FedUV: Uniformity and Variance for Heterogeneous Federated Learning

Supplementary Material

1. FedUV Pseudocode

Algorithm 1 PyTorch-like pseudocode of FedUV

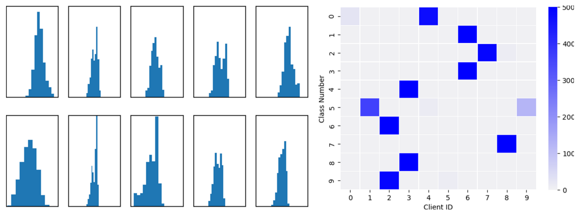
```
# n_classes: number of classes
# $f_\theta$: *local model* (outputs predictions $\hat{y}$ and representations of encoder $z$)
# $\mu$ := 1 (strength of hyperspherical uniformity regularization)
# $\lambda$ := n_classes / 5 (strength of variance regularization)

# Cross-entropy loss
CELoss = torch.nn.CrossEntropyLoss()

# Softmax function
softmax = torch.nn.Softmax(dim=1)

# dimension-wise probability distribution of ideal batch
$c$ = torch.eye(n_classes).std(dim=0).mean().item()

for $x, y$ in mini_batch:
    $\hat{y}$, $z$ = $f_\theta(x)$

    # Cross-entropy loss
    ce_loss = CELoss($\hat{y}, y$)

    # Hypershperical uniformity regularization
    pdist_z = torch.pdist($z$, p=2).pow(2)
    sigma = torch.median(pdist_z[pdist_z != 0])
    u_loss = pdist_z.mul(-1/sigma).exp().mean()

    # Variance regularization
    $\hat{y}$ = F.one_hot($\hat{y}$, num_classes=n_classes)
    $\hat{p}$ = softmax($\hat{y}$.float())
    v_loss = torch.mean(F.relu($c$ - $\hat{p}$.std(dim=0)))

    # Total loss
    loss = (ce_loss + $\mu$ * u_loss + $\lambda$ * v_loss)

    # Optimization
    loss.backward()
    optimizer.step()
```

2. Details on dataset

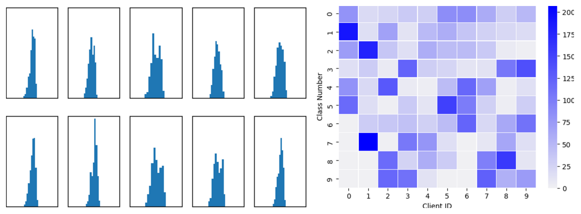
There are two main settings across our six datatsets. The first setting is label-shift, in which clients hold data of unbalanced classes, and the second setting is feature-shift, in which clients hold data from unbalanced features. The label-shift setting is simulated through the Dirichlet distribution, as is common in many FL studies. In the Dirichlet distribution, the α parameter influences the shape and concentration. We create a D-vector for each client, defined by this distribution, and use the distribution to represent data proportions per class. The feature-shift setting is not simulated, as we use real data that come from different sources. We verify that there are distributional shifts.

We show the feature and label distributions for included all datasets. The feature distribution is shown as a histogram of the average pixel value of each sample for each client. We set $\kappa = 10$ for the label-shift datasets, and $\kappa = 4$ for the domain-shift datasets following the available domains. The label distrbition is shown as a heatmap with the x-axis representing the client number, y-axis representing the class number, and the intensity of color showing how many data samples are available.

2.1. Label-shift datasets

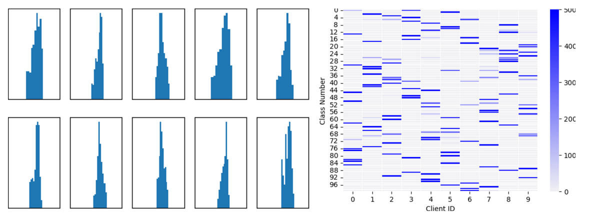


(a) $\alpha = 0.01$

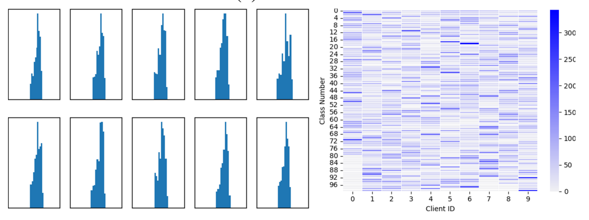


(b) $\alpha = 1.0$

Figure 1. Feature and label distribution of the STL-10 dataset

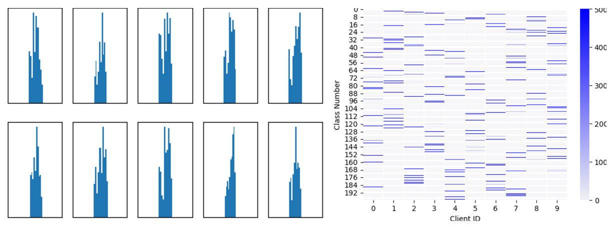


(a) $\alpha = 0.01$

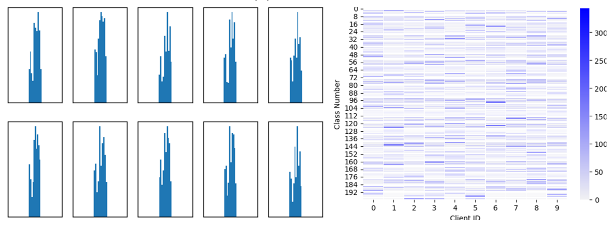


(b) $\alpha = 1.0$

Figure 2. Feature and label distribution of the CIFAR-100 dataset



(a) $\alpha = 0.01$



(b) $\alpha = 1.0$

Figure 3. Feature and label distribution of the Tiny-ImageNet dataset

2.2. Feature-shift datasets

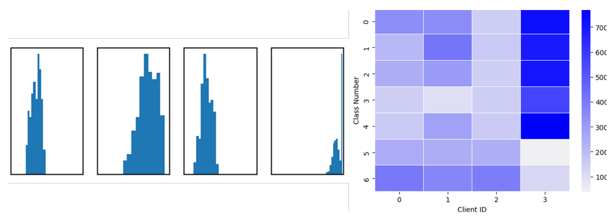


Figure 4. Feature and label distributions of the PACS dataet

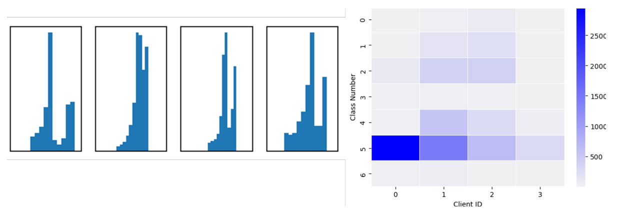


Figure 5. Feature and label distributions of the HAM10000 dataet

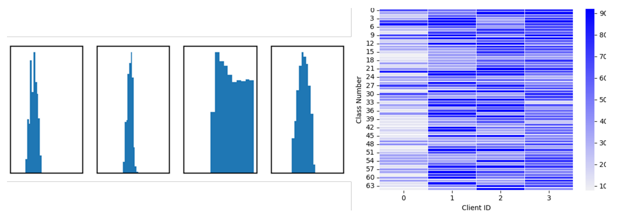


Figure 6. Feature and label distributions of the Office-Home dataet

3. The Small CNN model architecture

Small CNN Architecture
Conv. 3 x 3, 64-BN-ReLu
Conv. 1 x 1, 64-BN-ReLu
Conv. 3 x 3, stride 2, 128-BN-ReLu
Conv. 3 x 3, 128-BN-ReLu
Conv. 1 x 1, 128-BN-ReLu
Conv. 3 x 3, stride 2, 256-BN-ReLu
Conv. 3 x 3, 256-BN-ReLu
Conv. 1 x 1, 256-BN-ReLu
Global average pooling
FC. 256-BN-ReLu x2
Logits

Table 1. CNN architecture on STL-10 and PACS

4. Regularization Strength Tuning

In real-world FL applications, hyperparameter tuning may not be a realistic approach. Every step in hyperparameter tuning requires many local epochs and many aggregation rounds. This is a heavy burden for local devices. Because of this constraint, we train the Small CNN model (Table 1) on the CIFAR-10 dataset and report the best hyperparameters for each method based on our custom validation set (90-10 split from original training dataset). Table 2 and Table 3 show the results of the hyperparameter tuning on $\alpha := 0.1$ and $\alpha := 1.0$ on the CIFAR-10 dataset, respectively. Note that when the strength of regularization becomes too strong for any method, validation accuracy drops as convergence becomes more difficult.

Method	α	μ	λ	Acc.
FedAvg	0.01	-	-	34.7%
FedProx	0.01	0.001	-	35.5%
FedProx	0.01	0.01	-	35.4%
FedProx	0.01	0.1	-	32.8%
FedProx	0.01	1.0	-	32.2%
MOON	0.01	0.1	-	35.3%
MOON	0.01	1.0	-	35.7%
MOON	0.01	5.0	-	31.1%
MOON	0.01	10.0	-	32.0%
FedUV	0.01	0.1	n_classes/8	45.3%
FedUV	0.01	0.1	n_classes/4	45.1%
FedUV	0.01	0.1	n_classes/1	44.4%
FedUV	0.01	0.5	n_classes/8	45.4%
FedUV	0.01	0.5	n_classes/4	37.7%
FedUV	0.01	0.5	n_classes/1	35.8%
FedUV	0.01	1.0	n_classes/8	45.2%
FedUV	0.01	1.0	n_classes/4	45.7%
FedUV	0.01	1.0	n_classes/1	43.1%

Table 2. Hyperparameter tuning for methods on CIFAR-10 (α := 0.01)

Method	α	μ	λ	Acc.
FedAvg	1.0	-	-	72.4%
FedProx	1.0	0.001	-	73.5%
FedProx	1.0	0.01	-	71.4%
FedProx	1.0	0.1	-	68.5%
FedProx	1.0	1.0	-	67.5%
MOON	1.0	0.1	-	72.5%
MOON	1.0	1.0	-	74.7%
MOON	1.0	5.0	-	67.5%
MOON	1.0	10.0	-	66.3%
FedUV	1.0	0.1	n_classes/8	74.0%
FedUV	1.0	0.1	n_classes/4	74.5%
FedUV	1.0	0.1	n_classes/1	73.1%
FedUV	1.0	0.5	n_classes/8	75.0%
FedUV	1.0	0.5	n_classes/4	75.3%
FedUV	1.0	0.5	n_classes/1	73.1
FedUV	1.0	1.0	n_classes/8	75.1%
FedUV	1.0	1.0	n_classes/4	74.7%
FedUV	1.0	1.0	n_classes/1	74.3%

Table 3. Hyperparameter tuning for methods on CIFAR-10 (α := 1.0)

5. Additional results

Table 4 shows results for FedProto, FedDyn, Scaffold. Note that these methods also change the aggregation process.

Method	STL (.01)	STL (1.0)	CIFAR (.01)	CIFAR (1.0)	Tiny (.01)	Tiny (1.0)	PACS	HAM	Office
FedAvg	27.6	68.5	51.6	58.3	37.5	40.2	61.9	73.3	42.2
SCAFFOLD	24.7	**68.7**	50.3	**60.8**	39.3	43.1	60.1	73.7	42.1
FedDyn	22.8	65.7	50.7	59.7	38.3	41.3	63.1	72.4	41.1
FedProto	30.1	66.2	49.5	54.2	36.8	38.5	56.7	71.6	42.1
FedUV	**30.4**	68.5	**55.7**	59.1	**40.3**	**43.2**	**65.9**	**73.9**	**45.4**

Table 4. Additional Baselines

FedProto transfers class prototype features, while SCAFFOLD and FedDyn uses an additional term in aggregation. FedUV, MOON, FedProx, and Freeze follow FedAvg aggregation. These baselines do not change our conclusion. FedUV is unique in that it does not rely on the global model for regularization, rather focusing on emulating the IID setting regardless of the current non-IIDness. The global model is not a good source of regularization when data is highly non-IID.

Model	FedAvg				FedUV			
	STL	Tiny	PACS	Office	STL	Tiny	PACS	Office
Small CNN	27.6	23.4	61.9	41.2	30.4	24.2	65.9	42.9
ResNet-18	30.2	36.9	54.8	48.7	29.6	38.4	59.8	49.5
ResNet-50	27.2	37.5	47.2	42.2	28.9	40.3	49.3	45.4

Table 5. Model ablation

Ablations for architectures and datasets are shown in Table 5. With the exception of Tiny Imagenet, most datasets are quite small. This small data size is likely the reason the larger ResNet models underperform. Nevertheless, FedUV performs better than FedAvg across all but one (STL — ResNet-18) settings.

Method	STL-10 ($\alpha = 0.01$)			STL-10 ($\alpha = 1.0$)		
	λ=1.0	λ=2.5	λ=5.0	λ=1.0	λ=2.5	λ=5.0
No Hinge	29.7	29.7	28.4	67.9	67.6	68.1
Squared Hinge	29.2	27.1	26.8	67.5	67.8	67.3
Hinge (FedUV)	30.1	30.4	31.1	66.6	67.1	67.9

Table 6. Performance across different regularization strength λ

Table 6 shows results for no hinge, squared hinge, and linear hinge (FedUV). Our goal with hinge loss is to remove dimensions that have large variance (negative values) so it does not overpower dimensions with low variance. *No hinge* does not remove negative values and *Squared hinge* turns negative values into positive. This may explain the drop in performance for α=0.01.