Federated Learning on Multiple Data Sources

Lakshay Tyagi, Raviteja Chukkapalli

I. Introduction

DATASETS are a central component of Machine Learning, Collecting datasets at a central location to perform learning is not always possible, think of data privacy and data protection laws. In such situations we can perform learning at the source of data, this is the setting for federated learning. Through this work, we provide a quick introduction to federated learning and go over a few works to understand the landscape of problems and innovations happening in Federated Optimization. In sections 1 and 2, we formalize the federated learning setting and discuss Federated Averaging. In sections 3,4,5 we introduce Agnostic Federated Learning, conduct experiments and discuss works that build on top of agnostic federated learning. We conclude in section 6 by proposing a variation of the agnostic federated averaging algorithm.

A. Federated Learning

Federated Learning brings learning algorithms to data instead of the other way around. In a federated learning setting, there can be many sources of data generation for example mobile devices generating social media usage data. In Federated Learning terminology, These are also called clients. Since we want to learn a single model for all these clients federated optimization also involves communicating client models to a server that aggregates them. Optimization methods designed for federated learning have to address a few additional challenges compared to their standard counterparts like computation and communication constraints, data heterogeneity, client availability, etc.

1) Problem Formulation: The objective function of federated learning can be defined as

$$F(x) = \mathbb{E}_{i\sim\mathcal{P}}[F_i(x)], \text{ where, } F_i(x) = \mathbb{E}_{S\sim\mathcal{D}_i}[f_i(x,S)] \tag{1}$$

where $x \in \mathbb{R}^d$ represents the model parameters, $\mathcal{P}$ represents the distribution over clients, $\mathcal{D}_i$ represents the data distribution at client i and S the sample drawn from the distribution. $f_i(x,S)$ represents the local objective function of client client i evaluated on sample S drawn from $\mathcal{D}_i$.

Employing empirical risk minimization (ERM) to solve 1, we get

$$F^{\text{ERM}}(x) = \Sigma_{i=1}^{M} p_i F_i^{\text{ERM}}(x) \text{ where, } F^{\text{ERM}}(x) = \frac{1}{|\mathcal{D}_i|}\Sigma_{S\in\mathcal{D}_i} f_i(x,S) \text{ and,} \Sigma_{i=1}^{M} p_i = 1 \tag{2}$$

Where M denotes the total number of clients. Note that choosing p_i's is a design choice for the learning algorithm designers. Setting $p_i = |\mathcal{D}_i|/\Sigma_{i=1}^{M}|\mathcal{D}_i|$ would make the federated ERM similar to gathering all the local datasets at a single destination and performing ERM. We can potentially solve 1 using gradient decent (GD) which looks like $x^{(t+1)} = x^{(t)} - \eta_t \nabla F(x^{(t)})$. The gradient can be written as

$$\nabla F(x) = \nabla \mathbb{E}_{i\sim\mathcal{P}}[F_i(x)] = \mathbb{E}_{i\sim\mathcal{P}}[\nabla F_i(x)] \tag{3}$$

This shows that to use GD-based methods for federated optimization we need only transfer gradients to the server and the server can collate these local gradients to calculate the gradient for the federated learning objective. It is often hard to calculate the exact gradients even for the local models due to the size of the dataset, and in such cases, we can rely on stochastic approximation(SA) to replace the actual gradient with its unbiased stochastic approximation $g_i(x^{(t)})$ such that $\mathbb{E}_{S\in\mathcal{D}_i}[g_i(x^{(t)})] = \nabla F_i(x^{(t)})$. Transferring the gradients at each time step from each client to the server requires a lot of communication costs, hence *local steps* is a common technique employed in federated optimization. Each client updates the local model for τ_i and then sends the model delta to the server $\Delta_i = x_i^{(t,\tau_i)} - x^{(t)}$. Federated averaging is a very popular algorithm in federated optimization that uses both local steps and stochastic approximation.

II. Federated Averaging

A. Algorithm

Federated Averaging proposed by [6] is widely used for federated optimization. The algorithm divides the training process into rounds. At the start of each round, the model is broadcasted to all the clients. Each client performs τ_i local steps of stochastic gradient descent and updates the server with the delta in model parameters. The server then aggregates the client deltas and performs the global update. This process is repeated until the convergence of the global model. Not all clients need to participate in each round, which introduces another level of stochasticity.

Federated Averaging is also known as Local SGD or parallel SGD. It is also being used in distributed training settings.

Survey done as a requirement for the course Foundations of Machine Learning

```
Input: Initial model $\boldsymbol{x}^{(0)}$; ClientOpt, ServerOpt with learning rate $\eta, \eta_s$
for $t \in \{0, 1, \dots, T-1\}$ do
    Sample a subset $\mathcal{S}^{(t)}$ of clients
    for client $i \in \mathcal{S}^{(t)}$ in parallel do
        Initialize local model $\boldsymbol{x}_i^{(t,0)} = \boldsymbol{x}^{(t)}$
        for $k = 0, \dots, \tau_i - 1$ do
            Compute local stochastic gradient $g_i(\boldsymbol{x}_i^{(t,k)})$
            Perform local update $\boldsymbol{x}_i^{(t,k+1)} = \text{ClientOpt}(\boldsymbol{x}_i^{(t,k)}, g_i(\boldsymbol{x}_i^{(t,k)}), \eta, t)$
        end
        Compute local model changes $\Delta_i^{(t)} = \boldsymbol{x}_i^{(t,\tau_i)} - \boldsymbol{x}_i^{(t,0)}$
    end
    Aggregate local changes $\Delta^{(t)} = \sum_{i \in \mathcal{S}^{(t)}} p_i \Delta_i^{(t)} / \sum_{i \in \mathcal{S}^{(t)}} p_i$
    Update global model $\boldsymbol{x}^{(t+1)} = \text{ServerOpt}(\boldsymbol{x}^{(t)}, -\Delta^{(t)}, \eta_s, t)$
end
```

Fig. 1: Federated Averaging from [6] proposes multiple local gradient descent steps combined with stochastic gradients for federated optimization

B. Convergence

In a typical convergence proof, we want to show that our iterates $x^{(t,k)}$ are taking the objective $F(x^{(t,k)})$ closer to the optimal $F(x^*)$. In federated learning, we have multiple local iterates on clients, so we want to ensure all of them are approaching the optimal. To show this for multiple clients scenario the concept of shadow sequence is used. We can define the shadow sequence as

$$\bar{x}^{(t,k)} := \frac{1}{M}\Sigma_{i=1}^{M} x_i^{(t,k)} \implies \bar{x}^{(t,k+1)} = \bar{x}^{(t,k)} - \frac{\eta}{M}\Sigma_{i=1}^{M} g_i(x_i^{(t,k)}) \tag{4}$$

This implies that to prove the convergence we need to show that

$$\mathbb{E}\left[\frac{1}{\tau T}\Sigma_{t=0}^{T-1}\Sigma_{k=1}^{\tau} F(\bar{x}^{(t,k)}) - F(x^*)\right] \leq \text{An upper bound decreasing with } T \tag{5}$$

Under the following assumptions,

1) Each client participates in every round.
2) Each local objective is convex and L-smooth.
3) Each client has access to an unbiased stochastic gradient with uniformly bounded variance in l_2 norm σ^2.
$$\mathbb{E}[g_i(x_i^{(t,k)})|x_i^{(t,k)}] = \nabla F_i(x_i^{(t,k)}),\ \mathbb{E}[||g_i(x_i^{(t,k)}) - \nabla F_i(x_i^{(t,k)})||_2^2 | x_i^{(t,k)}] \leq \sigma^2$$
4) The difference between the local gradient and the global gradient is ζ-uniformly bounded in the l_2 norm.
$$\max_i \sup_x ||\nabla F_i(x_i^{(t,k)}) - \nabla F(x_i^{(t,k)})||_2 \leq \zeta$$
5) The client learning rate is bounded as $\eta \leq \frac{1}{4L}$

It can be shown that (5) can be bounded as (refer to [9] for complete proof)

$$\mathbb{E}\left[\frac{1}{\tau T}\Sigma_{t=0}^{T-1}\Sigma_{k=1}^{\tau} F(\bar{x}^{(t,k)}) - F(x^*)\right] \leq \frac{2LD^2}{\tau T} + \frac{2\sigma D}{\sqrt{M_\tau T}} + \frac{5L^{1/3}\sigma^{2/3}D^{4/3}}{\tau^{1/3}T^{2/3}} + \frac{19L^{1/3}\zeta^{2/3}D^{4/3}}{T^{2/3}} \tag{6}$$

where $D := ||x^{(0,0)} - x^*||$. The first two terms also appear in proof of Synchronous SGD, The last two terms are additional errors from local updates and non-IID data. This concludes that federated averaging indeed converges.

C. Client Drift in FedAvg

Let x^* be the global minima or best possible weights on the combined data of all servers. Then any federated learning algorithm aims to find these best weights. FedAvg minimizes the local objectives and then averages the weights after local steps. This means that FedAvg minimizes the local objective and not the global objective. In general if $F_i(.)$ are the local objectives and $F(.)$ is the global objective FedAvg is doing the following update

$$x^{(t+1)} = \sum_{i=1}^{C} \arg\min_x F_i(x) \tag{7}$$

Where the best global weight can be obtained by the following update

$$x^{(t+1)} = \arg\min_x \sum_{i=1}^{C} F_i(x) \tag{8}$$

Here, $F(.) = \sum_{i=1}^{C} F_i$. The sum of minima is not the same as the minima of the sum and this is the origin of the client drift problem. Over time, as the client model converges to the client minima it drifts away from the global minima. This means that FedAvg has to do many small local steps to avoid overfitting to the local objective which slows down convergence. Many research works have found ways of accelerating convergence by handling client drift. One such paper SCAFFOLD [5] uses control variates to address this issue and significantly speeds up the training of federated learning models.

III. Agnostic Federated Learning

A. Introduction

In practice, it is observed that FedAvg becomes biased toward some of the clients that form the majority of the dataset. In 2 we noted that the p_i's are a design choice for the algorithm designer. FedAvg usually uses $p_i = |\mathcal{D}_i|/\Sigma_{i=1}^{M}|\mathcal{D}_i|$ which can be thought of as assigning uniform weight for each datum(uniform distribution). Is uniform distribution expected at test time? Agnostic Federated Learning proposed by [7] asks the same question. They prove that uniform distribution is not the best choice when assigning probability mass to client distributions/objectives. They then develop a framework to learn a target model that has well-defined performance guarantees on any possible mixture of client distributions at test time.

B. AFL Formulation

Looking at a multi-class classification scenario, The hypothesis h maps each datum to a simplex on y.

$$h : \mathcal{X} \mapsto \Delta y \tag{9}$$

The target distribution $\mathcal{D}_\lambda$ that we are expecting to see at test time can be modeled as an unknown mixture of client distributions.

$$\mathcal{D}_\lambda = \Sigma_{k=1}^{q} \lambda_k \mathcal{D}_k \text{ for some } \lambda \in \Lambda \subseteq \Delta_p \tag{10}$$

where q is the number of clients available for the current run. Following this, the agnostic loss can be defined as

$$\mathcal{L}_{\mathcal{D}_\lambda}(h) = \max_{\lambda \in \Lambda} \mathcal{L}_{\mathcal{D}_\lambda}(h) \text{ and } \mathcal{L}_{\bar{\mathcal{D}}_\lambda}(h) = \max_{\lambda \in \Lambda} \mathcal{L}_{\bar{\mathcal{D}}_\lambda}(h) \tag{11}$$

Where $\mathcal{L}_{\bar{\mathcal{D}}_\lambda}$ is the empirical loss.

C. Analysis

Using cross-entropy loss the authors are able to show that AFL formulation is guaranteed to perform better than using uniform distribution by a constant margin.

$$\mathcal{L}_{\mathcal{D}_\Lambda}(h_{\bar{\mathcal{U}}}) \geq \mathcal{L}_{\mathcal{D}_\Lambda}(h_{\mathcal{D}_\lambda}) + \log \frac{2}{\sqrt{3}} \tag{12}$$

where $h_{\bar{\mathcal{U}}}$ is hypothesis returned by learner using uniform distribution. It can be observed that (12) is independent of sample size or iterations, therefore it is valid even for infinite samples. The authors further define a custom Rademacher complexity measure to understand the hypothesis set.

$$\begin{aligned} \mathfrak{R}_m(\mathcal{G}, \lambda) &= \mathop{\mathbb{E}}_{S_k \sim \mathcal{D}_k^{m_k},\, \sigma} \left[\sup_{h \in \mathcal{H}} \Sigma_{k=1}^{p} \frac{\lambda_k}{m_k} \Sigma_{i=1}^{m_k} \sigma_{k,i} l(h(x_{k,i}), y_{k,i}) \right] \\ \mathfrak{R}_m(\mathcal{G}, \Lambda) &= \max_{\lambda \in \Lambda} \mathfrak{R}_m(\mathcal{G}, \lambda) \end{aligned} \tag{13}$$

The skewness of Λ W.R.T empirical distribution over Δ_p defined as $\bar{\mathbf{m}} = (\frac{m_1}{\mathbf{m}}, \cdots, \frac{m_p}{\mathbf{m}})$ where $\mathbf{m} = \Sigma_{i=1}^{p} m_i$ can be defined as

$$s(\Lambda||\bar{\mathbf{m}}) = \max_{\lambda\in\Lambda} \chi^2(\lambda||\bar{\mathbf{m}}) + 1 \tag{14}$$

Where the chi-squared divergence for two distributions p and q in simplex Δ_p is defined as $\chi^2(p||q) = \Sigma_{k=1}^{p} \frac{(p_k - q_k)^2}{q_k}$. The L_1, ϵ covering number bound can be defined as

$$\Lambda_\epsilon = \underset{\Lambda' \in C(\Lambda,\epsilon)}{\operatorname{argmin}} |\Lambda| \tag{15}$$

The authors were able to give a bound on the expected loss as follows

$$\mathcal{L}_{\mathcal{D}}(h) \leq \mathcal{L}_{\bar{\mathcal{D}}_\Lambda}(h) + 2\Re_m(\mathcal{G},\Lambda) + Ml_1(\mathcal{D},\mathcal{D}_\Lambda) + M\epsilon + M\sqrt{\frac{s(\Lambda||\bar{\mathbf{m}})}{2m}\log\frac{|\Lambda_\epsilon|}{\delta}} \tag{16}$$

where $\mathcal{D}$ is some distribution according to which test samples are drawn and $l_1(\mathcal{D},\mathcal{D}_\Lambda) = \min_{\lambda\in\Lambda} l_1(\mathcal{D},\mathcal{D}_\lambda)$. From the bound (16) it is can be seen that when p is large and $\Lambda = \Delta_p$, then the number of samples per each domain decreases and the skewness parameter increases since $s(\Lambda||\bar{\mathbf{m}}) = \underset{i\leq k\leq p}{max} \frac{1}{m_k}$. This means the bounds become meaningless when the no.of domains increases. On the contrary when λ contains only one distribution and it is the uniform distribution then the skewness becomes 1. This shows that it is not always possible to declare your domains as your client devices. [7] gives a few guidelines for selecting your domains like using a clustering algorithm on your data to identify the domains present in it or using domain expertise. From the above analysis, we can conclude that choosing your domains is an important hyper-parameter introduced by AFL, and from our experiments, we find that it is a little tricky to get it right.

D. Optimization

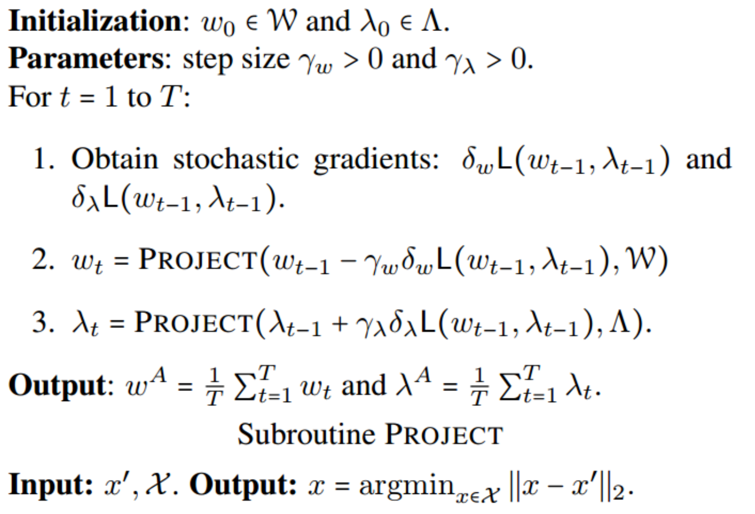
The optimization problem in AFL is over two sets of parameters $h \in \mathcal{H}$ and $\lambda \in \Lambda$. This loss can be written as

$$\begin{aligned} L(w,\lambda) &= \Sigma_{k=1}^{p} \lambda_k L_k(w) \text{ where} \\ L_k(w) &= \frac{1}{m_k} \Sigma_{i=1}^{m_k} l(h(x_{k,i}), y_{k,i}) \end{aligned} \tag{17}$$

This results in a min-max optimization problem

$$\min_{w\in\mathcal{W}} \max_{\lambda\in\Lambda} L(w,\lambda) \tag{18}$$

The authors propose a stochastic gradient-based optimization solution similar to FedAvg to solve the (18) min-max problem. This involves using gradients W.R.T both w_t and λ_t and making perturbations to w, λ in the descending and ascending directions respectively.

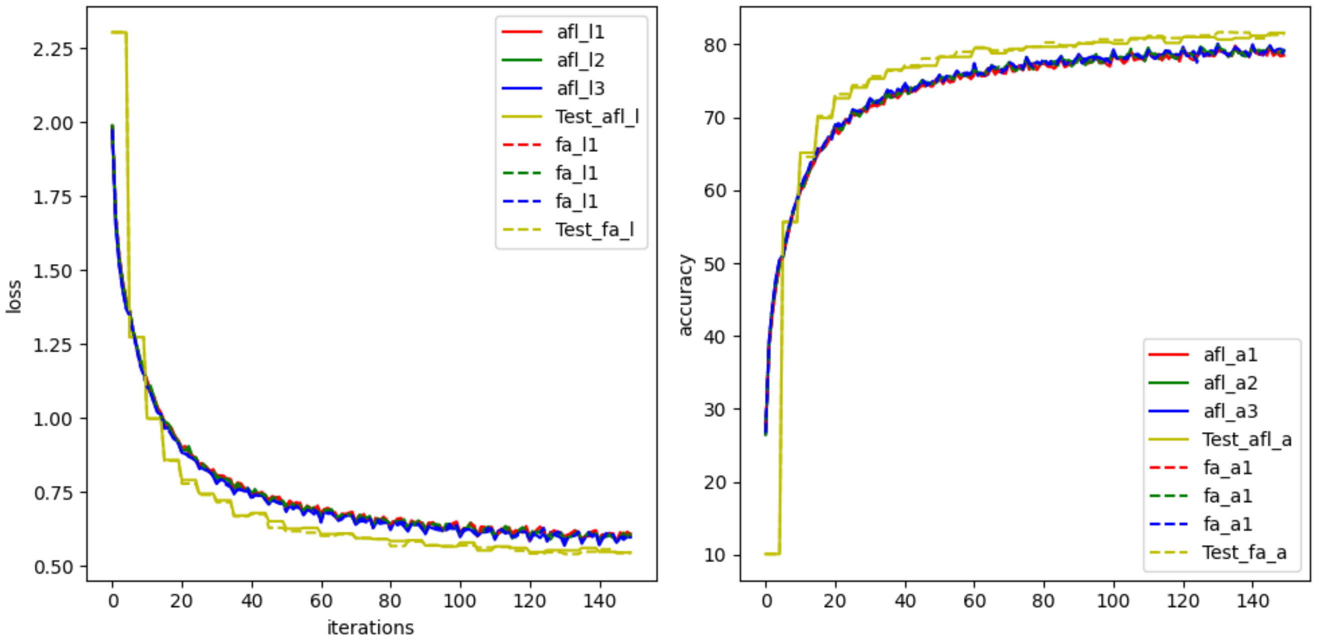


```
Initialization: $w_0 \in \mathcal{W}$ and $\lambda_0 \in \Lambda$.
Parameters: step size $\gamma_w > 0$ and $\gamma_\lambda > 0$.
For $t$ = 1 to $T$:
    1. Obtain stochastic gradients: $\delta_w \mathsf{L}(w_{t-1}, \lambda_{t-1})$ and
       $\delta_\lambda \mathsf{L}(w_{t-1}, \lambda_{t-1})$.
    2. $w_t = \text{PROJECT}(w_{t-1} - \gamma_w \delta_w \mathsf{L}(w_{t-1}, \lambda_{t-1}), \mathcal{W})$
    3. $\lambda_t = \text{PROJECT}(\lambda_{t-1} + \gamma_\lambda \delta_\lambda \mathsf{L}(w_{t-1}, \lambda_{t-1}), \Lambda)$.
Output: $w^A = \frac{1}{T}\sum_{t=1}^{T} w_t$ and $\lambda^A = \frac{1}{T}\sum_{t=1}^{T} \lambda_t$.
                    Subroutine PROJECT
Input: $x', \mathcal{X}$. Output: $x = \operatorname{argmin}_{x\in\mathcal{X}} ||x - x'||_2$.
```

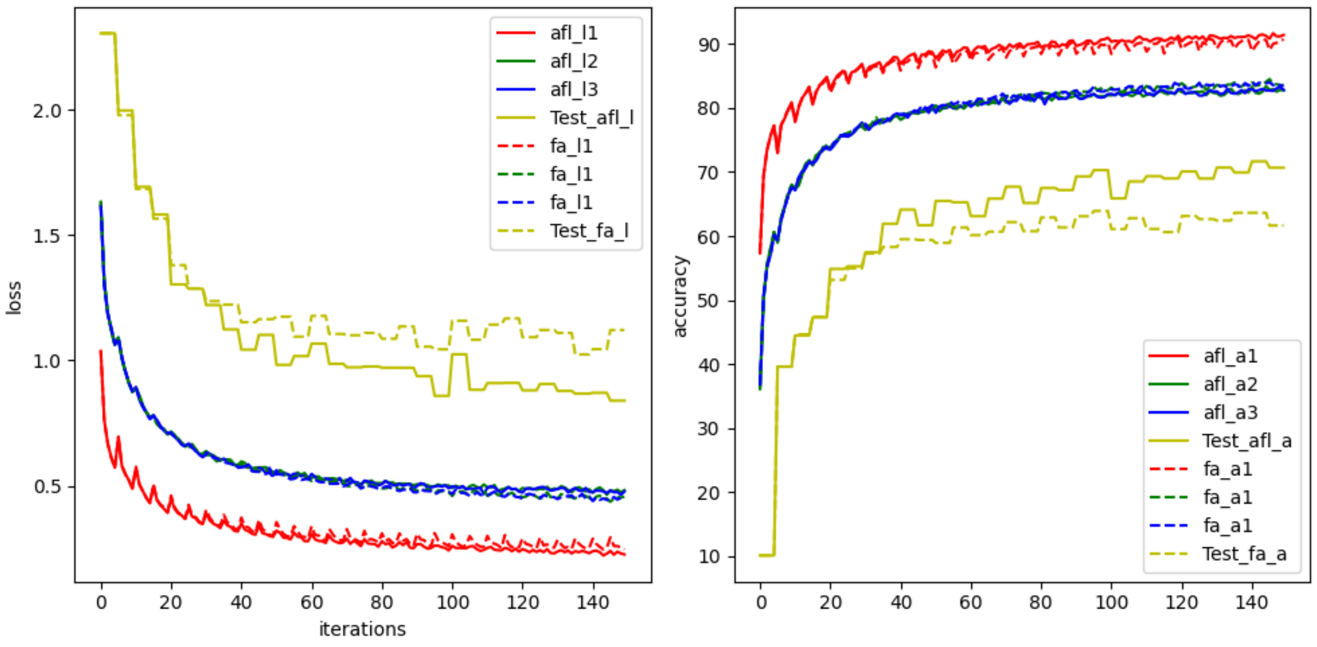
Fig. 2: Agnostic Federated Learning from [7] frames min-max objective and minimizes the loss of worst possible mixture of client distributions(maximization W.R.T λ). Which can be seen in the above algorithm as gradient ascent for λ and descent for w

Interested readers can refer to [7] for the proofs of the above results.

E. Experiments

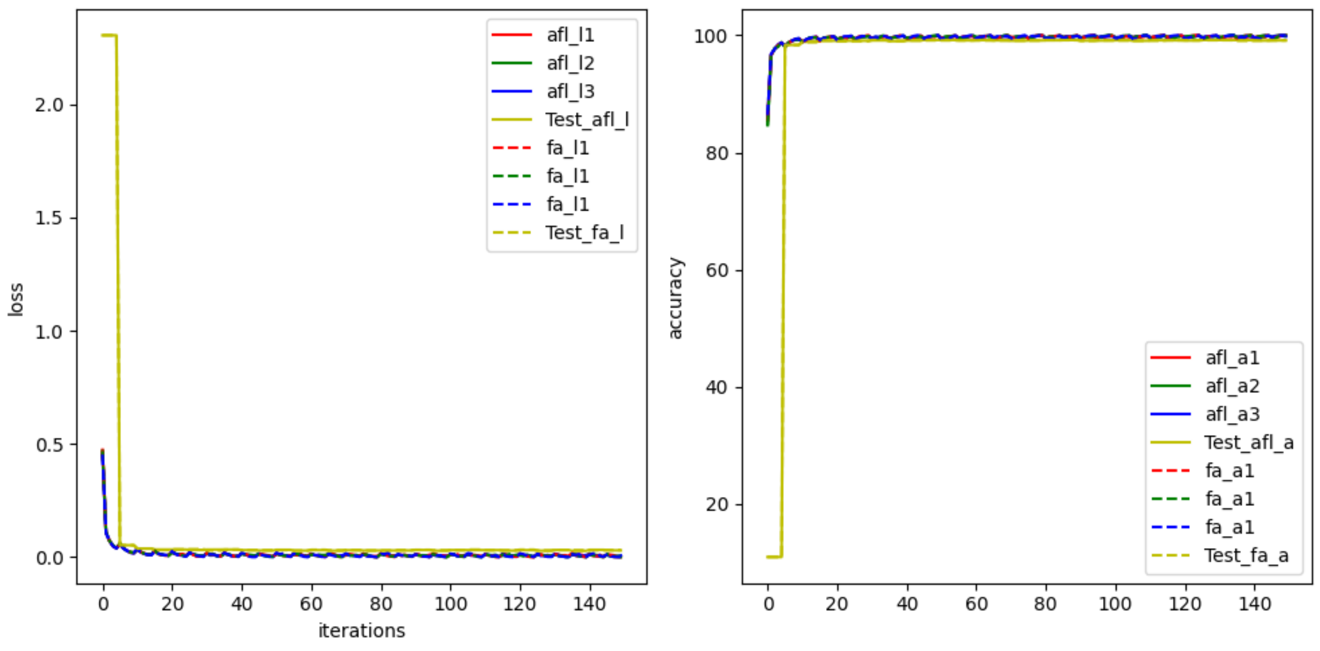


CIFAR-10 I.I.D

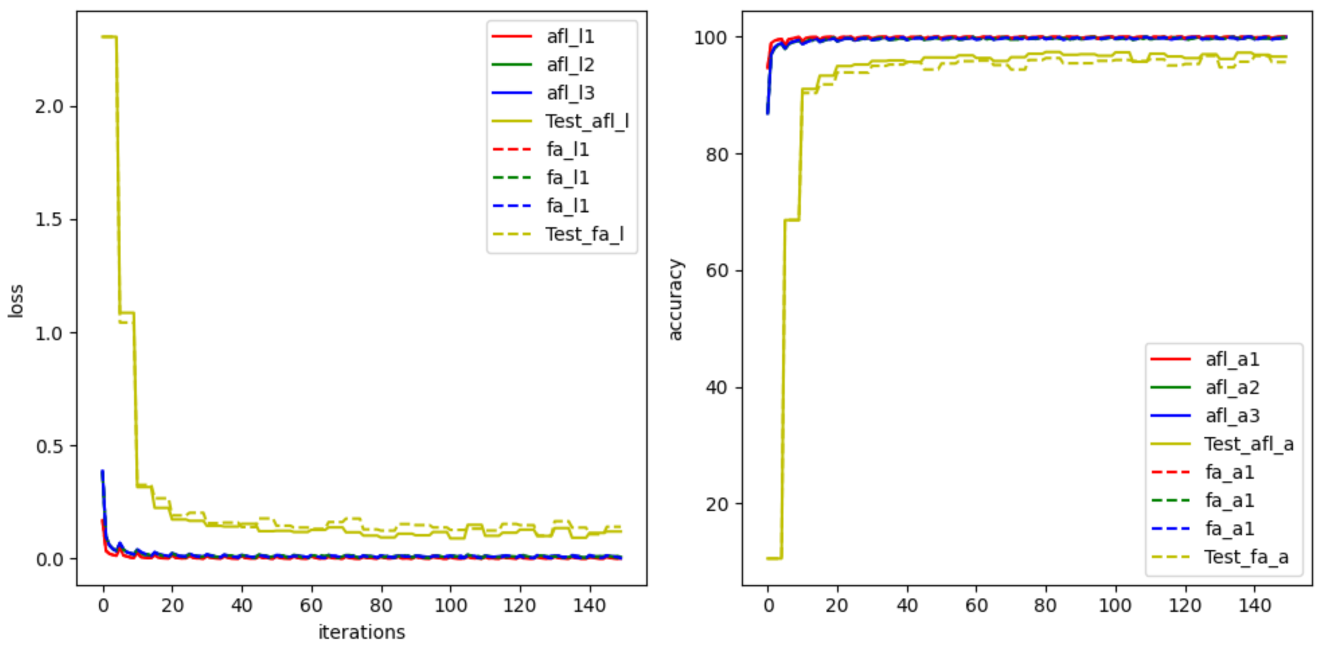


CIFAR-10 Not I.I.D

Fig. 3: Loss and Accuracy of FedAVG and AFL on CIFAR-10 dataset, with three clients and under I.I.D (Test data distribution and distribution at each client is $\mathcal{D}$) vs non-I.I.D (non-overlapping client distributions $\mathcal{D}_i$ and test data is a mixture of client distributions). alf=agnostic-federated-learning, fa=federated-averaging, l1=loss at client 1, a1 = accuracy at client 1

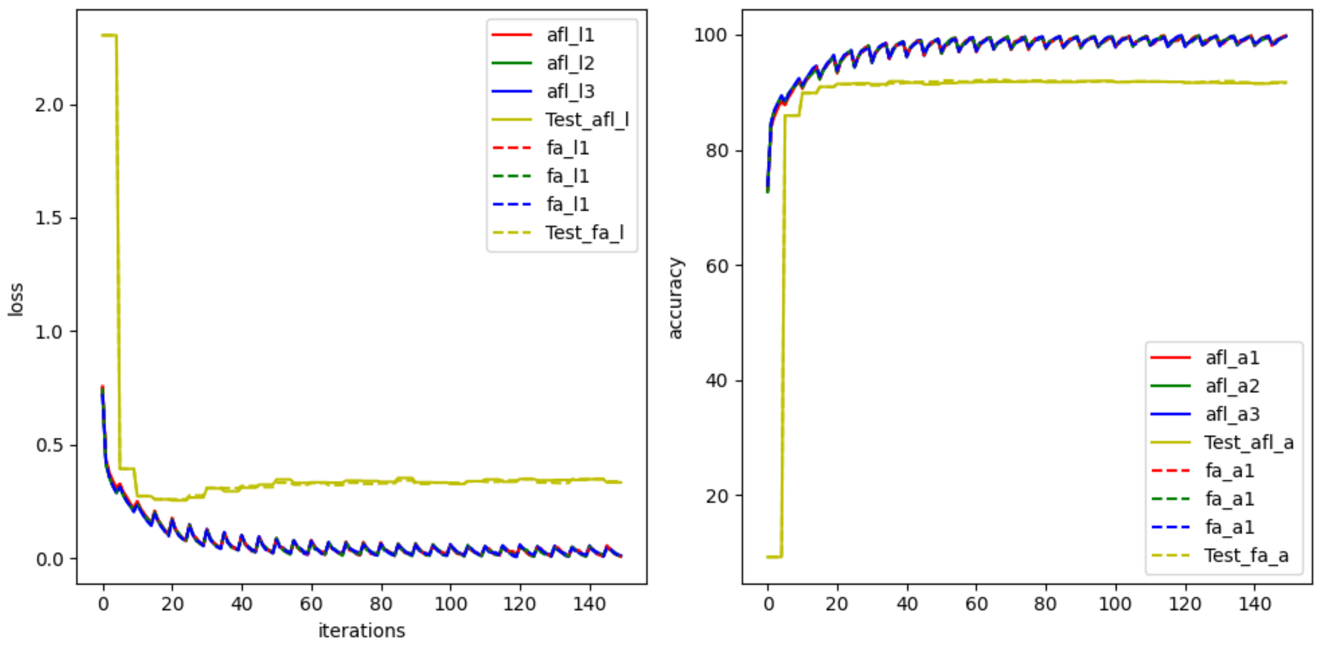


MNIST I.I.D

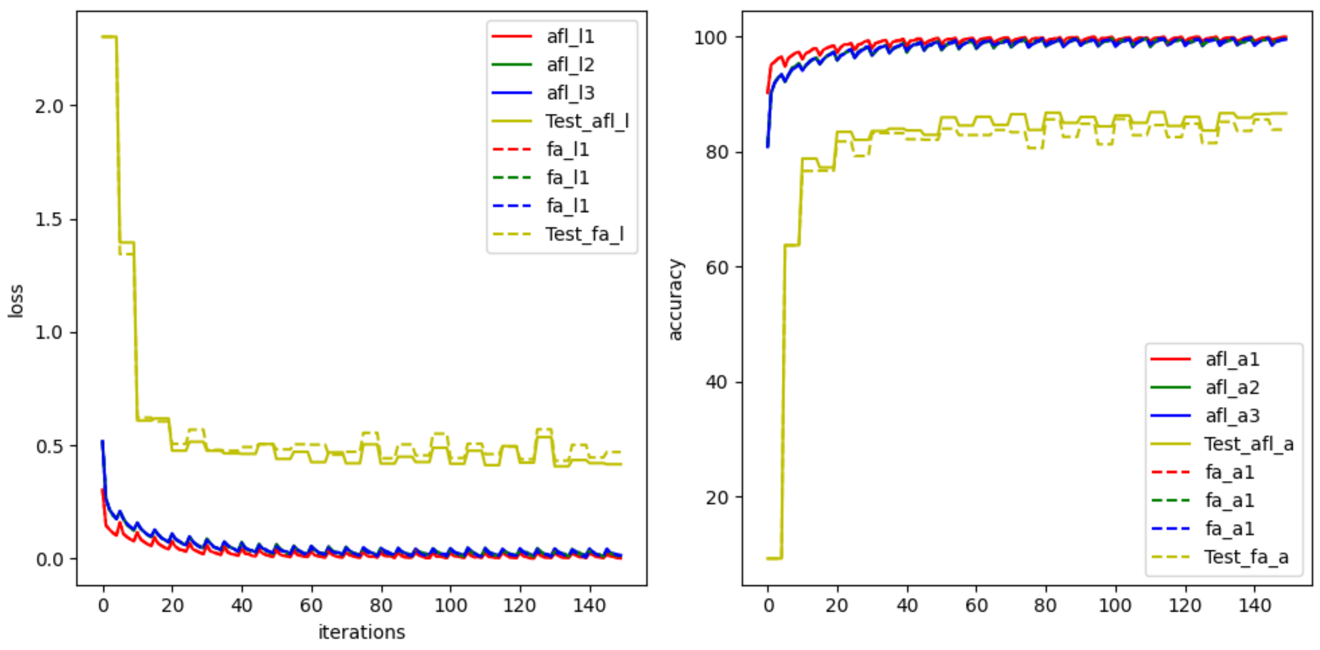


MNIST Not I.I.D

Fig. 4: Loss and Accuracy of FedAVG and AFL on MNIST dataset, with three clients and under I.I.D (Test data distribution and distribution at each client is $\mathcal{D}$) vs non-I.I.D (non-overlapping client distributions $\mathcal{D}_i$ and test data is a mixture of client distributions). alf=agnostic-federated-learning, fa=federated-averaging, l1=loss at client 1, a1 = accuracy at client 1



FMNIST I.I.D



FMNIST Not I.I.D

Fig. 5: Loss and Accuracy of FedAVG and AFL on FMNIST dataset, with three clients and under I.I.D (Test data distribution and distribution at each client is $\mathcal{D}$) vs non-I.I.D (non-overlapping client distributions $\mathcal{D}_i$ and test data is a mixture of client distributions). alf=agnostic-federated-learning, fa=federated-averaging, l1=loss at client 1, a1 = accuracy at client 1

We conducted experiments to compare the performance of FedAvg and AFL under two conditions 1) I.I.D where the data distribution at all three clients and at test time is the same. 2) non-I.I.D where the data distribution at each client is non-overlapping and data distribution at test time is a mixture of client distributions. The results of these experiments are visualized in 5. We repeat the experiments for CIFAR-10, MNIST and FMNIST datasets. AFL performed better than FedAvg on all three dataset, while the performance gap is significant on CIFAR-10. We were able to improve the performance gap on FMINST a little by tuning(compared to the graph) the domain selection. Our takeaway from these experiments was that choosing domains is an important hyper-parameter in AFL for better gains.

IV. Boosting with multiple sources

A. Agnostic Learning Objective

This research paper [3] develops a boosting-based algorithm for the agnostic learning framework with multiple data sources. They optimize the following objective

$$\mathcal{L}\left(\mathcal{D}_\Lambda, f\right) = \max_{\lambda \in \Lambda} \mathcal{L}\left(\mathcal{D}_\Lambda, f\right) \tag{19}$$

Subject to the constraints used in agnostic learning

$$\mathcal{D}_\Lambda = \sum_{k=1}^{p} \lambda_K \mathcal{D}_k \tag{20}$$

$$\lambda_i \geq 0, \forall i \in [1, p] \tag{21}$$

$$\sum_{k=1}^{p} \lambda_k = 1 \tag{22}$$

Where the learner receives labeled samples from p source domains each having distribution $\mathcal{D}_k$.

B. Ensemble predictor

To come up with an ensemble predictor first consider the standard ensemble predictor with N_l predictors for data source $\mathcal{D}_l$ which is

$$f = \sum_{i=1}^{p} \sum_{j=1}^{N_l} \alpha_{l,j} h_{l,j} \tag{23}$$

This ensemble function can produce more accurate solutions in some cases if a Q-function is introduced which gives the probability that the input belongs to a data source. In fact, the simple ensemble is just a special case of this "Q-ensemble" with equal weights given to all classes for each input. This Q-function can be used to get a weighted ensemble of the form

$$f(.) = \sum_{i=1}^{p} \sum_{j=1}^{N_l} \alpha_{l,j} h_{l,j} Q(l|.) h_{l,j}(.) \tag{24}$$

These Q functions can be learned from unlabeled data by only knowing which domain or data source the data point belongs to using techniques like conditional maximum entropy models.

C. Optimization details

They take a different approach to optimize this objective compared to Agnostic Federated Learning [7]. The constraints (30) and (31) are changed to λ being the full simplex. This leads to the following loss function

$$\mathcal{L}\left(\mathcal{D}_\Lambda, f\right) = \max_{k \in [1,p]} \mathcal{L}\left(\mathcal{D}_k, f\right) \tag{25}$$

Since this objective is not differentiable the subdifferential is used which is the convex hull of the derivative of the objective of each domain or data source. The largest absolute directional derivative is taken as the subdifferential since it suffices to choose a non-zero directional derivative. Similar to AdaBoost the standard 0-1 loss is upper bounded by the exponential function which is easier to optimize since it is convex and differentiable. This gives the following loss function

$$F = \max_{k=1}^{p} F_k \tag{26}$$

Where

$$F_k(\alpha) = \frac{1}{m_k} \sum_{i=1}^{m_k} \Phi \left(-y_{k,i} \sum_{l=1}^{p} \sum_{j=1}^{N_l} \alpha_{l,j} Q(l|x_{k,i}) h_{l,j}(x_{k,i}) \right) \tag{27}$$

D. MultiBoost Algorithm

Based on optimizing (26) using coordinate descent the MultiBoost algorithm 6 is obtained. The MutliBoost algorithm can be federated to get the FedMultiBoost algorithm 8. The FedMultiBoost algorithm first computes the global ensemble model to be used at the server. This model is then broadcasted to the clients. The clients then compute the loss for each domain or data source for client data which is then sent back to the server. The server computes the aggregate loss for each domain and then finds the domain with the largest error. Based on this the base classifier is selected for this round of boosting. The client then calculates the update for the base classifier which is then sent back to the server and averaged to get the update to the Boosting algorithm.

```
MultiBoost($S_1, \ldots, S_p$)
$\alpha_0 \leftarrow 0$
for $t \leftarrow 1$ to $T$ do
    $f_{t-1} \leftarrow \sum_{l=1}^{p} \sum_{j=1}^{N_l} \alpha_{t-1,l,j} \mathsf{Q}(l|\cdot)\, h_{l,j}$
    $\tilde{\Phi}_k \leftarrow \frac{1}{m_k} \sum_{i=1}^{m_k} \Phi(-y_{k,i} f_{t-1}(x_{k,i}))$
    $\mathcal{K}_t \leftarrow \left\{k : k \in \text{argmax}_{k \in [p]} \tilde{\Phi}_k \right\}$
    for $k \in \mathcal{K}_t$ do
        $Z_{t,k} \leftarrow \sum_{i=1}^{m_k} \Phi'\big(-y_{k,i} f_{t-1}(x_{k,i})\big)$
        for $i \leftarrow 1$ to $m_k$ do
            $\mathsf{D}_t(k, i) \leftarrow \frac{\Phi'(-y_{k,i} f_{t-1}(x_{k,i}))}{Z_{t,k}}$
    $(k, r) \leftarrow \underset{k \in \mathcal{K}_t, r \in [N_k]}{\text{argmax}} \frac{Z_{t,k}}{m_k} [1 - 2\epsilon_{t,k,r}]$
    $\eta_t \leftarrow \text{argmin}_{\eta > 0} F(\alpha_{t-1} + \eta \mathbf{e}_{k,r})$
    $\alpha_t \leftarrow \alpha_{t-1} + \eta_t \mathbf{e}_{k,r}$
$f \leftarrow \sum_{l=1}^{p} \sum_{j=1}^{N_l} \alpha_{T,l,j} \mathsf{Q}(l|\cdot) h_{l,j}$
return $f$
```

Fig. 6: MultiBoost algorithm given in [3] The algorithm uses N_l predictors for each domain $\mathcal{D}_l$ and requires a function $Q(l|.)$ which gives the probability of domain l given some input data. The function Φ here is the exponential function but can also be some other convex differentiable function that upper bounds the $0-1$ loss.

```
FedMultiBoost($S_1, S_2, \ldots, S_n$)
$\alpha_0 \leftarrow 0$
for $t \leftarrow 1$ to $T$ do
    $f_{t-1} \leftarrow \sum_{l=1}^{p} \sum_{j=1}^{N_l} \alpha_{t-1,l,j} \mathsf{Q}(l|\cdot)\, h_{l,j}$
    $C_{t,k} \leftarrow$ SelectClient$(k, r)$
    for $k \leftarrow 1$ to $p$ do
        for $c \in C_{t,k}$ do
            $\tilde{\Phi}_{k,c} \leftarrow \frac{1}{m_c} \sum_{i=1}^{m_c} \Phi(-y_{c,i} f_{t-1}(x_{c,i}))$
            $Z_{t,c} \leftarrow \sum_{i=1}^{m_c} \Phi'\big(-y_{c,i} f_{t-1}(x_{c,i})\big)$
    $\mathcal{K}_t \leftarrow \left\{k \in [p] : k \in \text{argmax}_{k \in [p]} \sum_{c \in C_{t,k}} \tilde{\Phi}_{k,c} \right\}$
    $Z_{t,k} \leftarrow \sum_{c \in C_{t,k}} Z_{t,c}, \forall k \in [p]$
    $\mathcal{H}_{t,k} \leftarrow$ SelectBaseClassifier$(k, s)$
    for $k \in \mathcal{K}_t$ do
        for $c \in C_{t,k}$ do
            for $i \leftarrow 1$ to $m_c$ do
                $\mathsf{D}_t(c, i) \leftarrow \frac{\Phi'(-y_{c,i} f_{t-1}(x_{c,i}))}{Z_{t,k}}$
            $\beta_{t,c,r} \leftarrow [1 - 2\epsilon_{t,c,r}]$
    $(k, r) \leftarrow \underset{k \in \mathcal{K}_t, r \in \mathcal{H}_{t,k}}{\text{argmax}} \frac{Z_{t,k}}{\sum_{c \in C_{t,k}} m_c} \sum_{c \in C_{t,k}} \beta_{t,c,r}$
    $\eta_t \leftarrow \eta_0 / \sqrt{t}$
    $\alpha_t \leftarrow \alpha_{t-1} + \eta_t \mathbf{e}_{k,r}$
$f \leftarrow \sum_{l=1}^{p} \sum_{j=1}^{N_l} \alpha_{T,l,j} \mathsf{Q}(l|\cdot) h_{l,j}$
return $f$
```

Fig. 7: Federated version of the MultiBoost algorithm given in [3]. Note that $\epsilon_{t,c,r}$ is the weighted error of $Q(k|.)h_{k,r}$. The blue steps here are done by the server and the red steps are done by the clients.

V. Communication-efficient agnostic federated averaging

A. Motivation

The Agnostic federated learning algorithm is not communication efficient and also requires the clients to reveal their domains to the server which may not be privacy-preserving. Furthermore, in the cross-device setting, this algorithm may be infeasible as it may need millions of steps. Thus, a communication-efficient agnostic federated averaging algorithm is suggested in [8].

B. Agnostic objective

The paper optimizes the same agnostic learning objective as considered in previous papers

$$\mathcal{L}\left(\mathcal{D}_\Lambda, f\right) = \max_{\lambda \in \Lambda} \mathcal{L}\left(\mathcal{D}_\Lambda, f\right) \tag{28}$$

Subject to the constraints used in agnostic learning

$$\mathcal{D}_\Lambda = \sum_{k=1}^{p} \lambda_K \mathcal{D}_k \tag{29}$$

$$\lambda_i \geq 0, \forall i \in [1, p] \tag{30}$$

$$\sum_{k=1}^{p} \lambda_k = 1 \tag{31}$$

C. Optimization details

Instead of what is done in [3] the authors directly optimize the minimax game to get updates for model weights and λ. This leads to an alternating optimization type algorithm where in one optimization step we first update the weights of the model and then update λ, the domain weights. The client-side step to update the client-specific model weights can be obtained by Stochastic gradient descent on the agnostic objective using client data. If S_k are samples drawn from the client distribution for training at every round the SGD update is as shown below

$$w = w - \gamma \nabla_w \left(\sum_{i=1}^{p} \frac{\lambda_i \sum_{j \in S_k \cap \hat{\mathcal{D}}_i} \mathbf{L}(w, x_j, y_j)}{N_i} \right) \tag{32}$$

$$w = w - \gamma \nabla_w \left(\sum_{i=1}^{p} \alpha_i \sum_{j \in S_k \cap \hat{\mathcal{D}}_i} \mathbf{L}(w, x_j, y_j) \right) \tag{33}$$

Here, $\alpha_i = \lambda_i / N_i$ and can be thought of as the weight given to each sample of domain $\mathcal{D}_i$. Note that to get the value of α we need the value of N_i which is not available at the server since the server does not have access to client data. Also, the client may not train on all of its data in a federated learning round meaning that having an exact value of N_i is not possible. Because of this reason the values of N_i need to be estimated and the algorithm does this by taking the average of the previous r rounds. For updates to λ exponentiated gradient step is used which is shown below

$$\lambda_t = \frac{\lambda_{t-1} \exp\left(\gamma \mathbf{L}\right)}{\sum_{i=1}^{p} \lambda_{t-1}^{i} \exp\left(\gamma \mathbf{L}_i\right)} \tag{34}$$

Algorithm AgnosticFedAvg is obtained by combining updates (33) and (34)

D. Communication cost

Although the communication cost of AgnosticFedAvg [8] is lower than the Agnostic Learning algorithm in [7] it needs more bandwidth compared to FedAvg. This is because sending α to the client and receiving $\mathbf{L}$ and $\mathbf{N}$ takes up extra communication bandwidth. This extra communication cost is $4cp$ where p is the number of domains and c is the number of clients. If the domains are chosen as the client distributions then the additional communication cost can be quadratic in the number of domains.

Algorithm	Communication Cost
FedAvg	$2c\|\mathcal{W}\|$
AgnosticFedAvg	$2c\|\mathcal{W}\| + 4cp$
ApproxFedAvg	$2c\|\mathcal{W}\|$

TABLE I: Comparison of communication cost of different FedAvg algorithms from [8]

E. Alternative AgnosticFedAvg

If we minimize the loss function used in [3] we can obtain an alternative Agnostic Federated Averaging algorithm. This means we minimize the objective

$$F = \max_{k=1}^{p} F_k \tag{35}$$

Algorithm 1 AgnosticFedAvg

```
1: procedure SERVER
2:     $w_0 \in \mathcal{W}, \lambda_0 \in \Delta_p, \mathbf{N}_0 \in \mathbb{N}^p$
3:     for round $t = 1$ to $T$ do
4:         $\alpha_t \leftarrow \frac{\lambda_{t-1}}{\sum_{j=t-r}^{t-1} \mathbf{N}_j / r}$
5:         $C_t \leftarrow$ (random set of $c$ clients)
6:         for client $k \in C_t$ do
7:             $w_t^k, \beta_t^k, \mathbf{L}_t^k, \mathbf{N}_t^k \leftarrow \text{CLIENT}(k, w_{t-1}, \alpha_t)$
8:         end for
9:         $w_t \leftarrow \sum_{k \in C_t} \beta_t^k w_t^k / \beta_t$
10:        $\mathbf{N}_t \leftarrow \sum_{k \in C_t} \mathbf{N}_t^k$
11:        $\mathbf{L} \leftarrow \sum_{k \in C_t} \mathbf{L}_t^k / \mathbf{N}_t$
12:        $\lambda_t \leftarrow \frac{\lambda_{t-1} \cdot \exp(\gamma_\lambda \mathbf{L})}{\sum_{i=1}^p \lambda_{t-1}^i \cdot \exp(\gamma_\lambda \mathbf{L}_i))}$
13:    end for
14: end procedure
```

```
1: procedure CLIENT($k, w, \alpha$)                ▷ Run on client $k$
2:     for domain $i = 1$ to $p$ do
3:         $\mathbf{L}_i^k \leftarrow |S_k \cap \widehat{\mathfrak{D}}_i| \cdot \mathsf{L}(w, S_k \cap \widehat{\mathfrak{D}}_i)$
4:         $\mathbf{N}_i^k \leftarrow |S_k \cap \widehat{\mathfrak{D}}_i|$
5:         $\beta^k \leftarrow \sum_{i=1}^p \alpha_i |S_k \cap \widehat{\mathfrak{D}}_i|$
6:     end for
7:     $\mathbf{B} \leftarrow$ (split $S_k$ into batches of size $B$)
8:     for epoch $e = 1$ to $E$ do
9:         for batch $b \in \mathbf{B}$ do
10:            $w \leftarrow w - \gamma_w \nabla \left( \frac{\sum_{i=1}^p \alpha_i \sum_{j \in b \cap \widehat{\mathfrak{D}}_i} \mathsf{L}(w, x_j, y_j)}{\beta^k} \right)$
11:        end for
12:    end for
13:    return $w, \beta^k, \mathbf{L}^k, \mathbf{N}^k$
14: end procedure
```

Fig. 8: Agnostic federated averaging is obtained by combining updates (33) and (34). The value of w, λ and $\mathbf{N}$ is initialized as w_0, λ_0 and $\mathbf{N}_0$. $\mathbf{N}$ is a vector of the number of samples in each domain. $\mathbf{L}$ is the loss for each domain.

This function is difficult to optimize and we replace the max function with the softmax function which is differentiable. This gives the following objective

$$\mathcal{L}\left(\mathcal{D}_\Lambda, f\right) = \frac{1}{\mu} \log \left(\sum_{k=1}^{p} \exp(\mu F_k) \right) \tag{36}$$

Minimizing this objective is equivalent to minimizing the loss on the top k domains with the worst performance because of the softmax function. As the value of μ increases this function approaches the maximum function and optimizing it is equivalent to minimizing the worst error out of all the domains. A stochastic gradient step on this objective will give the following update

$$w = w - \gamma \left(\frac{\sum_{k=1}^{p} \exp(\mu F_k) \nabla F_k}{\sum_{k=1}^{p} \exp(\mu F_k)} \right) \tag{37}$$

For agnostic federated averaging

$$F_k = \sum_{j \in S_k \cap \hat{\mathcal{D}}_k} L(w, x_j, y_j) \tag{38}$$

F_k is the objective for domain $\mathcal{D}_k$ where SGD is run on sample S_k. L is the loss function used for the problem.
This update can be used on the client side and removes the need for calculating λ and also sending back the loss, λ and $\mathbf{N}$ for each client. Based on this we suggest an alternate ApproxFedAvg algorithm which is shown in Algorithm 1 and Algorithm 2

Algorithm 1 ApproxFedAvg Server Update

```
1: $w_0 \in \mathcal{W}$
2: for $t \leftarrow 1, T$ do
3:     $C_t \leftarrow$ (random c clients)
4:     for client $k \in C_t$ do
5:         $w_t^k, \beta_t^k \leftarrow$ CLIENT $(k, \alpha_t)$
6:     end for
7:     $w_t \leftarrow \sum_{k \in C_t} \beta_t^k w_t^k / \beta_t$
8: end for
9: Return text, text
```

Algorithm 2 ApproxFedAvg Client update

```
1: for domain $\leftarrow 1, p$ do
2:     $\beta^k \leftarrow \sum_{i=1}^{p} \alpha_i |S_k \cap \hat{\mathcal{D}}_i|$
3: end for
4: B $\leftarrow$ Split $S_k$ into batches of size $B$
5: for $epoch \leftarrow 1, E$ do
6:     for batch b $\in B$ do
7:         $w \leftarrow w - \gamma \left( \frac{\sum_{k=1}^{p} \exp\left(\mu\left(\sum_{j \in S_k \cap \hat{\mathcal{D}}_i} \mathbf{L}(w, x_j, y_j)\right)\right)\left(\sum_{j \in S_k \cap \hat{\mathcal{D}}_i} g(w, x_j, y_j)\right)}{\sum_{k=1}^{p} \exp\left(\mu\left(\left(\sum_{j \in S_k \cap \hat{\mathcal{D}}_i} \mathbf{L}(w, x_j, y_j)\right)\right)\right)} \right)$
8:     end for
9: end for
10: Return $w$, $\beta^k$
```

F. Drawbacks of ApproxFedAvg

The new suggested algorithm's server-side update is the same as FedAvg with a weighted loss. The client-side update weighs the gradient of domain $\mathcal{D}_k$ by $\exp(\mu F_k)$ and takes a weighted average over all domains. This update then gives more precedence to gradients of domains that perform worse. Notably, we see that this algorithm is a special case of AgnosticFedAvg if we keep

$$\alpha_i = exp\left(\mu\left(\sum_{j \in S_k \cap \hat{\mathcal{D}}_i} \mathbf{L}(w, x_j, y_j)\right)\right) \tag{39}$$

Thus, ApproxFedAvg approximates the value of α_i on the client side. The main advantage of ApproxFedAvg is that it removes the need to deal with and update λ and consequently reduces the communication cost of ApproxFedAvg to the same as FedAvg $(2c|\mathcal{W}|)$

VI. Alternatives to Agnostic Learning

A. Analysis of Agnostic Learning

Agnostic learning minimizes a minimax loss that covers all convex combinations of domain distributions. However, real world test distributions may not be contained in the convex combination of domains a very simple counter example to show this is as follows.

Consider a multiclass classification setting where there are two domains and three classes. Each domain consists of a single source input x which has some probability of being one of the three classes. Consider the classes as 0, 1, and 2. If $\mathcal{D}(x, y)$ gives the probability that distribution $\mathcal{D}$ gives input x label y. Then the distributions $\mathcal{D}_1$ is given by

$$\mathcal{D}_1(x) = (1/2, 1/2, 0) \tag{40}$$

This means that domain $\mathcal{D}_1$ gives class 0 probability $1/2$, class 1 probability $1/2$ and class 2 probability 0. Distribution $\mathcal{D}_2$ is given by

$$\mathcal{D}_2(x) = (0, 0, 1) \tag{41}$$

Now any convex combination of $\mathcal{D}_1$ and $\mathcal{D}_2$ is given by

$$p\mathcal{D}_1 + (1-p)\mathcal{D}_2 = (p/2, p/2, (1-p)) \tag{42}$$

Now if the test distribution is given by $\mathcal{D}_{test}(x) = (1/2, 0, 1/2)$ then the convex combination of these domains can not represent the test distribution. This shows that the convex combination of domains may not be robust enough to match any test domain. It becomes harder to match any test domain as we increase the number of data examples x and the number of classes or the output dimension.

B. Tradeoff between the number of domains and generalization

The above example showed that agnostic learning is not robust enough to represent any distribution. But if we increased the number of domains and their diversity the agnostic learning objective becomes more robust. However, the Rademacher complexity bound of agnostic learning (16) suggests that the generalization of agnostic learning falls as we increase the number of domains. This suggests a tradeoff when increasing the number of domains. For better generalization, it may be better to cluster the clients and use the cluster as domains. But at the same time, this may reduce the robustness and coverage of the convex combination of their distribution.

C. Invariant Learning [2]

Invariant learning [2] provides an approach to learning relationships that are invariant across domains and hope to provide generalization by ignoring spurious relationships which may not be relevant for other domains. Invariant learning circumvents the generalization problem in agnostic learning as agnostic learning cannot prepare for distributions outside the convex hull of domains. The invariant risk minimization problem is formulated as follows: Consider datasets $D_e := (x_i^e, y_i^e)$ belonging to the set of training environments $e \in \mathcal{E}_{tr}$. The goal of invariant risk minimization is to learn a model that performs well across many unseen but closely related environments $\mathcal{E}_{all} \supset \mathcal{E}_{tr}$. A data presentation Φ from the input space $\mathcal{X}$ to embedding space $\mathcal{H}$ is an invariant predictor for environments $\mathcal{E}$ if there is a linear classifier w from the embedding space $\mathcal{H}$ to the output space $\mathcal{Y}$ which is optimal across all environments $e \in \mathcal{E}_{all}$.
This problem can be mathematically phrased as

$$\min_{\substack{\Phi:\mathcal{X}->\mathcal{H};\\ w:\mathcal{H}->\mathcal{Y}}} \sum_{e\in\mathcal{E}_{tr}} R^e(w \circ \Phi) \tag{43}$$

subject to the constraint

$$w \in \arg\min_{\hat{w}:\mathcal{H}->\mathcal{Y}} R^e(\hat{w} \circ \Phi) \text{ , for all } e \in \mathcal{E}_{tr} \tag{44}$$

However, this optimization problem is hard to solve so the authors suggest the following objective

$$\min_{\Phi:\mathcal{X}->\mathcal{Y}} \sum_{e\in\mathcal{E}_{tr}} R^e(\Phi) + \lambda \cdot ||\nabla_{w|w=1.0} R^e(w \cdot \Phi)||^2 \tag{45}$$

Where the regularization term is used to measure the optimality of a simple $w = 1$ classifier at each environment e by penalizing the gradient norm squared. This is because making the gradient closer to zero ensures that we are close to an optimum.

D. Federating Invariant Learning

Research work has been done on testing the federated version of invariant learning. [4] minimizes the objective given in [1] to develop an invariant federated learning algorithm. Work has not yet been done on contrasting the performance of agnostic learning, invariant learning, and other algorithms for learning from multiple sources, and finding which algorithm performs the best in the context of federated learning. This forms an interesting direction for future work.

VII. Conclusion

In this survey, we have gone over federated learning algorithms for learning from multiple data sources. This is an important problem because real-world deployments of machine learning models are often not able to cope with distribution shifts. Federated learning is an interesting way to learn machine learning models in a data-private fashion and is suited to deployment in real-world distributed systems. Making federated learning robust to shifts in data distribution is important to ensure that these model deployments do not suffer from a lack of generalization or fail on a distribution shift. We started by going over FedAvg [6], then covering three research papers on agnostic federated learning [7], [3] and [8]. Following this we also covered invariant risk minimization [2], an alternative approach to the learning from multiple sources problem.

References

[1] Kartik Ahuja et al. *Invariant Risk Minimization Games*. 2020. arXiv: 2002.04692 `[cs.LG]`.

[2] Martin Arjovsky et al. *Invariant Risk Minimization*. 2020. arXiv: 1907.02893 `[stat.ML]`.

[3] Corinna Cortes et al. "Boosting with Multiple Sources". In: *Advances in Neural Information Processing Systems*. Ed. by A. Beygelzimer et al. 2021.

[4] Sharut Gupta et al. *FL Games: A federated learning framework for distribution shifts*. 2022. arXiv: 2205.11101 `[cs.LG]`.

[5] Sai Praneeth Karimireddy et al. "SCAFFOLD: Stochastic Controlled Averaging for Federated Learning". In: *Proceedings of the 37th International Conference on Machine Learning*. Ed. by Hal Daumé III and Aarti Singh. Vol. 119. Proceedings of Machine Learning Research. PMLR, 13–18 Jul 2020, pp. 5132–5143.

[6] Brendan McMahan et al. "Communication-efficient learning of deep networks from decentralized data". In: *Artificial intelligence and statistics*. PMLR. 2017, pp. 1273–1282.

[7] Mehryar Mohri, Gary Sivek, and Ananda Theertha Suresh. "Agnostic federated learning". In: *International Conference on Machine Learning*. PMLR. 2019, pp. 4615–4625.

[8] Jae Ro et al. *Communication-Efficient Agnostic Federated Averaging*. 2021. arXiv: 2104.02748 `[cs.LG]`.

[9] Jianyu Wang et al. "A field guide to federated optimization". In: *arXiv preprint arXiv:2107.06917* (2021).