Digital-Twin–Driven Semi-Supervised GAN Federated Learning

Sugeet Sood
Vibhor Barguje

18 May 2025

Outline

Motivation

- Edge devices generate sensitive data requiring local processing.
- Centralized training compromises privacy and incurs latency.
- Federated Learning (FL) offers a solution but faces non-IID data and device heterogeneity.
- Semi-Supervised GANs (SGANs) leverage unlabeled data but need stabilization.

Contributions

1. **FedProx Integration:** Proximal regularisation for heterogeneity handling.
2. **SGAN Enhancements:** Feature Dropout and Mini-Batch Discrimination for robust training.
3. **Loss-Guided Selection:** Prioritize clients by loss with 10% exploration.

Federated Learning Essentials

- Clients compute local updates Δw_k on private data.
- Server aggregates: $w^{(t+1)} = w^{(t)} + \sum_k \frac{n_k}{\sum_j n_j} \Delta w_k$ [4].
- Challenges: statistical heterogeneity, system heterogeneity, privacy.

Semi-Supervised GAN (SGAN)

- Discriminator outputs K real classes $+ 1$ fake class [**?**].
- Combines supervised loss on scarce labels with unsupervised real/fake losses.
- Feature Matching aligns real and generated features.

FedProx & Client Selection

- **FedProx:** Adds $\frac{\mu}{2}\|\theta - \theta^{(t)}\|^2$ to local objective [5].
- **Loss-Guided Selection:** Rank by last-round loss $ar\ell_D$, reserve 10% for random exploration [**?**].

Random Client Sampling Algorithm

basicstyle= for t in 1..T:
$\mathrm{S}_t < -randomsubsetofclientsbroadcastw^(t)parallelforkinS_t : w_k = LocalTrain(w^(t))delta_{wk} = w_k - w^(t)w^(t+1) = w^(t) + Average(delta_{wk})$
Simple but ignores data utility and client speed.

Layered Architecture

Edge Server

- Selection Module: random / loss-guided.
- Aggregation Engine: FedAvg / FedProx.
- Monitoring: Prometheus Grafana.

Clients

- Local SGAN + FedProx training.
- Metrics reporting: losses, samples.
- Fault handling: timeouts, NaNs.

Digital Twin: Flask webapp for live inference updates.

Generator Architecture

- Input: $z \sim \mathcal{N}(0, I)^{100}$.
- MLP: 100-512-ReLU, 512-1024-BN-ReLU, 1024-3072-Tanh.
- Output: $3 \times 32 \times 32$ image in [-1,1].

Discriminator Architecture

- FC 3072-512-256 with LeakyReLU.
- Feature Dropout (p=0.5).
- Mini-Batch Discrimination: 50 kernels.
- Output: $(256 + 50) \rightarrow K + 1$ logits.

Loss Components

- Supervised: smoothed CE on labels.
- Unsupervised Real: $-\log(1 - p_{fake}(x))$.
- Unsupervised Fake: $-\log p_{fake}(G(z))$.
- Adversarial (G): $-\log(1 - p_{fake}(G(z)))$.
- Feature Matching: $\|E[\phi(x)] - E[\phi(G(z))]\|^2$.
- FedProx: $\frac{\mu}{2}\|\theta - \theta^{(t)}\|^2$.

Local Training Pseudocode

basicstyle= function LocalTraining(wG,wD): G.load(wG); D.load(wD) G0,D0 = G.clone(), D.clone() for epoch=1..E: for batch in data: D update: real, sup, fake + prox D.backward(); D.step() G update: adv + FM + prox G.backward(); G.step() return G.weights()-wG, D.weights()-wD Robust to NaNs and timeouts.

Loss-Guided Selector

basicstyle= function SelectClients(hist,M,alpha): explo = ceil(alpha*M) explorees = random$_c hoose(explo) ranked = sort(hist.loss, desc) selected = ranked[: M - explo] return explorees U selected$ Balances exploitation of high-loss clients with exploration.

Dataset Splits

- CIFAR-10 subset: 2 000 images (seed=42).
- Client 0: 100 labeled; Clients 1–3: 633 unlabeled each.
- Normalized to [-1,1]; no augmentation.

Key Metrics

- **Convergence:** 1.3x faster reduction in discriminator loss vs. random sampling.
- **Generation Quality:** 7.4-point FID improvement over random sampling.
- **Efficiency:** Comparable communication per round.

Code Data Pipeline

- **DataLoader & Splits:** CIFAR-10 $\rightarrow$ 2 000 samples, fixed seed, 100 labeled / 633 unlabeled.
- **Client Class:** `get_data_loader()`, `has_labels()`, empty-dataset fallbacks.
- **Reproducibility:** torch/NumPy seed, GPU/CPU toggle, DataLoader workers.

Training Metrics Tracking

- **Hyperparameters:** smoothing=0.1, clip_D=1.0, clip_G=1.0, updates_per_D=1.
- **Client-Returned Metrics:** avg_loss_D, avg_loss_G, samples_processed, epochs_performed.
- **Server Graphing:** accumulate server-side avg losses per round; plot at end.
- **Sample Generation:** fixed noise $\rightarrow$ 8 images each round for visual monitoring.

Future Work

1. Scale-up: 50+ distributed clients in WAN settings.
2. Privacy: DP-SGD and secure aggregation integration.
3. Optimization: Efficient MBD on IoT hardware.
4. Deployment: Real-world sensor network applications.

References I

[1] I. Goodfellow *et al.*, "Generative Adversarial Nets," NeurIPS, 2014.

[2] T. Salimans *et al.*, "Improved Techniques for Training GANs," NeurIPS, 2016.

[3] N. Srivastava *et al.*, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," JMLR, 2014.

[4] B. McMahan *et al.*, "Communication-Efficient Learning of Deep Networks from Decentralized Data," AISTATS, 2017.

[5] T. Li *et al.*, "Federated Optimization in Heterogeneous Networks," MLSys, 2020.