example, the parameters $W_l^{(1)}$ and $W_l^{(2)}$ may be misaligned, and we should search for proper matrices to match them, i.e., $\frac{1}{2}(W_l^{(1)} + M_l W_l^{(2)} M_{l-1}^T)$, rather than $\frac{1}{2}(W_l^{(1)} + W_l^{(2)})$ [33]. However, searching for appropriate $M_{\{l,l-1\}}$ is challenging. Generally, these works require additional data to search for proper alignment. In addition, the matching process has typically to solve complex optimization problems, such as optimal transport or optimal assignment, leading to additional computational overhead. An intuitive question is: *could we pre-align the neurons during local training instead of post-matching?*

### 4.2. Applying PANs to FL

Replacing traditional neurons with PANs in FL is straightforward to implement. Why does such a subtle improvement help? We heuristically expect PANs in FL could bring such effects: *PANs could limit the dislocation of neurons since the disturbance of them will bring significant changes to the outputs of the neural network and lead to higher training errors and fluctuations*. Theoretically, the forward pass on the $k$th client with PANs is as follows:

$$\text{PAN}_+ : \ h_l^{(k)} = f_l^{(k)}(W_l^{(k)} h_{l-1}^{(k)} + b_l^{(k)} + \underline{e_l}), \tag{12}$$

$$\text{PAN}_\circ : \ h_l^{(k)} = f_l^{(k)}((W_l^{(k)} h_{l-1}^{(k)} + b_l^{(k)}) \odot \underline{e_l}). \tag{13}$$

Notably, the position encodings are commonly utilized across clients, i.e., the forward pass across local clients share some consistent information. Then, the parameters' gradient of Eq. 12 and Eq. 13 can be calculated by:

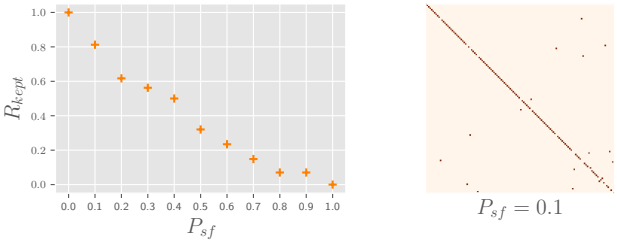


Figure 2. **Left**: how much neurons are not shuffled with various $P_{\text{sf}}$. **Right**: a permutation matrix demo with $P_{\text{sf}} = 0.1$.

$$\text{PAN}_+ : \ \partial h_l^{(k)} / \partial b_l^{(k)} = \text{D}(f_l^{(k)'}), \tag{14}$$

$$\text{PAN}_\circ : \ \partial h_l^{(k)} / \partial b_l^{(k)} = \text{D}(f_l^{(k)'})\text{D}(e_l), \tag{15}$$

where we only give the gradient of bias for simplification. The gradients of multiplicative PANs directly contain the same position information across clients (e.g., $e_l$) in spite of various data distributions (e.g., $h_{l-1}^{(k)}$). For the additive ones, the impact of $e_l$ is implicit because $f_l^{(k)'}$ is related to $e_l$, but nevertheless, the effect is not significant as multiplicative ones. *Overall, $e_l$ could regularize and rectify local gradient directions, keeping some ingredients consistent during backward propagation.* As an extreme case, if $A$ in $e_l$ is very large, the gradients in Eq. 14 and Eq. 15 will tend to be the same, mitigating the weight divergence completely. However, setting $e_l$ too large will make the neural network difficult to train and the data information is completely covered, so the strength of $e_l$ (i.e., $A$) is a tradeoff.

## 5. Experiments

We study how much influence the proposed PANs have on both centralized training and decentralized training (i.e., FL). The datasets used are Mnist [20], FeMnist [4], SVHN [29], GTSRB [34], Cifar10/100 [16], and Cinic10 [5]. FeMnist is recommended by LEAF [4] and FedScale [19]. We use MLP for Mnist/FeMnist, VGG [32] for SVHN/GTSRB/Cifar10, ResNet20 [10] for Cifar100/Cinic10 by default if without more declarations. We sometimes take VGG9 used in previous FL works [26, 38, 43]. For centralized training, we use the provided training and test set correspondingly. For FL, we split the training set according to Dirichlet distributions, where Dir($\alpha$) controls the non-i.i.d. level. Smaller $\alpha$ leads to more non-i.i.d. cases. For each FL scene, we report several key hyper-parameters: number of clients $K$, client participation ratio $R$, number of local training epochs $E$, Dirichlet alpha $\alpha$, number of communication rounds $H$. For PANs, we report $T$ and $A$. *With $A = 0.0$, we turn off PANs, i.e., using traditional neurons or the baselines; with $A > 0.0$, we turn on*

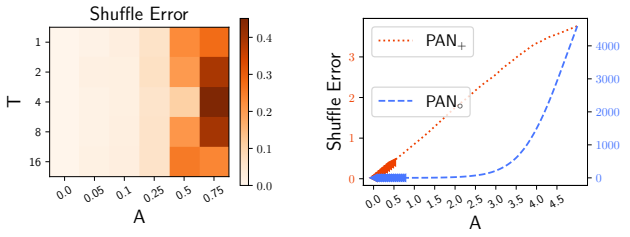


Figure 3. **Left**: shuffle error (Eq. 8) with various $T$ and $A$ (PAN$_\circ$). **Right**: the difference between PAN$_+$ and PAN$_\circ$ ($T$=1). (VGG13 is used, more networks are in Supp.)

*PANs. We* **leave PANs turned on by default** *if with no mention of the state on/off or the value of $A$.* Details of datasets, networks and training are presented in Supp.

### 5.1. Centralized Training

**Shuffle Test:** We first propose a procedure to measure the degree of permutation invariance of a certain neural network, that is, how large the shuffle error in Eq. 8 is after shuffling the neurons. We name this procedure *shuffle test*. Given a neural network and a batch of data, we first obtain the outputs. Then, we shuffle the neurons of hidden layers. The shuffle process is shown in Supp, where $P_{\text{sf}}$ controls the disorder level of the constructed permutation matrices. Then we could get the outputs after shuffling and then calculate the shuffle error. We vary $P_{\text{sf}}$ in $[0, 1]$ and plot the ratio of permutation matrices' diagonal ones (i.e., how much neurons are not shuffled). We denote this ratio as $R_{\text{kept}}$ and plot them in Fig. 2 (average of 10 experiments), where we also show a generated permutation matrix with $P_{\text{sf}} = 0.1$.

**Shuffle Error with Random Data:** With different hyper-parameters of $T$ and $A$ in Eq. 4/Eq. 5, we use random data generated from Gaussian distributions (i.e., $x_{i,\cdot} \sim \mathcal{N}(0, 1)$) to calculate the shuffle error. The results based on VGG13 are shown in Fig. 3. The error is more related to $A$ while less sensitive to $T$. This is intuitive because $T$ controls local volatility while neuron permutation could happen globally, e.g., the first neuron could swap positions with the last neuron. A larger $A$ leads to a larger shuffle error, i.e., the more serious the network loses the permutation invariance property. In addition, the shuffle error based on the additive PANs increases linearly, while that based on the multiplicative PANs increases quickly. This verifies the theoretical analysis in Sect. 3.2. However, in practice, a larger $A$ may cause training failure and we only set $A \in [0.0, 0.25]$ for additive PANs and $A \in [0.0, 0.75]$ for multiplicative PANs (the bold part on the right side of Fig. 3).

**Influence on Inference:** We study the influence of PANs on test accuracies. We use MLP on Mnist, VGG13 on SVHN, and ResNet20 on Cifar10. We first train models with various PANs until convergence, and the model performances are shown in the first figure of Fig. 4. The horizontal dotted lines show the accuracies of normal networks, and the solid segments show the results of networks with various PANs. We find that introducing PANs to neural networks does not improve performances, but brings a slight degradation. That is, PANs could make the network somewhat harder to train. More studies of how PANs influence the network predictions could be found in Supp. Then, we investigate the shuffle error reflected by the change of test accuracies. Specifically, we shuffle the trained network to make predictions on the test set. We vary several groups of $T$ and $A$ for PANs. We show the results in the last three figures of Fig. 4. With larger $P_{\text{sf}}$, i.e., more neurons are shuffled, the test accuracy of the network with $A = 0.0$ does not change (the permutation invariance property). However, larger $A$ leads to more significant performance degradation ($A = 0.25$ vs. $A = 0.01$ for PAN$_+$; $A = 0.75$ vs. $A = 0.05$ for PAN$_\circ$). PAN$_\circ$ makes the network more sensitive to shuffling than PAN$_+$ (curves with "$\circ$" degrades significantly). With different $T \in \{1, 8\}$, the performance degradation is nearly the same, again showing that PANs are robust to $T$. These verify the conclusions in Sect. 3.2. *Overall, PANs work as a tradeoff between model performances and control of permutation invariance.*

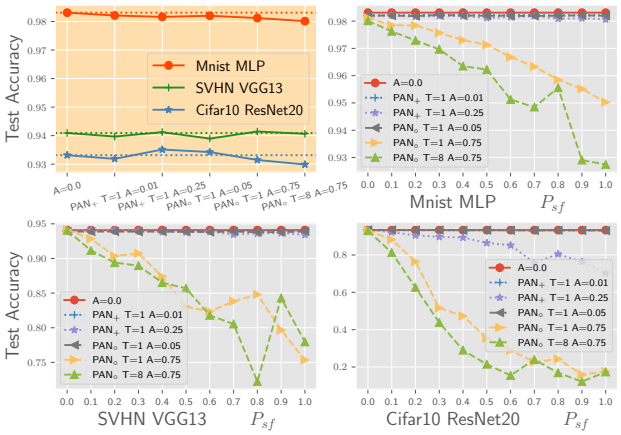


Figure 4. **The first**: test accuracy of models trained with different PANs. **The other three**: test accuracy change after manual permutation with various $P_{\text{sf}}$.

### 5.2. Decentralized Training

Then we study the effects of introducing PANs to FL. We first present some empirical studies to verify the pre-alignment effects of PANs, and then show performances.

**How many neurons are misaligned in FL?** Although some previous works [38, 43, 45] declare that neurons could

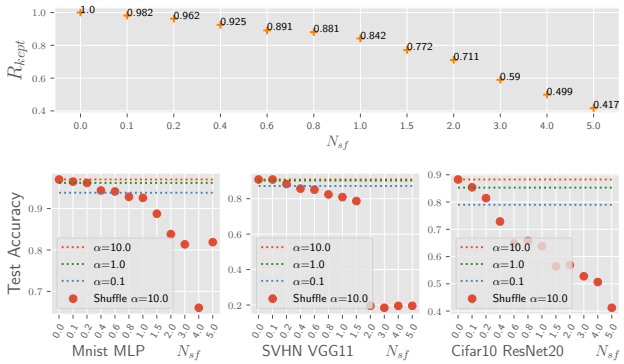


Figure 5. **Top**: how much neurons are not shuffled with various $N_{\text{sf}}$. **Bottom**: test accuracies of FL with various $\alpha$ (dotted lines) and accuracies after manual shufflling on i.i.d. data ($\alpha = 10.0$) (red scatters).

be dislocated when faced with non-i.i.d. data, they do not show this in evidence and do not show the degree of misalignment. We present a heuristic method: *we manually shuffle the neurons during local training with i.i.d. data and study how much misalignment could cause the performance to drop to the same as training with non-i.i.d. data.* Specifically, during each client's training step (each batch as a step), we shuffle the neurons with a probability $\frac{N_{\text{sf}}}{E \times N_k / B}$, where $B$,$E$,$N_k$ are respectively the batch size, the number of local epochs, and the number of local data samples. In each shuffle process, we keep $P_{\text{sf}} = 0.1$. $N_{\text{sf}}$ determines how many times the network could be shuffled during local training. Larger $N_{\text{sf}}$ means more neurons are shuffled upon finishing training, e.g., $N_{\text{sf}} = 1.0$ keeps approximately $84\%$ neurons not shuffled as shown in Fig. 5. The calculation of $R_{\text{kept}}$ in Fig. 5 is presented in Supp. Then, we show the test accuracies of FedAvg [27] under various levels of non-i.i.d. data, i.e., $\alpha \in \{10.0, 1.0, 0.1\}$. The results correspond to the three horizontal lines in the bottom three figures of Fig. 5. The scatters in red show the performances of shuffling neurons with various $N_{\text{sf}}$. Obviously, even with i.i.d. data, the larger the $N_{\text{sf}}$, the worse the performance. This implies that neuron misalignment could actually lead to performance degradation. Compared with non-i.i.d. performances, taking Cifar10 as an example, setting $N_{\text{sf}} = 0.2$ could make the i.i.d. ($\alpha$=10.0) performance degrade to the same as non-i.i.d. ($\alpha$=0.1), that is, approximately $3.8\%$ neurons are misaligned on each client. This may provide some enlightenment for the quantitative measure of how many neurons are misaligned in FL with non-i.i.d. data.

**Do PANs indeed reduce the possibility of neuron misalignment?** We propose several strategies from aspects of parameters, activations, and preference vectors to compare the neuron correspondences in FL with PANs off/on. For

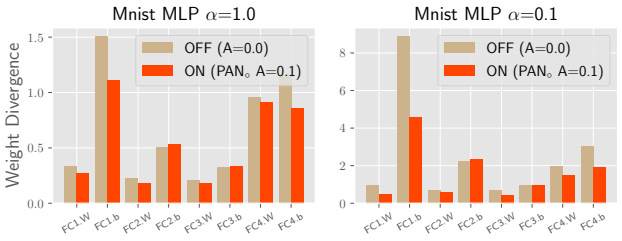


Figure 6. Weight divergence with PANs off/on. ($E = 5$, MLP on Mnist, more datasets' results are in Supp.)

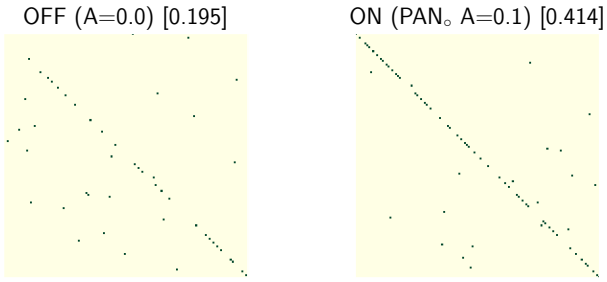


Figure 7. Optimal assignment matrix with PANs off/on, left vs. right. ($\alpha = 1.0$, $E = 20$, VGG9 Conv5 on Cifar10, more results are in Supp.)

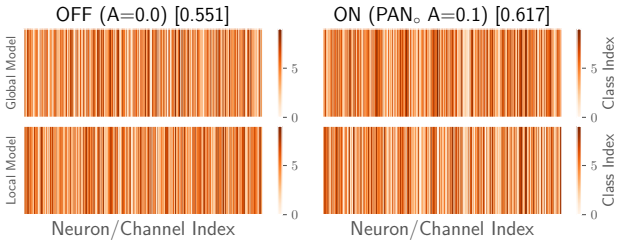


Figure 8. Preference vectors with PANs off/on, left vs. right. ($\alpha = 1.0$, VGG9 Conv6 on Cifar10, more results are shown in Supp.)

PANs turned on, we use multiplicative PANs with $T = 1.0$ and $A = 0.1$ by default.

**I. Weight Divergence:** Weight divergence [47] measures the variances of local parameters. Specifically, we calculate $\frac{1}{|S_t|}\sum_{k \in S_t} \|W_l^{(k)} - W_l\|_2$ for each layer $l$. $W_l = \frac{1}{|S_t|}\sum_{k \in S_t} W_l^{(k)}$ denotes the averaged parameters. The weight divergences of MLP on Mnist with $\alpha \in \{1.0, 0.1\}$ are in Fig. 6, where PANs could reduce the divergences a lot (the red bars). This corresponds to the explanation in Sect. 4.2 that clients' parameters are partially updated towards the same direction.

**II. Matching via Optimal Assignment:** We feed 500 test samples into the network and obtain the activations of each neuron as its representation. Neurons' representations of global and local model are denoted as $h_l \in \mathcal{R}^{J_l \times m}$ and $h_l^{(k)} \in \mathcal{R}^{J_l \times m}$, where $m = 500$. Then we search for the

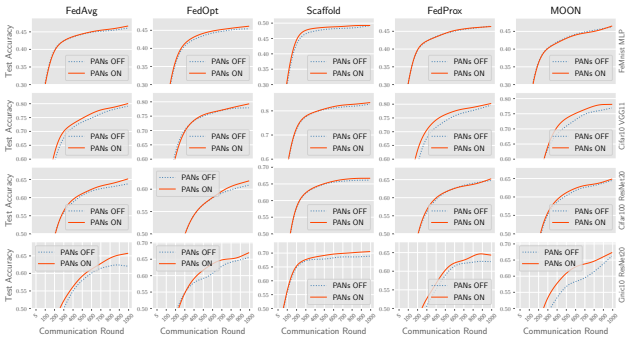


Figure 9. Comparison results on non-i.i.d. data ($\alpha$=0.1). Rows show datasets and columns show FL algorithms. PANs could universally improve these algorithms. (More datasets are shown in Supp.)

optimal assignment matrix $Q \in \{0,1\}^{J_l \times J_l}$ that minimizes $\sum_{i=1}^{J_l}\sum_{j=1}^{J_l} Q_{ij}\|h_{l,i} - h_{l,j}^{(k)}\|_2$ and satisfies $\sum_i Q_{i,\cdot} = 1$, $\sum_j Q_{\cdot,j} = 1$. In fact, $Q$ is a permutation matrix that could approximately reflect the disturbance of neurons, and it could match neurons with similar outputs. We plot the solved matching matrix in Fig. 7, where the number in "[]" shows the ratio of the diagonal ones. Using PANs could make the diagonal denser, implying that neurons at the same coordinates output similarly.

**III. Visualizing Neurons via Preference Vectors:** Then, we correspond neurons to classes via calculating preference vectors as done in [43]. Specifically, we calculate $p_c = \sum_{b=1}^{B} \text{Acti}(x_{c,b}) \frac{\partial Z_c}{\partial \text{Acti}(x_{c,b})}$ for each class $c$, and then concatenate all classes as the preference vector $[p_1, p_2, \cdots, p_C]$. $\text{Acti}(\cdot)$ denotes the activation value and $Z_c$ is the prediction score of the $c$th class. Then, $\arg\max_c p_c$ implies which class the neuron contributes to more. The results are shown in Fig. 8, where each vertical line represents a neuron/channel. The number in "[]" shows how much neurons/channels correspond to the same class between global and local models. With PANs, the coordinate matching results are better. These empirical results verify the pre-alignment effects brought by PANs.

**Do PANs bring performance improvement in FL?** We then compare the performances of FL with PANs off/on.

**I. Universal Application of PANs:** We first apply PANs to some popular FL algorithms as introduced in Sect. 2, including FedAvg [27], FedProx [23], FedOpt [30], Scaffold [14], MOON [22]. These methods solve the non-i.i.d. problem from different aspects. Training details of these algorithms are provided in Supp. We add PANs to them and investigate the performance improvements on FeMnist, Cifar10, Cifar100, and Cinic10, where $\alpha = 0.1$, $K = 100$, $R = 10\%$, $E = 5$, $H = 1000$. We use $A = 0.0$ as the baseline. Hyper-parameters are searched from three groups: $\text{PAN}_+$ with $A = 0.05$, $\text{PAN}_\circ$ with $A = 0.05$, $\text{PAN}_\circ$ with

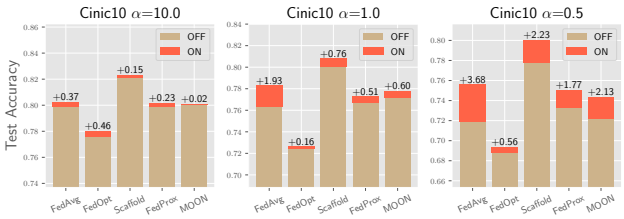


Figure 10. Comparisons under various levels of non-i.i.d. data on Cinic10. Smaller $\alpha$ implies more non-i.i.d. data. (More datasets are shown in Supp.)

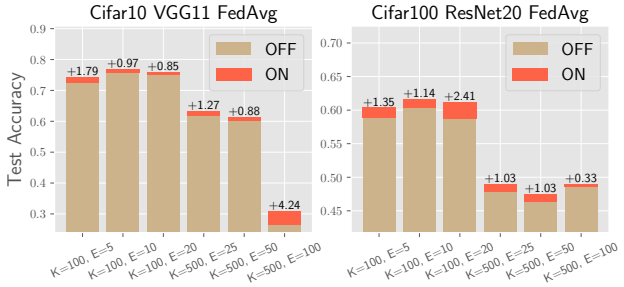


Figure 11. Comparisons under different FL scenes ($K$, $E$) based on FedAvg. (Scaffold results are shown in Supp.)

$A = 0.1$, and the best result is reported in Fig. 9. PANs indeed improve these algorithms. With various non-i.i.d. levels of decentralized data, i.e., $\alpha \in \{10.0, 1.0, 0.5\}$. We report the averaged accuracy of the last five communication rounds in Fig. 10 ($H = 200$ communication rounds with other hyper-parameters the same). Obviously, more non-i.i.d. scenes (smaller $\alpha$) experience more significant improvements. This is related to the regularization effect as analyzed in Sect. 4.2. We also investigate the results with various numbers of clients and local training epochs, i.e., $K$ and $E$. The results of FedAvg on Cifar10 and Cifar100 are shown in Fig. 11, where we take $\alpha = 0.1$ and $H = 400$. On average, introducing PANs could lead to about $1\%$ to $2\%$ improvement on various scenes. These studies verify that PANs could be universally and effectively applied to FL algorithms under various settings.

**II. Hyper-parameter Analysis:** We first vary $A$ on Cifar10 and plot the results on the left of Fig. 12. We set $T = 1.0$ and only report the results of multiplicative PANs. Setting $A$ around 0.1 could improve the performance a lot, while using larger $A$ experiences degradation, which is because neural networks become harder to train. This again shows that $A$ is a tradeoff between neuron pre-alignment and network performance. The proportions of the optimal hyper-parameters from the results of the above experiments are shown on the right of Fig. 12. Using $A = 0.1$ in multiplicative PANs is a good choice. $A = 0.0$ means turning off PANs, and its ratio is only about $13\%$, which means turning

| Settings $(K, R, \alpha, E)$ | FedAvg | FedProx | FedMA | Fed$^2$ | FedDF | FedAvg$^\star$ | FedAvg$^\star$+PANs |
|---|---|---|---|---|---|---|---|
| $(16, 1.0, 0.5, 20)$ | 86.29 | 85.32 | 84.0 (87.53, $E = 150$) | 88.29 | - | 86.83 | **88.49±0.07** |
| $(20, 0.4, 1.0, 40)$ | 78.34 | 78.60 | 65.0 | - | 80.36 | 79.76 | **81.94±0.09** |

Table 1. Comparison results with other popular FL algorithms on Cifar10 with VGG9. The left shows settings. The middle shows the cited results from FedMA [38], Fed$^2$ [43], and FedDF [26]. The last two columns show the results we implement.

| $(K, R, \alpha, E)$ | FedMA$^\star$ | Fed$^{2\star}$ | FedAvg$^\star$+PANs |
|---|---|---|---|
| $(16, 1.0, \underline{0.1}, 20)$ | 83.91 | 82.26 | **85.82 ±0.16** |
| $(16, \underline{0.4}, 0.5, 20)$ | 48.25 | 81.23 | **82.87 ±0.21** |

Table 2. Comparison results with SOTA on more scenes. The results are all implemented by our reproduced code.

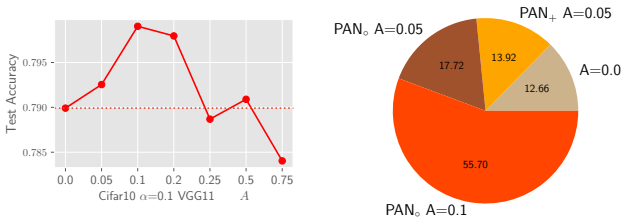


Figure 12. **Left**: performance comparisons under various $A$. **Right**: the distributions of optimal hyper-parameters.

on PANs is useful in most cases.

**III. Comparing with SOTA:** FedMA [38] and Fed$^2$ [43] are representative works that solve the parameter alignment problems in FL. We collect the reported settings and results in FedMA, Fed$^2$, and FedDF [26], and compare the performances under the same settings. We list the results on Cifar10 with VGG9 in Tab. 1, where the last three columns show our results. Although our reproduced FedAvg performs slightly better than the cited results, the performance gain via introducing PANs is remarkable. We then vary the settings of $(16, 1.0, 0.5, 20)$ from two aspects: (1) decreasing the non-i.i.d. $\alpha$ from $0.5$ to $0.1$, i.e., a more non-i.i.d. scene; (2) decreasing the client selection ratio from $1.0$ to $0.4$, i.e., partial client participation. Aside from the above changes, other hyper-parameters are kept the same. We run the code provided by FedMA[1] and reproduce Fed$^2$ via our implementations. The results are listed in Tab. 2. FedMA performs especially worse under partial client participation. Fed$^2$ also performs not so well. Our method surpasses the compared methods obviously in these cases. Furthermore, our method is more efficient, e.g., with four 10-core Intel(R) Xeon(R) Silver 4210R CPUs @ 2.40GHz and one NVIDIA GeForce RTX 3090 GPU card, FedMA needs about 4 hours for a single communication round while ours only requires several minutes.

**IV. More Studies:** We study using optimal transport to fuse neural networks with PANs as done in [33]. We also investigate the BatchNorm [12] and GroupNorm [40] used in VGG or ResNet, where PANs are more applicable to BatchNorm. We finally investigate some varieties of PANs for better personalization in FL [7]. These are provided in Supp.

**V. Disadvantages:** Fusing different values makes the magnitudes of neuron activations/gradients varied, which requires a customized neuron-aware optimizer. In supp, we try applying the adaptive optimizer Adam [15] to PANs, but we do not find too much improvement. Hence, advanced optimizers should be explored in future work.

[1] https://github.com/IBM/FedMA

## 6. Conclusions

We propose position-aware neurons (PANs) to disable/enable the permutation invariance property of neural networks. PANs bind themselves in their positions, making parameters pre-aligned in FL even faced with non-i.i.d. data and facilitating the coordinate-based parameter averaging. PANs keep the same position encodings across clients, making local training contains consistent ingredients. Abundant experimental studies verify the role of PANs in parameter alignment. Future works are to find an optimization method specifically suitable for PANs, and extend PANs to large-scale FL benchmarks or more scenarios that require parameter alignment.

## Acknowledgements

This work is partially supported by National Natural Science Foundation of China (Grant No. 41901270), NSFC-NRF Joint Research Project under Grant 61861146001, and Natural Science Foundation of Jiangsu Province (Grant No. BK20190296). Thanks to Huawei Noah's Ark Lab NetMIND Research Team and CAAI-Huawei MindSpore Open Fund (CAAIXSJLJJ-2021-014B). Thanks for Professor Yang Yang's suggestions. Professor De-Chuan Zhan is the corresponding author.

## References


[1] Durmus Alp Emre Acar, Yue Zhao, Ramon Matas Navarro, Matthew Mattina, Paul N. Whatmough, and Venkatesh Saligrama. Federated learning based on dynamic regularization. In *ICLR*, 2021. 2

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

[3] Rohan Anil, Gabriel Pereyra, Alexandre Passos, Róbert Ormándi, George E. Dahl, and Geoffrey E. Hinton. Large scale distributed neural network training through online distillation. In *ICLR*, 2018. 2

[4] 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. 4

[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. 4

[6] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V. Le, Mark Z. Mao, Marc'Aurelio Ranzato, Andrew W. Senior, Paul A. Tucker, Ke Yang, and Andrew Y. Ng. Large scale distributed deep networks. In *NeurIPS*, pages 1232–1240, 2012. 1

[7] Alireza Fallah, Aryan Mokhtari, and Asuman E. Ozdaglar. Personalized federated learning with theoretical guarantees: A model-agnostic meta-learning approach. In *NeurIPS*, 2020. 8

[8] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. Convolutional sequence to sequence learning. In *ICML*, pages 1243–1252, 2017. 2

[9] Thomas L. Griffiths and Zoubin Ghahramani. The indian buffet process: An introduction and review. *JMLR*, 12:1185–1224, 2011. 2

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016. 2, 3, 4

[11] 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

[12] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, pages 448–456, 2015. 8

[13] Peter Kairouz, H. Brendan McMahan, and et al. Advances and open problems in federated learning. *CoRR*, abs/1912.04977, 2019. 1

[14] 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. 2, 7

[15] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015. 2, 8

[16] Alex Krizhevsky. Learning multiple layers of features from tiny images. 2012. 4

[17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *NeurIPS*, pages 1106–1114, 2012. 2, 3

[18] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955. 2

[19] Fan Lai, Yinwei Dai, Xiangfeng Zhu, Harsha V. Madhyastha, and Mosharaf Chowdhury. Fedscale: Benchmarking model and system performance of federated learning. In *ResilientFL Workshop*, pages 1–3, 2021. 4

[20] 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. 4

[21] Mu Li, Li Zhou, Zichao Yang, Aaron Li, Fei Xia, David G Andersen, and Alexander Smola. Parameter server for distributed machine learning. In *NeurIPS*, volume 6, page 2, 2013. 1

[22] Qinbin Li, Bingsheng He, and Dawn Song. Model-contrastive federated learning. In *CVPR*, pages 10713–10722, 2021. 1, 2, 7

[23] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. In *MLSys*, 2020. 1, 2, 7

[24] Xin-Chun Li and De-Chuan Zhan. Fedrs: Federated learning with restricted softmax for label distribution non-iid data. In *KDD*, pages 995–1005, 2021. 1

[25] Xin-Chun Li, De-Chuan Zhan, Yunfeng Shao, Bingshuai Li, and Shaoming Song. Fedphp: Federated personalization with inherited private models. In *ECML/PKDD*, pages 587–602, 2021. 1

[26] Tao Lin, Lingjing Kong, Sebastian U. Stich, and Martin Jaggi. Ensemble distillation for robust model fusion in federated learning. In *NeurIPS*, 2020. 2, 4, 8

[27] 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. 1, 4, 6, 7

[28] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, pages 807–814, 2010. 2

[29] Yuval Netzer, Tiejie Wang, Adam Coates, A. Bissacco, Bo Wu, and A. Ng. Reading digits in natural images with unsupervised feature learning. 2011. 4

[30] 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. 2, 7

[31] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. Self-attention with relative position representations. In *NAACL-HLT*, pages 464–468, 2018. 2

[32] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015. 2, 3, 4

[33] Sidak Pal Singh and Martin Jaggi. Model fusion via optimal transport. In *NeurIPS*, 2020. 1, 2, 4, 8

[34] 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. 4

[35] Romain Thibaux and Michael I. Jordan. Hierarchical beta processes and the indian buffet process. In *AIS*, pages 564–571, 2007. 2

[36] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, pages 5998–6008, 2017. 2, 3

[37] Benyou Wang, Lifeng Shang, Christina Lioma, Xin Jiang, Hao Yang, Qun Liu, and Jakob Grue Simonsen. On position embeddings in BERT. In *ICLR*, 2021. 2

[38] Hongyi Wang, Mikhail Yurochkin, Yuekai Sun, Dimitris S. Papailiopoulos, and Yasaman Khazaeni. Federated learning with matched averaging. In *ICLR*, 2020. 1, 2, 4, 5, 8

[39] Yu-An Wang and Yun-Nung Chen. What do position embeddings learn? an empirical study of pre-trained language model positional encoding. In *EMNLP*, pages 6840–6849, 2020. 2

[40] Yuxin Wu and Kaiming He. Group normalization. *IJCV*, 128(3):742–755, 2020. 8

[41] Rui Xu, Xintao Wang, Kai Chen, Bolei Zhou, and Chen Change Loy. Positional encoding as spatial inductive bias in gans. In *CVPR*, pages 13569–13578, 2021. 2

[42] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications. *ACM TIST*, 10(2):12:1–12:19, 2019. 1

[43] Fuxun Yu, Weishan Zhang, Zhuwei Qin, Zirui Xu, Di Wang, Chenchen Liu, Zhi Tian, and Xiang Chen. Fed2: Feature-aligned federated learning. In *KDD*, pages 2066–2074, 2021. 1, 4, 5, 7, 8

[44] Mikhail Yurochkin, Mayank Agarwal, Soumya Ghosh, Kristjan H. Greenewald, and Trong Nghia Hoang. Statistical model aggregation via parameter matching. In *NeurIPS*, pages 10954–10964, 2019. 1, 2, 4

[45] Mikhail Yurochkin, Mayank Agarwal, Soumya Ghosh, Kristjan H. Greenewald, Trong Nghia Hoang, and Yasaman Khazaeni. Bayesian nonparametric federated learning of neural networks. In *ICML*, pages 7252–7261, 2019. 1, 2, 4, 5

[46] Manzil Zaheer, Sashank J. Reddi, Devendra Singh Sachan, Satyen Kale, and Sanjiv Kumar. Adaptive methods for nonconvex optimization. In *ICLR*, pages 9815–9825, 2018. 2

[47] Yue Zhao, Meng Li, Liangzhen Lai, Naveen Suda, Damon Civin, and Vikas Chandra. Federated learning with non-iid data. *CoRR*, abs/1806.00582, 2018. 1, 2, 6