Population Normalization for Federated Learning

Zhuoyao Wang[1,*], **Fan Yi**[1,*], **Peizhu Gong**[1], **Caitou He**[2], **Cheng Jin**[1,3], **Weizhong Zhang**[1,4†]
[1]Fudan University [2] Wenzhou University
[3]Innovation Center of Calligraphy and Painting Creation Technology, MCT, China
[4]Shanghai Key Laboratory of Intelligent Information Processing
{18110240064,pzgong,jc,weizhongzhang}@fudan.edu.cn, fyi21@m.fudan.edu.cn, hctou89@wzu.edu.cn

Abstract

Batch normalization (BN) is widely recognized as an essential method in training deep neural networks, facilitating convergence and enhancing model stability. However, in Federated Learning (FL) contexts, where training data are typically heterogeneous and clients often face resource constraints, the effectiveness of BN is considerably limited for two primary reasons. First, the population statistics, specifically the mean and variance of these heterogeneous datasets, vary substantially, resulting in inconsistent BN layers across client models, which ultimately drives these models to diverge further. Second, estimating statistics from a mini-batch is often imprecise since the batch size has to be small in resource-limited clients. This paper introduces Population Normalization, a novel technique for FL, in which the statistics are learned as trainable parameters rather than calculated from mini-batches as in BN. Thus, our normalization layers are homogeneous among the clients and the adverse impact of small batch size is eliminated as the model can be well-trained even when the batch size equals to one. To enhance the flexibility of our method in practical applications, we investigate the role of stochastic uncertainty in BN's statistical estimation. When larger batch sizes are available, we demonstrate that injecting simple artificial noise can effectively mimic this stochastic uncertainty and improve the model's generalization capability. Experimental results validate the efficacy of our approach across various FL tasks.

1. Introduction

Batch Normalization (BN) [17] has become a standard component in deep neural networks (DNNs). It is proposed to stabilize and accelerate the training of DNNs, and the original motivation was to reduce the so-called internal covariate shift. In each training step, it normalizes the layer input of a neural network using the mean and variance computed within a randomly sampled mini-batch. Specifically, consider a particular neuron in a network, and we denote $\{x_1, \ldots, x_m\}$ to be its pre-activations on a mini-batch $\mathcal{B}$ of size m, BN first normalizes each x_i as

$$\hat{x}_i = \frac{x_i - \mu_{\mathcal{B}}}{\sigma_{\mathcal{B}}}, \tag{1}$$

where $\mu_{\mathcal{B}}$ and $\sigma^2_{\mathcal{B}}$ are the sample mean and variance of the pre-activation on $\mathcal{B}$. It then scales and shifts each $\hat{x}_i$ as $y_i = \alpha\hat{x}_i + \beta$ to compensate the effect of normalization, where γ and β are two parameters to be learned during the training process. BN finally takes y_i as the new pre-activation of this neuron. It has been reported in literature [12, 16, 17, 37, 40] that BN can significantly improve the convergence of training and the generalization performance of the final learned models.

However, we notice that the effectiveness of BN is compromised in Federated Learning [14, 19, 42], which typically aims at collaboratively training models on resource limited clients (e.g., mobiles) with locally private and heterogeneous training data. The main reasons can be summarized as follows. One is that the statistics, i.e., the mean and variance, of heterogeneous datasets in the clients differ greatly. Eqn. (1) implies that this can lead to inconsistent BN layers among the client models and finally causes these models to drift further away from each other. The other is that in resource limited clients, the mini-batch size has to be small, which makes $\mu_{\mathcal{B}}$ and $\sigma^2_{\mathcal{B}}$ inaccurate estimations of the population mean and variance for normalization and the obtained BN would cause instability during training that slows down convergence.

Some other normalization techniques, such as batch renormalization [16], group normalization [40], and layer normalization [2], are proposed to address this heavy reliance on batch size. The key ideas are to estimate the population mean and variance as accurately as possible by using techniques such as moving average [16] or to normalize the activation along other dimensions such as the channel dimension [2]. Nevertheless, these approaches focus on training acceleration, which typically under-perform

*Equal Contribution, †Corresponding authors.

BN in terms of the trained model's accuracy [40]. Recently, FedBN [25] is proposed to extend BN to FL. It keeps the client BN layers updated locally, without communicating and aggregating them during training. This enables the clients to achieve better performances in their own respective environments. However, since the BN layers among the clients can be heterogeneous due to the non-IID datasets, FedBN cannot produce an off-the-shelf global model after training, which is often required in reality. Consequently, FedBN suffers from the cold start issue, i.e., the learned model cannot be directly deployed in the new clients due to the lack of trained local BN layers. What's more, FedBN still relies on estimation within mini-batches, which cannot address the issue of inaccurate estimation caused by small batch size in cross-device FL.

In this paper, we propose Population Normalization (PN), a simple yet effective normalization method for FL to eliminate the negative impacts of statistical heterogeneity and estimation inaccuracies on BN. The key idea is to learn population statistics as trainable parameters in the training process by introducing two constraints derived from the definitions of mean and variance (see Section 3), instead of estimating them from the mini-batches as most existing methods have done. As the population statistics take place of the sample mean and variance of the local datasets in BN, and are treated as normal trainable parameters in local training and aggregation, our normalization layers among the clients would be consistent although the datasets are heterogeneous. This enables our method to eventually produce a global model with the normalization layer among the clients. In addition, since we no longer need to calculate the sample mean and variance, the resulting training problem can be solved by stochastic algorithms of FL with arbitrary batch sizes (even equals to one) and thus the issue of inaccurate statistic estimation because of small batch size is eliminated.

In practice, we observe that PN tends to under-perform BN given very large batch sizes. We then further explore this issue and find its cause to be that PN lacks the stochastic uncertainty in the estimation of mean and variance. Specifically, we show that the noise introduced by the mini-batch based mean and variance estimations in the BN is essential for its good generalization behavior. Our empirical results demonstrate that injecting a reasonable amount of data independent noise into BN with extremely large batch sizes can significantly improve the generalization performance to be comparable to that of BN with proper batch sizes. Inspired by this finding, we propose Noisy Population Normalization (NPN) by injecting simple data independent artificial noises into the learned population mean and variance in PN during training. The injected noise can serve the same purpose as the uncertainty in estimating population statistics in BN with small batch sizes. This makes the proposed method more flexible in the applications of FL where large batch size is available, e.g., collaboration among companies in cross-silo FL [9, 15, 21].

Finally, we conduct a series of experiments to verify the effectiveness of our proposed PN and NPN in the context of FL with small and large batch sizes, respectively (Section 5). The results demonstrate that PN effectively reduces the reliance on training batch size and alleviates the negative impact of data heterogeneity on normalization. Furthermore, the artificial noise injected in our NPN method effectively improves the generalization performance given large training batch sizes.

Our main contributions are summarized as follows:

- We propose a mormalization technique PN for federated learning, which demonstrates superior performance given small batch sizes and data heterogeneity.
- In training with large batch sizes, we propose NPN based on PN, which introduces stochastic uncertainty by injecting artificial noise to improve generalization ability.
- We conduct extensive experiments to verify the effectiveness of our method.

Notations: Given a positive integer d, we define $[d]$ to be the set $\{1, 2, \ldots, d\}$. For a vector $x \in \mathbb{R}^d$, we let $[x]_i$ be its i-th component with $i \in [d]$. We denote by $w_{i,j}$ the element in the i-th row and the j-th column of a real-valued matrix w. Let $A \cup B$ be the union of two sets A and B. Moreover, we denote by $\mathcal{N}(\mu, \sigma^2)$ the Gaussian distribution with mean of μ and variance of σ^2.

2. Related Work

2.1. Normalization Techniques

BN [17] is a standard normalization technique, which enables the training of very deep neural networks, and the most representative result is ResNet[11], which cannot be easily trained to converge without BN. It can also achieve significant imporement in generalization. Therefore, the invention of BN is a major milestone in the development of deep learning.

As aforementioned, small batch size or non-iid mini-batch can diminish the effect of BN on the training acceleration or even result in training failure, due to the inaccurate mean and variance estimations [16]. Moreover, large batch size leads to huge memory cost. This issue hinders BN's usage in FL applications, where the clients often have limited computational resources and the datasets are usually heterogeneous. This in turn prevents the training of large networks in FL. Recently, some new normalization methods have been proposed as alternatives to BN, and they can be roughly divided into two categories, i.e., normalization along the batch dimension [16], and mini-batch independent normalization [2, 31, 37, 40]. The former alleviates the reliance on the batch size by using techniques such as

moving average. The latter ones achieve this by normalizing the activation of neurons in a network along other dimensions. For example, Group Normalization [40] divides the channels in each hidden layer into groups and normalizes the activations in each group with its mean and variance. Empirical studies show that all these methods achieve comparable training speedups with BN. However, the generalization performance they achieved is inferior to that of BN.

Besides the above studies, there is another line of research [4, 26, 29, 32, 39] that focuses on understanding BN. One of the main conclusions in these investigations is that with BN, the effective objective function used in training is different from the original one. Specifically, [32] showed that by inserting BN layers into a network, the loss landscape becomes smoother, which naturally explains why BN enables one to use large learning rates. The studies [29, 39] focuse on the regularization effect of BN. They showed that the objective function with BN can be divided into two parts: an empirical loss under population normalization and an explicit regularizer named gamma decay.

2.2. Federated Learning

FL emerges as an effective paradigm to train models on distributed devices without centralizing local private data [3, 8, 13, 19, 42]. Specifically, it can be formulated as follows:

$$\min_{\theta} \mathcal{L}(\theta, \mathcal{D}) \triangleq \sum_{m=1}^{M} \rho_m \mathcal{L}_m(\theta, \mathcal{D}_m), \tag{2}$$

where M is the number of clients and θ is the model to be learnt. The global dataset $\mathcal{D} = \cup_{m=1}^{M} \mathcal{D}_m$, where $\mathcal{D}_m = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{|\mathcal{D}_m|}$ is the locally kept dataset on client m. $\mathcal{L}_m(\theta, \mathcal{D}_m) = \frac{1}{|\mathcal{D}_m|} \sum_{\xi \in \mathcal{D}_m} \ell(\theta, \xi)$ denotes the local empirical risk with the loss function $\ell(\cdot, \cdot)$ and $\rho_m = \frac{|\mathcal{D}_m|}{|\mathcal{D}|}$. Typically, the most popular FL algorithm FedAvg [30] solves the problem (2) based on local-SGD [34]. Specifically, in the c^{th} round, it updates the global model on the server as follows:

$$\theta^{c+1} = \frac{1}{M} \sum_{m=1}^{M} \rho_m \theta_m^c, \tag{3}$$

where θ_m^c denotes the received local model from client m after local training in this round. The updated global model θ^{c+1} is then distributed to the clients to initiate the $(c+1)$-th round of training.

In FL, especially in mobile edge scenarios, both limited device capacity and non-IID distribution of clients' local data raise challenges to the training procedure [27]. Various works attempt to tackle the non-IID distribution of data for federated learning from the perspective of local training regularization [18, 22, 23], server aggregation [5, 28, 38, 41, 43], personalized federated learning [10, 20, 24, 35], etc. However, they do not explore the reduced effectiveness of batch normalization in federated learning. FedBN [25] uses local BN parameters to tackle non-iid federated learning. However, FedBN is only applicable to local inferences where only clients participating in the training with local statistics can perform inference, without providing a global model for new clients or global data. Also, FedBN does not consider the small batches caused by limited local resources.

Algorithm 1 Population Normalization

Input: activation: x; mean and variance: μ and $1/\gamma^2$; scale and shift parameters: α and β.
Normalize using the population statistics:

$$\hat{x} = \frac{x - \mu}{\sqrt{\gamma^2 + \epsilon}}$$

Scale and shift:

$$y = \alpha \hat{x} + \beta \equiv \mathrm{PN}_{\alpha,\beta}^{\mu,\gamma}(x)$$

Return: y

3. Population Normalization

We now present our method PN (see Alg.1). Its key idea is to eliminate the impacts of heterogeneous statistics and inaccurate statistic estimations by learning the population mean and variance during the training process, instead of estimating them within the mini-batch as BN has done. Precisely, given an activation x, we normalize it as follows:

$$\hat{x} = \frac{x - \mu}{\sqrt{\gamma^2 + \epsilon}},$$

where μ and γ^2 are the population mean and variance, respectively, and they will be learned during the training process, of which the details are specified later in Eqn.(3). $\epsilon > 0$ is a small constant. Then, to compensate the effect of normalization on the effective capacity of a neural network, we scale and shift the normalized activation $\hat{x}$ as:

$$y = \alpha \hat{x} + \beta \equiv \mathrm{PN}_{\alpha,\beta}^{\mu,\gamma}(x), \tag{PN}$$

where the coefficients α and β are learned in the training process together with μ and γ.

Below, we take a fully connected neural network $f(\boldsymbol{x}) \in \mathbb{R}$ as an example to show how to insert PN layers into DNNs. Specifically, given an input $\boldsymbol{x} \in \mathbb{R}^d$, let the input layer be $f_j^{(0)}(\boldsymbol{x}) = [\boldsymbol{x}]_j$ with $j \in [n^{(0)}] = [d]$. The ℓ-th hidden layer containing $n^{(\ell)}$ hidden nodes with $\ell \in [L]$ takes

the form of

$$f_j^{(\ell)}(\boldsymbol{x}) = h^{(\ell)}\left(z_j^{(\ell)}(\boldsymbol{x})\right) \text{ with } \tag{4}$$

$$z_j^{(\ell)}(\boldsymbol{x}) = \sum_{k=1}^{n^{(\ell-1)}} w_{j,k}^{(\ell)} \underbrace{\left(\alpha_k^{(\ell-1)} \frac{f_k^{(\ell-1)}(\boldsymbol{x}) - \mu_k^{(\ell-1)}}{\sqrt{\left(\gamma_k^{(\ell-1)}\right)^2 + \epsilon}} + \beta_k^{(\ell-1)} \right)}_{(\ell-1)-\text{th PN layer}},$$

where $j \in [n^{(\ell)}]$, $z_j^{(\ell)}(\boldsymbol{x})$ is the pre-activation, $h^{(\ell)}$ is the activation function and $w^{(\ell)} \in \mathbb{R}^{n^{(\ell)} \times n^{(\ell-1)}}$ is the weight matrix connecting layer ℓ and $\ell - 1$. $\mu_k^{(\ell-1)}$ and $\gamma_k^{(\ell-1)}$ are the population mean and standard deviation of the hidden nodes $f_k^{(\ell-1)}(\boldsymbol{x})$. $\alpha_k^{(\ell-1)}$ and $\beta_k^{(\ell-1)}$ are the parameters of the linear transformation. The final output layer is defined as

$$f(\boldsymbol{x}) = \sum_{k=1}^{n^{(L)}} u_k \underbrace{\left(\frac{f_k^{(L)}(\boldsymbol{x}) - \mu_k^{(L)}}{\sqrt{\left(\gamma_k^{(L)}\right)^2 + \epsilon}} \right)}_{\text{final PN layer}} + c_0, \tag{5}$$

where $u \in \mathbb{R}^{n^{(L)}}$ is the weight vector connecting the output to the L-th hidden layer, $\mu_k^{(L)}$ and $\gamma_k^{(L)}$ are the population mean and standard deviation of the hidden nodes $f_k^{(L)}(\boldsymbol{x})$. The scaling and shifting parameters are absorbed into u_k and c_0, respectively.

To eliminate the dependency of PN on batch size, we learn the statistics $\mu_k^{(\ell)}$ and $\gamma_k^{(\ell)}$ in training by introducing two natural constraints derived from the definition of mean and variance. The statistic estimation is optimized together with the model parameters during local training, and aggregated on the server by FL algorithms, such as FedAVG. To be precise, given the loss $\phi(\cdot,\cdot) : \mathbb{R} \times \{0,1\} \to \mathbb{R}$ and the global dataset $\mathcal{D} = \cup_{m=1}^{M} \mathcal{D}_m$, the training objective in Eqn. (2) of a FL problem becomes

$$\min_{\theta} \mathcal{L}(\theta, \mathcal{D}) \quad s.t. \begin{cases} \frac{1}{|\mathcal{D}|} \sum_{(\boldsymbol{x},y)\in\mathcal{D}} \frac{\left(f_k^{(\ell)}(\boldsymbol{x}) - \mu_k^{(\ell)}\right)^2}{\left(\gamma_k^{(\ell)}\right)^2 + \epsilon} = 1 \\ \frac{1}{|\mathcal{D}|} \sum_{(\boldsymbol{x},y)\in\mathcal{D}} f_k^{(\ell)}(\boldsymbol{x}) = \mu_k^{(\ell)} \end{cases},$$
$$k \in [m^{(\ell)}], \ell \in \{0\} \cup [L]$$

where θ is the set of all the trainable parameters, i.e., $\theta = \{w^{(\ell)}\} \cup \{u\} \cup \{\mu^{(\ell)}\} \cup \{\gamma^{(\ell)}\} \cup \{\alpha^{(\ell)}\} \cup \{\beta^{(\ell)}\} \cup \{c_0\}$ and $\mathcal{L}(\theta, \mathcal{D}) = \sum_{m=1}^{M} \rho_m \sum_{(\boldsymbol{x},y)\in\mathcal{D}_m} \phi(f(\boldsymbol{x}), y)$. This training problem can be solved by various stochastic algorithms with arbitrary batch sizes (even equals to 1), such as moving average or the method of Lagrangian multipliers. In the experiments of this paper, we use stochastic Lagrangian multipliers and the details can be found in the appendix.

Discussion. (1) Since the statistics μ and γ are learned in the training process instead of being estimated from mini-batches, the resulting training problem can be solved by stochastic solvers of FL with arbitrary batch sizes (even equals to 1). Thus the reliance on batch size is eliminated. This enables resource limited clients to train larger deep neural networks. (2) Both μ and γ are treated as normal trainable parameters in local training, communication and aggregation, our PN layers are consistent among the clients although the datatsets on the clients are heterogeneous. This enables us to eventually obtain an off-the-shelf global model, which is often required in reality.

Remark 3.1 *We should point out that our PN layer can also be inserted before the activation functions.*

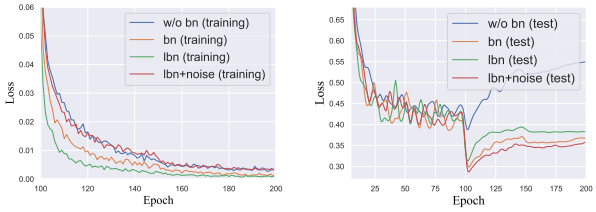


Figure 1. Training loss (left) and test loss (right) of BN, LBN and LBN+noise. Injecting artificial noise into estimated BN statistics successfully decreases the test loss and improves generalization.

4. Noisy Population Normalization

We observe that PN demonstrates inferior performance relative to BN when batch size becomes larger. We explore this issue and find its cause to be that PN lacks stochastic uncertainty in the mean and variance estimations. In practice, large batch sizes may often be used in cross-silo federated learning, where the participants are companies and organizations with huge amounts of data as well as abundant computational resources [9, 15, 21]. Therefore, to enable our method to be more flexibly used in such scenarios, we propose NPN, which injects artificial noise into the learned statistics during training to boost generalization. The details are presented in the following subsections.

4.1. The Role of Stochastic Uncertainty in BN

We provide an in-depth exploration into the role of stochastic uncertainty of the mini-batch statistics in BN. This provides a deep understanding of BN's batch size reliance.

We start with a simple experiment to show the effect of batch size on BN. Precisely, we compare the performances of BN with a reasonable batch size (m=128) and a very large size (m=2048) in training VGG-16 on CIFAR-10. For simplicity, we refer to BN with m=2048 as LBN. For fair comparison, in LBN, to eliminate the effect of large batch size on SGD, we only use large batches to obtain batch statistics and calculate the stochastic gradient with respect to the loss on mini-batches of size 128.

We present both the training and test losses of the learned models with BN and LBN over the training process in Fig

1, from which we can see that LBN converges faster than BN, due to the more accurate population statistics estimation. However, there is a noticeable gap between their test losses, which becomes larger after the learning rate decays. Therefore, the effect of BN in improving generalization is reduced when the batch size becomes too large. This indicates that the stochastic uncertainty in estimates of the mean and variance is important for BN.

The next question we want to investigate is whether we can artificially inject a proper amount of data-independent noise into LBN to improve its generalization performance. Note that the noise of estimating $\mu_{\mathcal{B}}$ and $\sigma_{\mathcal{B}}$ in BN is data dependent. To answer this question, we should first calculate the uncertainty explicitly. The studies [29, 36] show that for sufficiently large batch size M, the estimated $\mu_{\mathcal{B}}$ and $\sigma_{\mathcal{B}}$ can be viewed as two random variables sampled from two Gaussian distributions, i.e.,

$$\mu_{\mathcal{B}} \sim \mathcal{N}\left(\mu, \frac{\sigma^2}{m}\right) \text{ and } \sigma_{\mathcal{B}} \sim \mathcal{N}\left(\sigma, \left(\frac{\mu_4/\sigma^4 - 1}{4m}\right)\sigma^2\right), \quad (6)$$

where μ and σ are the mean and variance of the activation $x \in \mathbb{R}$ and μ_4 is its 4-th central moment, i.e., $\mu_4 = \mathbb{E}_x(x-\mu)^4$. The item μ_4/σ^4 in Eqn.(6) is well-known as the *kurtosis* [6]]. It holds that $\mu_4/\sigma^4 \geq 1$ for any distribution. Although there is no upper limit to kurtosis of a general distribution, for most well-known distributions, e.g., Gaussian distribution, uniform distribution and logistic distribution, it is finite [6]. Therefore, the variance of $\sigma_{\mathcal{B}}$ can always be regarded as $\mathcal{O}(\frac{\sigma^2}{m})$.

Notice that the magnitudes of both $\mu_{\mathcal{B}}$ and $\sigma_{\mathcal{B}}$ of different neurons could be quite different. The analysis above indicates that to inject noise into $\mu_{\mathcal{B}}$ and $\sigma_{\mathcal{B}}$, it is better to multiply them by a Gaussian noise centered at 1 with proper variance instead of add them by two zero-mean noises. One reason is that the standard deviation of $\sigma_{\mathcal{B}}$ is proportional to σ when m is sufficiently large. More importantly, it is impossible to choose the right amount of noise for each $\mu_{\mathcal{B}}$ and $\sigma_{\mathcal{B}}$ if we add noises additively. Therefore, we multiplicatively inject noise into its mean and variance estimations of LBN as follows:

$$\mu_{\mathcal{B}} \leftarrow \mu_{\mathcal{B}} \cdot n_\mu \text{ and } \sigma_{\mathcal{B}} \leftarrow \sigma_{\mathcal{B}} \cdot n_\sigma,$$

where n_μ and n_σ are two noises sampled from $\mathcal{N}(1, 0.1)$ and they are treated as untrainable parameters in each back propagation. The result is presented in Fig 1. It shows that with such noise, the performance of LBN can be boosted to be comparable with BN.

The main conclusion we can reach in the experiment above is that data-independent noise can be used to improve the performance of LBN. Moreover, it also indicates that the effect of BN on improving generalization can be decomposed into two parts: one is population normalization (can be seen from the improvement of LBN over w/o bn) and the other is the stochastic uncertainty contained in the mini-batch statistics (can be seen from the improvement of LBN+noise over LBN). This phenomenon inspires us to inject stochastic noise into our PN to improve its generalization performance.

4.2. Noisy Population Normalization

Below, we develop our final method NPN (Alg.2) to achieve better generalization performance. It is inspired from our above understandings of the stochastic uncertainty in BN. In particular, we showed that data independent noise can be injected into LBN to boost the generalization performance.

At first, for each activation x, we insert data-independent noise into the corresponding μ and γ to improve generalization. In order to reduce the effects of different magnitudes of μ and γ corresponding to different neurons, similar to LBN+noise in Section 4.1, we multiply μ and γ by two noises centered at 1 instead of adding two zero-mean ones as follows:

$$\hat{\mu} = \mu(1 + n_\mu) \text{ and } \hat{\gamma} = \max\{0, \gamma(1 + n_\gamma)\},$$

where n_μ and n_γ are independently sampled noises from a zero-mean distribution, e.g., Gaussian, and $\max$ is used to avoid negative $\hat{\gamma}$. We then normalize, scale and shift x by:

$$\hat{x} = \frac{x - \hat{\mu}}{\sqrt{\hat{\gamma}^2 + \epsilon}} \text{ and } y = \alpha\hat{x} + \beta, \quad \text{(NPN)}$$

where μ, γ, α and β are trainable parameters. The training process with NPN is almost the same as PN, except that the injected noises n_μ and n_γ are treated as constants in back propagation. The details of the training process can be found in the appendix.

Discussion. NPN has the following appealing features. As the injected noises n_μ and n_γ are data independent,we are able to flexibly inject a proper amount of noise into the learnt estimation of statistics to improve generalization ability, which is particularly important when large batch sizes are used. This scenario is often encountered in cross silo federated learning. The idea of injecting data independent noise to improve generalization can also be combined with other normalization methods. The success of NPN shows that one can directly learn the population statistics in the training process instead of estimating them from the mini-batches. This can inspire new normalization methods.

Remark 4.1 *It is unclear that whether NPN has some other nice properties of BN, e.g., auto-tuning of learning rates due to scale invariance [1, 7], while our empirical results demonstrate the effectiveness of NPN. We argue that different from standard training, the performance improvements achieved by these properties could be negligible compared to normalization itself in FL as the training in FL is more challenging than standard training. Such theoretical analysis is left as future work. Thus, we would like to point out that NPN is a normalization technique for FL instead of a general approach.*

batch size	Normalization	α (CIFAR-10)				α (CIFAR-100)			
		0.04	0.08	0.16	1.00	0.04	0.08	0.16	1.00
8	w/o	69.83	83.22	85.05	87.59	45.13	50.43	54.26	58.47
	IN	67.68	78.55	82.62	84.09	35.51	40.02	45.48	45.45
	GN	71.05	82.54	83.35	86.11	45.45	49.22	52.20	54.75
	BN	70.22	82.27	85.77	89.42	52.13	55.91	58.16	64.88
	FedBN	71.49	83.05	84.41	89.30	52.05	56.12	57.97	64.43
	PN	**78.59**	**85.45**	**88.05**	**89.63**	**53.06**	**57.43**	**61.87**	**65.00**
4	w/o	70.69	82.02	85.11	86.93	46.99	50.83	55.08	59.45
	IN	60.53	74.08	78.11	82.39	36.04	42.96	46.17	49.89
	GN	68.63	78.53	82.01	85.52	44.21	48.18	51.85	54.67
	BN	70.18	82.47	86.32	89.52	51.31	55.78	60.35	64.32
	FedBN	72.24	83.57	86.03	89.46	51.45	55.77	59.91	65.10
	PN	**78.40**	**85.05**	**87.52**	**89.59**	**52.28**	**56.85**	**62.42**	**65.82**
2	w/o	67.03	79.98	83.43	87.01	43.93	48.56	54.77	59.43
	IN	55.42	68.93	75.43	78.79	36.15	43.90	47.72	47.30
	GN	66.02	78.82	81.75	85.23	44.40	46.17	50.60	53.24
	BN	65.32	79.68	83.30	87.25	45.87	50.66	55.06	58.74
	FedBN	65.50	80.02	82.17	86.40	46.04	51.30	55.52	59.68
	PN	**72.53**	**82.59**	**86.23**	**88.98**	**48.97**	**54.92**	**60.64**	**65.46**
1	w/o	60.49	71.83	78.65	84.93	38.24	42.26	49.49	54.59
	IN	35.47	54.93	66.45	73.74	33.60	39.06	43.29	48.82
	GN	60.55	71.83	75.25	79.68	46.04	49.26	53.97	56.64
	BN	NA	NA	NA	NA	NA	NA	NA	NA
	FedBN	NA	NA	NA	NA	NA	NA	NA	NA
	PN	**64.38**	**78.89**	**82.13**	**86.69**	**46.13**	**49.32**	**59.19**	**62.97**

Table 1. The results of different normalization methods on CIFAR-10/100 with different batch sizes and Non-IID degree α. Or PN consistently outperforms other methods by a large margin across different α and batch sizes.

Algorithm 2 Noisy Population Normalization

Input: activation: x; population mean and variance: μ and $1/\gamma^2$; scale and shift parameter: α and β; noise level: σ.

Inject noise into μ and γ:

$$\hat{\mu} = \mu(1 + n_\mu) \text{ and } \hat{\gamma} = \max\{0, \gamma(1 + n_\gamma)\},$$

where n_μ and n_γ are noises from $\mathcal{N}(0, \sigma^2)$.

Normalize using noisy population statistics:

$$\hat{x} = \frac{x - \hat{\mu}}{\sqrt{\hat{\gamma}^2 + \epsilon}}$$

Scale and shift:

$$y = \alpha\hat{x} + \beta \equiv \text{NPN}^{\mu,\gamma}_{\alpha,\beta}(\sigma; x)$$

Return: y

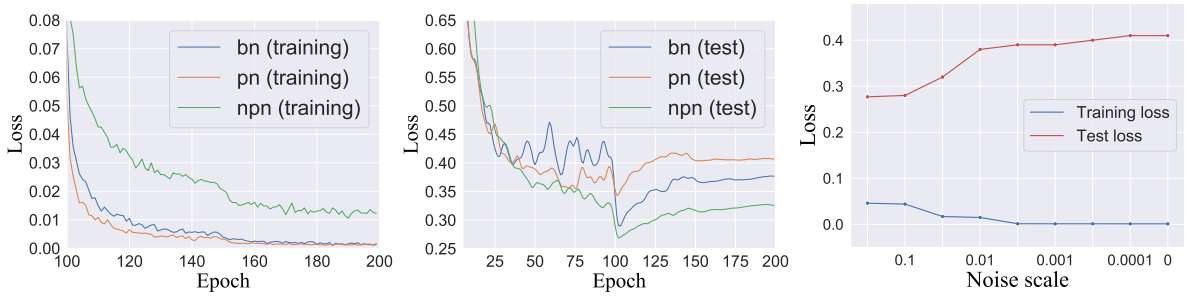
5. Experiments

In this section, we conduct two sets of experiments. Firstly, we verify the effectiveness of of PN in the federated learning setting given small batch sizes and heterogeneous client data. Secondly, we demonstrate the superior generalization ability of our NPN given large training batch sizes.

5.1. Settings and Baselines

To verify the advantage of our population normalization method, we compare it with other normalization counterparts in the FL setting. Specifically, we compare with batch normalization [16], group normalization [40], instance normalization [37] and without normalization. We compare with FedBN [25] by averaging the BN parameters in the clients. Extensive experiments are conducted on two image classification benchmarks (*i.e.*, CIFAR-10 and CIFAR-100) with models VGG-16 [33] and ResNet [11] to evaluate the effectiveness of our method.

For FL experiments, we adopt FedAvg [30] as the base algorithm. We use 80 clients in total with participation rate set to 0.4 throughout all experiments. For the clients' local update, we adopt SGD with momentum. For batch size of 8, the learning rates are set to be the best among {0.005, 0.01, 0.15, 0.02, 0.25} for each normalization method, and linearly down-scaled for other smaller batch sizes. We run 200 global communication rounds with local epoch set to 1. We examine the superiority of our method against the baseline under various batch sizes and heterogeneity degrees. We follow prior work [5, 28] to use Dirichlet distribution for simulating the non-IID data distribution, where the degree of heterogeneity is defined by α, smaller α value corresponds to more severe heterogeneity. We report the global model's average performance in the last five rounds evaluated using the test split of the datasets. For other details, please refer to the Appendix.



(a) (b)

Figure 2. (a) Training (left) and test (right) loss of BN, PN, and NPN as the training processes; (b) Sensitivity of NPN on noise level.

For experiments with large batch sizes, we adopt the distributed training setting, where the model is being trained by 4 different participants. The batch size is set to 2048. The other hyper-parameters are set to be the same as in the FL setting. We conduct the experiments using VGG-16 trained on CIFAR-10 and CIFAR-100 datasets.

5.2. Federated Learning with Small Batch Size

We point out that with small batch sizes, SGD already introduces enough stochastic uncertainty in gradient estimation, which helps generalization. Therefore, we do not need to inject additional noise for PN and thus the experiments with NPN are omitted in this section.

Comparison with Various Normalization Methods We compare PN with other counterparts under different batch sizes and non-iid degrees in Table 1. Specifically, we conduct experiments with $\alpha \in \{0.04, 0.08, 0.16, 1.00\}$ and batch sizes $\{1, 2, 4, 8\}$. We adopt VGG16 as the trained network. As demonstrated in the table, we observe that our method consistently outperforms BN and other baselines, while the advantage is more significant when the batch size is small and the heterogeneity degree is higher. Notably, our method works well even with batch size set to 1, while the vanilla BN becomes inapplicable under this scenario. This enables our method to be used when the computational resource is limited on the client side.

Experiments with Various Networks We confirm the effectiveness of PN across different network architectures and capacities. As shown in Table 3, we conduct experiments with VGG16, ResNet18 and ResNet50. We fix the heterogeneity degree α to be 0.08 and trial with batch sizes of $\{1, 2, 4, 8\}$. We observe consistent performance advantage over the baselines for different networks. Interestingly, one may notice that networks with higher capacities do not necessarily lead to better performances, which is opposite to standard training. This phenomenon is also reported in [5], which may be due to clients' local over-fitting or further drifting from each other in the parameter space. However, PN still demonstrates superior performances compared with other normalization techniques.

5.3. Federated Learning with Large Batch Size

We now extend our PN to NPN to enable stronger generalization ability for training with large batch sizes. The results are demonstrated in Table 4. As can be seen, when training with large batch sizes, PN is outperformed by BN due to the lack of stochastic uncertainty in the normalization layer. On the other hand, NPN remedies this issue by injecting artificial noise into the statistics estimation, which in turn leads to better test performance. The training progress is also visualized in Figure 2(a). We can see that although the training loss of NPN is larger than that of BN, its testing loss is significantly smaller than BN. This demonstrates that benefiting from the injected noise, NPN can prevent the training process from over-fitting more effectively than BN.

Method	Loss	Top-1	Top-5	Top-10
BRN	0.44	93.21	97.52	98.83
BRN + noise	0.29	93.60	97.83	99.02

Table 2. Batch Renormalization + noise.

Finally, we point out that there are only a small number of participants in this scenario, which makes the trained model less prone to the adverse impact of heterogeneity and thus shows better performance than the previous experiments with more participating clients (Table 1).

5.4. Abative Experiments

Noise Level Sensitivity. We evaluate the sensitivity of NPN to the injected noise level. We conduct experiments with VGG-16 on CIFAR-10. We set the batch size to 2048 and vary the variance of the noise injected into NPN in $\{0.5, 0.1, 0.01, 0.001\}$. The results are presented in Figure 2(b). It shows that the test loss increases gradually and slowly when we vary the variance of the noise from 0.5 to

Normalization	batch size (VGG-16)				batch size (ResNet-18)				batch size (ResNet-50)			
	1	2	4	8	1	2	4	8	1	2	4	8
w/o	71.83	79.98	82.02	83.22	75.48	79.94	80.21	81.49	65.59	72.33	77.80	79.94
IN	35.47	68.93	74.03	78.55	63.53	69.36	70.68	71.35	60.37	64.65	68.28	69.10
GN	60.55	78.82	78.53	82.54	66.03	70.44	72.59	75.13	60.63	68.72	70.73	71.25
BN	NA	78.65	82.09	82.27	NA	78.55	79.64	81.94	NA	72.39	78.20	81.83
PN	**78.89**	**82.22**	**85.13**	**85.45**	**80.20**	**83.97**	**84.53**	**84.67**	**67.72**	**73.63**	**79.59**	**83.54**

Table 3. Results of different normalization methods on CIFAR-10. PN consistently outperforms other methods for different ntwork architectures and batch sizes.

Normalization	CIFAR-10				CIFAR-100			
	Loss	Top-1	Top-2	Top-3	Loss	Top-1	Top-2	Top-3
w/o	0.54	92.31	96.91	98.20	2.77	70.34	87.96	92.11
IN	0.46	92.45	97.00	98.31	2.54	70.68	88.49	92.33
GN	0.43	92.92	97.14	98.52	2.23	71.35	88.93	93.10
BN	0.39	93.13	97.66	98.87	2.05	72.21	90.28	93.60
PN	0.41	93.10	97.78	98.97	1.85	72.15	90.07	93.89
NPN	**0.32**	**93.66**	**98.05**	**99.12**	**1.22**	**72.74**	**91.47**	**94.90**

Table 4. Results of different normalization methods on CIFAR-10 / CIFAR-100 with large batch size.

0.001. Therefore, it is easy to select a suitable noise variance that falls within a reasonable range.

Artificial Noise for Other Normalization Methods. We demonstrate that the artificially injected noise is also able to improve existing normalization methods with limited stochastic uncertainty. We take batch renormalization (BRN) [16] as an example and conduct experiments on CIFAR-10 with VGG-16. For simplicity and excluding other factors, we conduct this experiment with regular training instead of FL. We inject noise sampled from $\mathcal{N}(0, 0.1)$ to the moving averages μ and σ of BRN as we do for NPN. The results are given in Table 2. It shows that with the injected noise, the performance of BRN can be improved effectively. Therefore, the idea of data-independent noise injection is a very general regularization technique that can be used to improve existing normalization methods.

5.5. Supervised Learning with PN

Intuitively, PN can also be used in non-FL scenarios. We apply PN to supervised learning and conduct erperiments on two image classification benchmarks (i.e., CIFAR-10, CIFAR-100) with VGG-16 and ResNet to evaluate the effectiveness of PN in supervised learning. The batch sizes of all the methods are set to 128. Table 5 presents the test loss and accuracy of the final learned models. From Table 5 we can see that our NPN can achieve much lower test loss than BN and comparable (if no better than) test accuracy with BN on all the three models. We can also observe that NPN can achieve a noticeable improvement over PN, which demonstrates that artificially injected data independent noise can be used to reinforce the performance of the normalization methods with limited stochastic uncertainty.The comparable performance of BN and NPN implies that the effect of BN can be decomposed into two parts: one is population normalization, and the other is the stochastic uncertainty containing in the mini-batch statistics.

Table 5. Results of different normalization methods on image classification benchmarks

Network	Normalization	Loss	Accuracy
VGG-16	BN	0.39	92.31
	PN	0.41	93.63
	NPN	**0.28**	**93.66**
ResNet-18	BN	0.46	93.76
	PN	0.45	93.97
	NPN	**0.38**	**94.26**
ResNet-56	BN	0.33	93.05
	PN	0.42	92.56
	NPN	**0.25**	**93.99**

6. Conclusions

We propose a learning-based normalization technique, Population Normalization (PN), which reduces the reliance on training batch size and improves robustness against data heterogeneity in federated learning (FL). Extensive experiments demonstrate that PN outperforms existing normalization techniques by a significant margin. We further develop Noisy Population Normalization (NPN), which injects artificial noise into the learnt normalization statistics to enhance generalization under large batch size conditions. We believe that our work will offer valuable insights for the design and development of more stable and adaptable normalization techniques for FL.

7. Acknowledgments

This work was supported by the National Nature Science Foundation of China (62472097) and AI for Science Foundation of Fudan University (FudanX24AI028). The computations in this research were performed on the CFFF platform of Fudan University.

References

[1] Sanjeev Arora, Kaifeng Lyu, and Zhiyuan Li. Theoretical analysis of auto rate-tuning by batch normalization. In *7th International Conference on Learning Representations, ICLR 2019*, 2019. 5

[2] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016. 1, 2

[3] Tobias Bernecker, Annette Peters, Christopher L. Schlett, Fabian Bamberg, Fabian J. Theis, Daniel Rueckert, Jakob Weiss, and Shadi Albarqouni. Fednorm: Modality-based normalization in federated learning for multi-modal liver segmentation. *ArXiv*, abs/2205.11096, 2022. 3

[4] Yongqiang Cai, Qianxiao Li, and Zuowei Shen. A quantitative analysis of the effect of batch normalization on gradient descent. In *International Conference on Machine Learning*, pages 882–890, 2019. 3

[5] Hong-You Chen and Wei-Lun Chao. Fedbe: Making bayesian model ensemble applicable to federated learning. In *ICLR*, 2021. 3, 6, 7

[6] Wikipedia contributors. Kurtosis. Online; accessed 2-June-2020, 2020. Wikipedia, The Free Encyclopedia. 5

[7] Hadi Daneshmand, Amir Joudaki, and Francis Bach. Batch normalization orthogonalizes representations in deep random networks. In *Advances in Neural Information Processing Systems*, 2021. 5

[8] Jian-Hui Duan, Wenzhong Li, and Sanglu Lu. Feddna: Federated learning with decoupled normalization-layer aggregation for non-iid data. In *Machine Learning and Knowledge Discovery in Databases. Research Track*, pages 722–737, Cham, 2021. Springer International Publishing. 3

[9] Aiden Durrant, Milan Markovic, David Matthews, David May, Jessica Enright, and Georgios Leontidis. The role of cross-silo federated learning in facilitating data sharing in the agri-food sector. *Computers and Electronics in Agriculture*, 193:106648, 2022. 2, 4

[10] Filip Hanzely, Slavomír Hanzely, Samuel Horváth, and Peter Richtárik. Lower bounds and optimal algorithms for personalized federated learning. *Advances in Neural Information Processing Systems*, 33:2304–2315, 2020. 3

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 2, 6

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 1

[13] Kevin Hsieh, Amar Phanishayee, Onur Mutlu, and Phillip Gibbons. The non-IID data quagmire of decentralized machine learning. In *Proceedings of the 37th International Conference on Machine Learning*, pages 4387–4398. PMLR, 2020. 3

[14] Kevin Hsieh, Amar Phanishayee, Onur Mutlu, and Phillip B. Gibbons. The non-iid data quagmire of decentralized machine learning, 2020. 1

[15] Chao Huang, Jianwei Huang, and Xin Liu. Cross-silo federated learning: Challenges and opportunities. *arXiv preprint arXiv:2206.12949*, 2022. 2, 4

[16] Sergey Ioffe. Batch renormalization: Towards reducing minibatch dependence in batch-normalized models. In *Advances in neural information processing systems*, pages 1945–1953, 2017. 1, 2, 6, 8

[17] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 448–456, Lille, France, 2015. PMLR. 1, 2

[18] Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank J Reddi, Sebastian U Stich, and Ananda Theertha Suresh. Scaffold: Stochastic controlled averaging for on-device federated learning. In *ICML*, 2020. 3

[19] Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016. 1, 3

[20] Viraj Kulkarni, Milind Kulkarni, and Aniruddha Pant. Survey of personalization techniques for federated learning. In *2020 Fourth World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4)*, pages 794–797. IEEE, 2020. 3

[21] Qinbin Li, Bingsheng He, and Dawn Song. Practical one-shot federated learning for cross-silo setting. *arXiv preprint arXiv:2010.01017*, 2020. 2, 4

[22] Qinbin Li, Bingsheng He, and Dawn Song. Model-contrastive federated learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10713–10722, 2021. 3

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

[24] Tian Li, Shengyuan Hu, Ahmad Beirami, and Virginia Smith. Ditto: Fair and robust federated learning through personalization. In *International Conference on Machine Learning*, pages 6357–6368. PMLR, 2021. 3

[25] Xiaoxiao Li, Meirui Jiang, Xiaofei Zhang, Michael Kamp, and Qi Dou. Fed{bn}: Federated learning on non-{iid} features via local batch normalization. In *International Conference on Learning Representations*, 2021. 2, 3, 6

[26] Xiangru Lian and Ji Liu. Revisit batch normalization: New understanding and refinement via composition optimization. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 3254–3263, 2019. 3

[27] Wei Yang Bryan Lim, Nguyen Cong Luong, Dinh Thai Hoang, Yutao Jiao, Ying-Chang Liang, Qiang Yang, Dusit

Niyato, and Chunyan Miao. Federated learning in mobile edge networks: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, 22(3):2031–2063, 2020. 3

[28] Tao Lin, Lingjing Kong, Sebastian U Stich, and Martin Jaggi. Ensemble distillation for robust model fusion in federated learning. *Advances in Neural Information Processing Systems*, 33:2351–2363, 2020. 3, 6

[29] Ping Luo, Xinjiang Wang, Wenqi Shao, and Zhanglin Peng. Towards understanding regularization in batch normalization. *arXiv preprint arXiv:1809.00846*, 2018. 3, 5

[30] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017. 3, 6

[31] Tim Salimans and Durk P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in neural information processing systems*, pages 901–909, 2016. 2

[32] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? In *Advances in Neural Information Processing Systems*, pages 2483–2493, 2018. 3

[33] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015. 6

[34] Sebastian U Stich. Local sgd converges fast and communicates little. In *International Conference on Learning Representations*, 2018. 3

[35] Canh T Dinh, Nguyen Tran, and Josh Nguyen. Personalized federated learning with moreau envelopes. *Advances in Neural Information Processing Systems*, 33:21394–21405, 2020. 3

[36] Mattias Teye, Hossein Azizpour, and Kevin Smith. Bayesian uncertainty estimation for batch normalized deep networks. In *International Conference on Machine Learning*, pages 4907–4916, 2018. 5

[37] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016. 1, 2, 6

[38] Jianyu Wang, Qinghua Liu, Hao Liang, Gauri Joshi, and H Vincent Poor. Tackling the objective inconsistency problem in heterogeneous federated optimization. *Advances in neural information processing systems*, 33:7611–7623, 2020. 3

[39] Xiaoxia Wu, Edgar Dobriban, Tongzheng Ren, Shanshan Wu, Zhiyuan Li, Suriya Gunasekar, Rachel Ward, and Qiang Liu. Implicit regularization of normalization methods. *arXiv preprint arXiv:1911.07956*, 2019. 3

[40] Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 3–19, 2018. 1, 2, 3, 6

[41] Jianhang Xiao, Chunhui Du, Zijing Duan, and Wei Guo. A novel server-side aggregation strategy for federated learning in non-iid situations. In *2021 20th International Symposium on Parallel and Distributed Computing (ISPDC)*, pages 17–24. IEEE, 2021. 3

[42] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(2):1–19, 2019. 1, 3

[43] Yousef Yeganeh, Azade Farshad, Nassir Navab, and Shadi Albarqouni. Inverse distance aggregation for federated learning with non-iid data. In *Domain Adaptation and Representation Transfer, and Distributed and Collaborative Learning*, pages 150–159. Springer, 2020. 3