Efficient and Light-Weight Federated Learning via Asynchronous Distributed Dropout

Anonymous Author(s)

Abstract

We focus on dropout techniques for asynchronous distributed computations in federated learning (FL) scenarios. We propose `AsyncDrop`, a novel asynchronous FL framework with smart (i.e., informed/structured) dropout that achieves better performance compared to state of the art asynchronous methodologies, while resulting in less communication and training time costs. The key idea revolves around sub-models out of the global model, that take into account the device heterogeneity. We conjecture that such an approach can be theoretically justified. We implement our approach and compare it against other asynchronous baseline methods, by adapting current synchronous FL algorithms to asynchronous scenarios. Empirically, `AsyncDrop` significantly reduces the communication cost and training time, while improving the final test accuracy in non-i.i.d. scenarios.

1 Introduction

Background on Federated Learning (FL). Federated Learning (FL) [21, 17, 15] is a distributed learning protocol that has witnessed fast development the past demi-decade, as it deviates from the traditional distributed learning paradigms and allows the integration of small computing (edge) devices —such as smartphones [30], drones [25], and IoT devices [22]— in the learning procedure. Such real-life, edge devices though are extremely *heterogeneous* [33]: they have drastically different specifications in terms of compute power, device memory and communication bandwidths. Directly applying common *synchronized* FL algorithms –such as FedAvg and FedProx [17, 21] that require full model broadcasting and global synchronization– results often in a "stragglers" effect [23, 13, 31].

The dominant use of synchronous computations. One way to handle such issues is by utilizing *asynchrony* instead of *synchrony* in the learning process. To explain the main differences, let us first set up the background. In a synchronous distributed algorithm, a global model is usually stored at a central server and is broadcast periodically to all the participating devices. Then, each device performs some local training steps on its own model copy, before the device sends the updated model to the central server. Finally, the central server updates the global model by aggregating the received updated model copies. This protocol is followed in most FL protocols, including the most well-known baselines like FedAvg [21], FedProx [17], FedNova [34] and SCAFFOLD [15]. The main criticism against synchronous learning could be that it often results in heavy communication-/computation-overheads when mobile devices are used, long idle/waiting times for workers and stragglers effects.

Main dispute in asynchronous learning. While asynchrony in computations seems a favorable alternative, the development of a asynchronous learning method, however, is often convoluted. In the past decade, HogWild! [24, 20] has emerged as a general asynchronous distributed methodology, and has been applied initially in basic ML problems like sparse linear/logistic regression [42, 41, 12]. Ideally, based on sparsity arguments, each edge device can independently update parts of the global model –that overlap only slightly with the updates of other workers– in a lock-free fashion [24, 20]. This way, faster, more powerful edge workers do not suffer from idle waiting due to slower stragglers. Yet, the use of asynchrony has been a topic of dispute in distributed neural network training [5, 2]. Asynchronous training often inherently suffers from lower accuracy as compared to synchronized SGD, which results in the domination of synchronized SGD in neural network training [2].

Resurgence in asynchrony. Recently, asynchronous methods have regained popularity, mainly due to the interest in applying asynchronous motions within FL on edge devices. Yet, traditional

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

off-the-self asynchronous algorithms still have issues, which might be exacerbated in the FL setting. As slower devices take longer local training time before updating the global model, this might result in inconsistent update schedule of the global model, compared to that of faster devices. This might have negative ramifications: i) For FL on i.i.d. data, this will cause the gradient staleness problem and result in convergence rate decrease; and, ii) on non-i.i.d. data, this will result in significant drop in global model final accuracy.

To overcome such problems, fresh proposals on asynchronous FL develop weighted global aggregation techniques that take into consideration the heterogeneity of the devices [38, 3, 28]; yet, these methods often place a heavy computation/communication burden, as they rely on broadcasting the whole model's updates to all the clients and/or the server. Other works monitor client speed to help training assignment [18, 1]. Finally, recent efforts propose semi-asynchronous methods, where participating devices are selected and buffered in order to complete a semi-synchronous global update periodically [13, 37]. A thorough discussion on asynchronous methods in FL can be found in [39].

What is different in this work? As most algorithms stem from directly adapting asynchrony in synchronous FL, *one still needs to broadcast full model to all devices, following a data parallel distributed protocol [9, 26], regardless of device heterogeneity*. This inspire us to ask a key question:

> *"Can we select sub-models out of the global model and send these to each device, taking into account the device heterogeneity? Potentially, this could further reduce the negative effect of gradient staleness and overfitting problem in asynchronous FL."*

We answer this question affirmatively, by proposing a novel distributed dropout method for FL. We dub our method `AsyncDrop`. Our approach assigns different sub-models to each device[1]; empirically, such a strategy decreases the required time to converge to an accuracy level, while preserving favorable final accuracy. This work reinstitutes the dispute between synchrony and asynchrony and focuses on asynchronous computations should in heterogeneous distributed scenarios, as in FL. Our idea is based on the ideas of HogWild! [24, 20] –in terms of sparse sub-models– and Independent Subnetwork Training (IST) [40, 8, 19, 36] –where sub-models are deliberately created for distribution, in order to decrease both computational and communication requirements.

Yet, we deviate from these works in the following sense: i) To the best of our knowledge, we are not aware of any fully asynchronous FL implementations; the combination of HogWild! and IST ideas was not stated and tested before this work. ii) While HogWild!-line of work provides optimization guarantees, we are the first to consider the non-trivial, non-convex neural network setting and provide theoretical guarantees for convergence; such a result combines tools from asynchronous optimization, Neural Tangent Kernel assumptions and focuses on convolutional neural networks, that deviate from fully-connected layer simplified scenarios. Finally, iii) we provide system-level algorithmic solutions for our approach, mirroring best-practices found during our experiments. Overall, the contributions of this work can be summarized as follows:

- We consider and propose *asynchronous distributed dropout* for efficient large-scale FL. Our focus is on non-trivial, non-convex ML models –as in neural network training– and our framework provides specific engineering solutions how to handle these cases in practice.
- We theoretically characterize and support our proposal with rigorous and non-trivial convergence rate guarantees. Currently, our conjecture assumes bounded delays; our future goal is to exploit recent developments on asynchronous optimization theory that drop such assumptions. Yet, our theory considers the harder case of neural network training, which is often omitted in existing –but rather abstract– theory results.
- We provide specific implementation instances and share "best practices" for faster distributed FL neural network training in practice. Our preliminary results include baseline asynchronous implementations of many synchronous methods (such as FedAvg, FedProx, and more), that are not existent currently, to the best of our knowledge.

2 Problem Setup and Challenges

Federated Learning formulation. We use the following notation for data representation: given a data source distribution $\mathcal{D}$, $\{\mathbf{x}_i, y_i\} \sim \mathcal{D}$ represents a data point where $\mathbf{x}_i$ is a data sample, and

[1] We consider both random assignment, as well as structured assignments, based on the computation power of the devices.

y_i is its corresponding label (for classification purposes). We use n to denote the total number of samples, unless otherwise stated. Let S be the total number of clients in a distributed FL scenario. Each client i has its own local data $\mathcal{D}_i$ such that the whole dataset satisfies $\mathcal{D} = \cup_i \mathcal{D}_i$, and usually $\mathcal{D}_i \cap \mathcal{D}_j = \emptyset, \forall i \neq j$. The goal of federated learning is to find a global model $\mathbf{W}$ that achieves good accuracy on all data $\mathcal{D}$, by minimizing the following optimization problem:

$$\mathbf{W}^\star = \underset{\mathbf{W}\in\mathcal{H}}{\operatorname{argmin}} \left\{ \mathcal{L}(\mathbf{W}) := \frac{1}{S}\sum_{i=1}^{S} \ell\left(\mathbf{W}, \mathcal{D}_i\right) \right\}, \text{ where } \ell\left(\mathbf{W}, \mathcal{D}_i\right) = \frac{1}{|\mathcal{D}_i|} \sum_{\{\mathbf{x}_j, y_j\}\in\mathcal{D}_i} \ell\left(\mathbf{W}, \{\mathbf{x}_j, y_j\}\right)$$

With a slight abuse of notation, $\ell\left(\mathbf{W}, \mathcal{D}_i\right)$ denotes the *local* loss function for user i, associated with a local model $\mathbf{W}_i$ (not indicated above), that gets aggregated with the models of other users. Local data distribution $\mathcal{D}_i$ can be heterogeneous and follow a non-i.i.d. distribution. In the following sections, we consider both i.i.d. case and non-i.i.d. case.

Details of asynchronous training. An abstract description of how asynchronous FL operates is provided in Algorithm 1. In particular, given a number of server iterations T, each client i gets the updated global model $\mathbf{W}_t$ from the server, and further locally trains it using $\mathcal{D}_i$ for a number of local iterations l.[2] Asynchronous federated learning assumes each client has different computation power and communication bandwidth; this can be abstracted by different wall-clock times required to finish a number of local training iterations. Thus, when client i has completed its round, the updated model is shared with the server to be aggregated, before the next round of communication and computation starts for client i. This is different from classical synchronous federated learning, where the global model is updated only when all participating clients finish certain local training iterations.

Algorithm 1 Meta Asynchronous FL

Parameters: T iters, S clients, l local iters., $\mathbf{W}$ as current global model, $\mathbf{W}_i$ as local model for i-th client, $\alpha \in (0, 1)$, η step size.

∞

$\mathbf{W} \leftarrow$ randomly initialized global model.
`//On each client` i `asynchronously:`
for $t = 0, \ldots, T-1$ **do**
  $\mathbf{W}_{i,t} \leftarrow \mathbf{W}$
  `//Train` $\mathbf{W}_i$ `for` l `iters. via SGD`
  for $j = 1, \ldots, l$ **do**
    $\mathbf{W}_{i,t} \leftarrow \mathbf{W}_{i,t} - \eta \frac{\partial \mathcal{L}}{\mathbf{W}_{i,t}}$
  end for
  `//Write local to global model`
  $\mathbf{W} \leftarrow (1-\alpha)\cdot\mathbf{W} + \alpha \cdot \mathbf{W}_{i,t}$
end for

3 Asynchronous Distributed Dropout in the i.i.d. case

Intricacies of asynchronous FL. Asynchronous motions often lead to inconsistent update schedules of the global model and are characterized by gradient staleness and overfitting. While there are some recent solutions [38, 3, 28, 18, 1, 13, 37], real-life FL applications include edge devices with limited communication and computation capabilities that further exacerbate such issues(e.g., how often and fast they can connect with the central server, and how powerful as devices they are). For instance, edge devices such as IoT devices or mobile phones [22], might only be able to communicate with the server within short time windows, due to network conditions or user permission policy (e.g., a device might need to upload and download the updated model in a short time window). Such edge devices have limited computational capabilities, as compared to GPU-based systems.

Finally, existing semi-asynchronous methods [38, 3, 28, 18, 1, 13, 37] might require fast clients wait until all other clients' updates are completed, in order to receive the updated model for the next round of local training. This might further result in nondeterministic long waiting time.

(Distributed) Dropout. Dropout [32, 29, 10, 4] is a widely-accepted regularization technique in deep learning. The procedure of Dropout is as follows: per training round, a random mask over the parameters is generated; this mask is used to nullify part of the neurons in the neural network for this particular iteration. Variants of dropout include the drop-connect [32], multisample dropout [14], Gaussian dropout [35], and the variational dropout [16].

The idea of dropout has also been used in efficient distributed and/or federated learning scenarios. [11] introduces FjORD and the Ordered Dropout, a *synchronous* distributed dropout technique that leads to ordered, nested representation of knowledge in models, and enables the extraction of lower footprint submodels without the need of retraining. Such submodels are more suitable in client system

[2] Details on the use of the optimizer, how it is tuned with respect to step size, mini-batch size, etc. are intentionally hidden at this point of the discussion.

heterogeneity, as they adapt submodel's width to the client's capabilities. Similar approaches include Nested Dropout [27] and HeteroFL [6].[3]

Our proposal and main hypothesis. In this work, we focus on the *asynchronous version of distributed dropout.* We first study theoretically whether asynchrony provably works in non-trivial non-convex scenarios –*as in training convolutional neural networks*– with random masks that generate submodels for each worker/participating device. The algorithm we focus on is described in Algorithm 2 and is based on recent distributed protocols [40, 8, 19, 36]. The main difference of Algorithm 2, compared to Algorithm 1, is that the former splits the model vertically per iteration, where each submodel contains all layers of the neural network, but only with a (non-overlapping) subset of neurons being active in each layer. Multiple local SGD steps can be performed without the workers having to communicate.

Algorithm 2 `AsyncDrop` algorithm

Parameters: T iters, S clients, l local iters., $\mathbf{W}$ as current global model, $\mathbf{W}_i$ as local model for i-th client, $\alpha \in (0,1)$, η step size.

∞

$\mathbf{W} \leftarrow$ randomly initialized global model.
`//On each client` i `asynchronously:`
for $t = 0, \ldots, T-1$ **do**
  Generate mask $\mathbf{M}_{i,t}$
  $\mathbf{W}_{i,t} \leftarrow \mathbf{W}_t \odot \mathbf{M}_{i,t}$
  `//Train` $\mathbf{W}_{i,t}$ `for` l `iters. via SGD`
  for $j = 1, \ldots, l$ **do**
    $\mathbf{W}_{i,t} \leftarrow \mathbf{W}_{i,t} - \eta \frac{\partial \mathcal{L}}{\mathbf{W}_{i,t}}$
  end for
  `//Write local to global model`
  $\mathbf{W}_{t+1} \leftarrow \mathbf{W}_t \odot (\mathbf{M}_{i,t})^c + ((1-\alpha)\cdot\mathbf{W}_t + \alpha\cdot\mathbf{W}_{i,t}) \odot \mathbf{M}_{i,t}$
end for

Theoretical Results. We perform theoretical analysis on a one-hidden-layer convolutional neural network (CNN), and show the convergence with random filter dropout. Consider a training dataset $(\mathbf{X}, \mathbf{y}) = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$, where each $\mathbf{x}_i \in \mathbb{R}^{\hat{d} \times p}$ is an image and y_i being its label. Here, $\hat{d}$ is the number of input channels and p the number of pixels. Let q denote the size of the filter, and let m be the number of filters in the first layer. We let $\hat{\phi}(\cdot)$ denote the patching operator with $\hat{\phi}(x) \in \mathbb{R}^{q\hat{d} \times p}$ [7]. Consider the first layer weight $\mathbf{W} \in \mathbb{R}^{m \times q\hat{d}}$, and second layer (aggregation) weight $\mathbf{a} \in \mathbb{R}^{m \times p}$. We assume that only the first layer weights $\mathbf{W}$ is trainable. The CNN model, trained on the means squared error, has the form:

$$f(\mathbf{x}, \boldsymbol{\zeta}) = \left\langle \mathbf{a}, \sigma\left(\mathbf{W}\hat{\phi}(\mathbf{x})\right)\right\rangle; \quad \mathcal{L}(\mathbf{W}) = \|f(\mathbf{X}, \mathbf{W}) - \mathbf{y}\|_2^2,$$

where $\boldsymbol{\zeta}$ abstractly represents all training parameters, $f(\mathbf{x}, \cdot)$ denotes the output of the one-layer CNN for input $\mathbf{x}$, and $\mathcal{L}(\cdot)$ is the loss function.

Assumption 3.1 *(Training Data) Assume that for all data points, we have* $\|\mathbf{x}_i\|_F = q^{-\frac{1}{2}}$ *and* $|y_i| \leq C$ *for some constant* C*. Moreover, for all* i, i' *we have* $\mathbf{x}_i \neq \mathbf{x}_{i'}$*.*

This assumption can be easily satisfied with data normalization. For simplicity, let $d := q\hat{d}$.

Assumption 3.2 *(Initialization)* $\mathbf{w}_{0,r} \sim \mathcal{N}\left(0, \kappa^2 \mathbf{I}\right)$ *and* $a_{rj} \sim \left\{\pm \frac{1}{p\sqrt{m}}\right\}$ *for* $r \in [m]$ *and* $j \in [p]$*.*

In an asynchronous scenario, we assume the neural network weights are updated with stale gradients: here, d_t denotes the delay at training step t. Further, for all t iterations, we assume d_t is bounded by a constant E. Then, gradient descent motions with delays take the form:

$$\mathbf{W}_t = \mathbf{W}_t - \eta \nabla_{\mathbf{W}} \mathcal{L}\left(\mathbf{W}_{t-d_t}\right), \quad \text{where } d_t \leq E.$$

Based on the above we make the following conjecture:

Conjecture 3.1 *Let* $f(\cdot,\cdot)$ *be a one-hidden-layer CNN with the second layer weight fixed. Assume the number of hidden neurons satisfies* $m = \Omega\left(\max\{\frac{n^4 K^2}{\lambda_0^4 \delta^2}\max\{n, d\}, \frac{n}{\lambda_0}\}\right)$ *and the step size satisfies* $\eta = O\left(\frac{\lambda_0}{n^2}\right)$*. Let Assumptions 1 and 2 be satisfied. Then, we conjecture that:*

$$\mathbb{E}_{[\mathbf{M}_{t-1}]}\left[\frac{\sum_{t=1}^T \|f(\mathbf{X}, \mathbf{W}_t) - \mathbf{y}\|_2^2}{T}\right] \leq \left(1 - O(\text{poly}(\tfrac{\theta\eta\lambda_0}{4}, E)\right)^t \cdot \|f(\mathbf{X}, \mathbf{W}_0) - \mathbf{y}\|_2^2 + O(\text{poly}(\xi, n, d, \lambda_0, \tfrac{1}{m}))$$

[3] Beyond the restriction to synchronous training, such approaches often lack a specific instantiation of the proposed methodology to neural network training. This present work proposes specific algorithmic solutions on how to consolidate these ideas into a single practical framework for neural network training in FL scenarios.

4 Smart Partition in Asynchronous Dropout for FL

Potential issues in asynchronous FL. Despite theoretical support, asynchronous algorithms might show slower convergence rates and/or lower final model accuracies, mainly due to inconsistent global model updates between slower devices and faster devices. With non-i.i.d. data as in the case of FL, more frequent updates by faster devices could lead to global model overfitting over local data on the same devices; this often results in significantly sub-optimal final model accuracy. *These facts suggest a more careful handling of model splitting and model distribution among heterogeneous workers.*

Algorithm 3 `Smart Dropout` for Asynchronous FL

Parameters: T iters, S clients, l local iters., $\mathbf{W}$ as current global model, $\mathbf{W}_i$ as local model for i-th client, η_g as global LR, $q(\cdot)$ weight score function, $\psi(i)$ computes the computation capacity of i-th worker, $\varphi(\mathbf{W}, \psi(i), q(\cdot))$ is the `Smart Dropout` function that creates the mask, based on worker capacity ψ and score function q, $\alpha \in (0, 1)$.

∞

$\mathbf{W} \leftarrow$ randomly initialized global model.
`//On each client` i `asynchronously:`
for $t = 0, \ldots, T-1$ **do**
  `//For` i`-th fastest worker,` $\varphi(\cdot)$ `drops weights with` i`-th largest` $q(\cdot)$ `score`
  Generate mask $\mathbf{M}_{i,t} = \varphi(\mathbf{W}_t, \psi(i), q(\cdot))$
  $\mathbf{W}_{i,t} \leftarrow \mathbf{W}_t \odot \mathbf{M}_{i,t}$
  `//Train` $\mathbf{W}_{i,t}$ `for` l `iters. via SGD`
  for $j = 1, \ldots, l$ **do**
    $\mathbf{W}_{i,t} \leftarrow \mathbf{W}_{i,t} - \eta_g \frac{\partial \mathcal{L}}{\mathbf{W}_{i,t}}$
  end for
  if i-th client is fastest **then**
    Update η_g
  end if
  `//Write local to global model`
  $\mathbf{W}_{t+1} \leftarrow \mathbf{W}_t \odot (\mathbf{M}_{i,t})^c + ((1-\alpha) \cdot \mathbf{W}_t + \alpha \cdot \mathbf{W}_{i,t}) \odot \mathbf{M}_{i,t}$
  `//Update score` q
  $q(\mathbf{W}^j_{t+1}) = \left\| \mathbf{W}^j_{t+1} - \mathbf{W}^j_0 \right\|_1, \forall j \in \mathcal{J}$
end for

`Smart Dropout`**: improved asynchronous solution**. We propose to resolve inconsistency in global model updates by "balancing" the contribution from different devices with a `Smart Dropout` method. Briefly, `Smart Dropout` assigns different weights to different devices, based on the updated rate of the weights, and the computation power of the devices.

An algorithmic description is provided in Algorithm 3. We assign model weights that are less updated to the faster devices; this way, we guarantees that such weights will be updated more frequently than random weight assignment. Similarly, we assign model weights that are updated more often – and, thus, converging faster– to the slower devices. The premise behind such a protocol is that *all weights, eventually, will be updated/will converge with a similar rate.*

In the i.i.d. case, this would reduce the gradient staleness: slower devices take fewer steps to converge on the assigned weights; this is compensated by assigning "fast-converging weights", that were assigned to faster devices in the previous iterations. In the non-i.i.d. case, such a strategy would reduce any overfitting over the local data in faster devices: The weights previously updated the most by the faster devices will not be assigned to the fast devices in the near-future iterations.

`Smart Dropout`**: its ingredients**. The above are encapsulated with a weight score function $q(\cdot)$, a device-capacity score function $\psi(\cdot)$ and the mask generator function $\varphi(\cdot)$ in Algorithm 3. The function $q(\cdot)$ quantifies the update speed of each weight. One simple score can be the norm of weight change: Given predefined grouping of weights in set $\mathcal{J}$, define $\mathbf{W}^j_t$ for $j \in \mathcal{J}$ as the j-th weight/group of weights of global network at the t-th update. Then, the score function is selected as: $q(\mathbf{W}^j_t) = \left\| \mathbf{W}^j_t - \mathbf{W}^j_0 \right\|_1$, i.e., the score function measures *how far from the initial values the j-th group of weights has moved.* In our experiments, we have tested grouping of weights $\mathcal{J}$ by filters and by layers. Further, we have some measure of the computation capacity of each device $\psi(i)$, and order the devices in descending order such that the first device is the fastest and last the slowest.

The mask generator function $\varphi(\mathbf{W}, \psi(i), q(\cdot))$ generates masks that nullify weights in $\mathbf{W}$, based on the score function q over the weights, and the device capacity list in $\psi(\cdot)$. The `Smart Dropout` strategy is that for the i-th fastest worker, we drop weights with i-th largest q score.

5 Experiments

We focus on *fully asynchronous FL using CNNs.* Yet, we highlight that, with proper tuning, our ideas are model agnostic by design. To simulate the heterogeneous edge devices, we implement devices

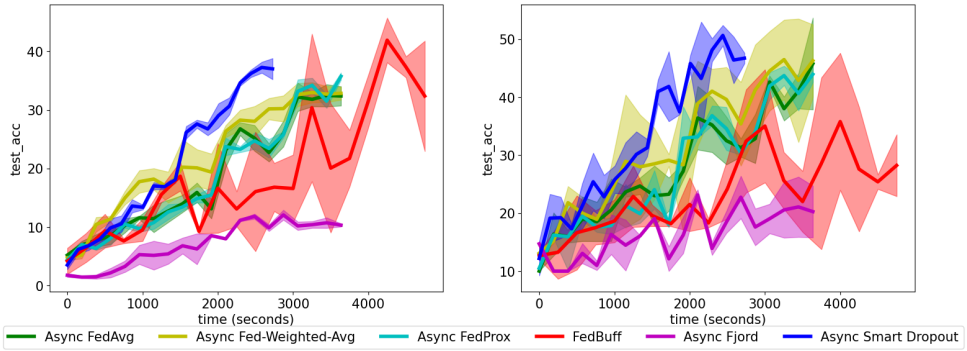

Figure 1: ***Left:*** CIFAR10 non-i.i.d.; ***Right:*** CIFAR100, non-i.i.d..

as different processes, while we manually insert different time delays to simulate different compute nodes and communication bandwidths. *In these preliminary results, we focus only on non-i.i.d. scenarios.* We consider the difficult case where data distribution will be correlated to computation power, which results in a more heterogeneous model update during training. In all experiments, we use 104 devices/users in total and activate 8 of them at any moment. In all `Smart Dropout` experiments, we use 25% dropout rate.

Baselines. We compare asynchronous implementations of five algorithms: $i)$ asynchronous FedAvg; $ii)$ asynchronous FedAvg with weighted aggregation based on gradient staleness; $iii)$ asynchronous FjORD; $iv)$ asynchronous FedProx; and $iv)$ FedBuff. These baselines are straightforward implementations of synchronous versions (except from FedBuff).

Preliminary results. We only highlight key points in our preliminary results; more findings and further discussion will be included during the workshop. Focusing on the non-i.i.d. case for CIFAR10 (Table 1), we observe that our proposal gets the maximum accuracy compared to the rest of the algorithms, while being stable: notice the high variability in values coming from different runs of our code (indicated with **red colored text**). At the same time, our algorithm is faster and requires less communication over the course of the execution.

Similar observations can be made for the case of non-i.i.d. CIFAR100 (Table 2). Here, observe that FedBuff achieves the best maximum accuracy. However, again the achieved models are not consistent over various runs of the algorithm; our algorithm shows a stable performance, within reach of the high variance by FedBuff. At the same time, our algorithms shows great performance in terms of acceleration in execution and required communication bandwidth. This behavior is further highlighted in Figure 1.

Table 1: Test accuracy on non-i.i.d. CIFAR10 datasets.

	Max. Accuracy	Time to 35% accuracy	Time Saved Ratio	Comm. Saved Ratio
Smart Dropout	**50.67** $\pm$ 1.75	**1579.1s**	-	
Async FedAvg	45.79 $\pm$ **7.9**	2105.5s	-33.34%	-15.56%
Async Fed-Weighted-Avg	46.51 $\pm$ **6.8**	2105.5s	-33.34%	-15.56%
Async FedProx	43.97 $\pm$ 1.35	2296.9s	-45.46%	-26.06%
Async Fjord	23.14 $\pm$ 0.90	N/A	N/A	N/A
FedBuff	35.81 $\pm$ **11.83**	2296.9s	-45.46%	-443.03%

Table 2: Test accuracy on non-i.i.d. CIFAR100 datasets.

	Max. Accuracy	Time to 32% accuracy	Time Saved Ratio	Comm. Saved Ratio
Smart Dropout	37.26 $\pm$ 0.93	**2296.8s**	-	
Async FedAvg	32.47 $\pm$ 1.89	3062.5s	-33.38%	-15.56%
Async Fed-Weighted-Avg	32.98 $\pm$ 1.71	3062.5s	-33.38%	-15.56%
Async FedProx	35.75 $\pm$ 0.61	3062.5s	-33.38%	-15.56%
Async Fjord	12.07 $\pm$ 0.83	N/A	N/A	N/A
FedBuff	**41.91** $\pm$ **3.80**	3253.9s	-41.72%	-488.28%

References

[1] Zheng Chai, Yujing Chen, Liang Zhao, Yue Cheng, and Huzefa Rangwala. Fedat: A communication-efficient federated learning method with asynchronous tiers under non-iid data. *ArXivorg*, 2020.

[2] Jianmin Chen, Xinghao Pan, Rajat Monga, Samy Bengio, and Rafal Jozefowicz. Revisiting distributed synchronous sgd. *arXiv preprint arXiv:1604.00981*, 2016.

[3] Yang Chen, Xiaoyan Sun, and Yaochu Jin. Communication-efficient federated deep learning with layerwise asynchronous model update and temporally weighted aggregation. *IEEE transactions on neural networks and learning systems*, 31(10):4229–4238, 2019.

[4] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. *Advances in neural information processing systems*, 28, 2015.

[5] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, et al. Large scale distributed deep networks. *Advances in neural information processing systems*, 25, 2012.

[6] Enmao Diao, Jie Ding, and Vahid Tarokh. HeteroFL: Computation and communication efficient federated learning for heterogeneous clients. *arXiv preprint arXiv:2010.01264*, 2020.

[7] Simon Du, Jason Lee, Haochuan Li, Liwei Wang, and Xiyu Zhai. Gradient descent finds global minima of deep neural networks. In *International conference on machine learning*, pages 1675–1685. PMLR, 2019.

[8] Chen Dun, Cameron R Wolfe, Christopher M Jermaine, and Anastasios Kyrillidis. ResIST: Layer-wise decomposition of ResNets for distributed training. In *Uncertainty in Artificial Intelligence*, pages 610–620. PMLR, 2022.

[9] P. Farber and K. Asanovic. Parallel neural network training on multi-spert. In *Proceedings of 3rd International Conference on Algorithms and Architectures for Parallel Processing*, pages 659–666, 1997.

[10] Yarin Gal and Zoubin Ghahramani. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059. PMLR, 2016.

[11] Samuel Horvath, Stefanos Laskaridis, Mario Almeida, Ilias Leontiadis, Stylianos Venieris, and Nicholas Lane. FjORD: Fair and accurate federated learning under heterogeneous targets with ordered dropout. *Advances in Neural Information Processing Systems*, 34:12876–12889, 2021.

[12] Cho-Jui Hsieh, Hsiang-Fu Yu, and Inderjit Dhillon. PaSSCODE: Parallel asynchronous stochastic dual co-ordinate descent. In *International Conference on Machine Learning*, pages 2370–2379. PMLR, 2015.

[13] Dzmitry Huba, John Nguyen, Kshitiz Malik, Ruiyu Zhu, Mike Rabbat, Ashkan Yousefpour, Carole-Jean Wu, Hongyuan Zhan, Pavel Ustinov, Harish Srinivas, et al. Papaya: Practical, private, and scalable federated learning. *Proceedings of Machine Learning and Systems*, 4:814–832, 2022.

[14] Hiroshi Inoue. Multi-sample dropout for accelerated training and better generalization. *arXiv preprint arXiv:1905.09788*, 2019.

[15] Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank J. Reddi, Sebastian U. Stich, and Ananda Theertha Suresh. SCAFFOLD: Stochastic controlled averaging for federated learning, 2019.

[16] Durk P Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. *Advances in neural information processing systems*, 28, 2015.

[17] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks, 2018.

[18] Xingyu Li, Zhe Qu, Bo Tang, and Zhuo Lu. Stragglers are not disaster: A hybrid federated learning algorithm with delayed gradients. *arXiv preprint arXiv:2102.06329*, 2021.

[19] Fangshuo Liao and Anastasios Kyrillidis. On the convergence of shallow neural network training with randomly masked neurons. *arXiv preprint arXiv:2112.02668*, 2021.

[20] Ji Liu, Steve Wright, Christopher Ré, Victor Bittorf, and Srikrishna Sridhar. An asynchronous parallel stochastic coordinate descent algorithm. In *International Conference on Machine Learning*, pages 469–477. PMLR, 2014.

[21] 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.

[22] Dinh C Nguyen, Ming Ding, Pubudu N Pathirana, Aruna Seneviratne, Jun Li, and H Vincent Poor. Federated learning for internet of things: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, 23(3):1622–1658, 2021.

[23] John Nguyen, Kshitiz Malik, Hongyuan Zhan, Ashkan Yousefpour, Mike Rabbat, Mani Malek, and Dzmitry Huba. Federated learning with buffered asynchronous aggregation. In *International Conference on Artificial Intelligence and Statistics*, pages 3581–3607. PMLR, 2022.

[24] Feng Niu, Benjamin Recht, Christopher Re, and Stephen J. Wright. Hogwild!: A lock-free approach to parallelizing stochastic gradient descent, 2011.

[25] Yuben Qu, Haipeng Dai, Yan Zhuang, Jiafa Chen, Chao Dong, Fan Wu, and Song Guo. Decentralized federated learning for UAV networks: Architecture, challenges, and opportunities. *IEEE Network*, 35(6):156–162, 2021.

[26] R. Raina, A. Madhavan, and A. Ng. Large-scale deep unsupervised learning using graphics processors. In *ICML*, pages 873–880. ACM, 2009.

[27] Oren Rippel, Michael Gelbart, and Ryan Adams. Learning ordered representations with nested dropout. In *International Conference on Machine Learning*, pages 1746–1754. PMLR, 2014.

[28] Guomei Shi, Li Li, Jun Wang, Wenyan Chen, Kejiang Ye, and ChengZhong Xu. Hysync: Hybrid federated learning with effective synchronization. In *2020 IEEE 22nd International Conference on High Performance Computing and Communications; IEEE 18th International Conference on Smart City; IEEE 6th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, pages 628–633, 2020.

[29] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[30] Branislav Stojkovic, Jonathan Woodbridge, Zhihan Fang, Jerry Cai, Andrey Petrov, Sathya Iyer, Daoyu Huang, Patrick Yau, Arvind Sastha Kumar, Hitesh Jawa, et al. Applied federated learning: Architectural design for robust and efficient learning in privacy aware settings. *arXiv preprint arXiv:2206.00807*, 2022.

[31] Rashish Tandon, Qi Lei, Alexandros G Dimakis, and Nikos Karampatziakis. Gradient coding: Avoiding stragglers in distributed learning. In *International Conference on Machine Learning*, pages 3368–3376. PMLR, 2017.

[32] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *International conference on machine learning*, pages 1058–1066. PMLR, 2013.

[33] Jianyu Wang, Zachary Charles, Zheng Xu, Gauri Joshi, H Brendan McMahan, Maruan Al-Shedivat, Galen Andrew, Salman Avestimehr, Katharine Daly, Deepesh Data, et al. A field guide to federated optimization. *arXiv preprint arXiv:2107.06917*, 2021.

[34] 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.

[35] Sida Wang and Christopher Manning. Fast dropout training. In *international conference on machine learning*, pages 118–126. PMLR, 2013.

[36] Cameron R Wolfe, Jingkang Yang, Arindam Chowdhury, Chen Dun, Artun Bayer, Santiago Segarra, and Anastasios Kyrillidis. GIST: Distributed training for large-scale graph convolutional networks. *arXiv preprint arXiv:2102.10424*, 2021.

[37] Wentai Wu, Ligang He, Weiwei Lin, Rui Mao, Carsten Maple, and Stephen Jarvis. Safa: A semi-asynchronous protocol for fast federated learning with low overhead. *IEEE Transactions on Computers*, 70(5):655–668, 2020.

[38] Cong Xie, Sanmi Koyejo, and Indranil Gupta. Asynchronous federated optimization, 2019.

[39] Chenhao Xu, Youyang Qu, Yong Xiang, and Longxiang Gao. Asynchronous federated learning on heterogeneous devices: A survey, 2021.

[40] Binhang Yuan, Cameron R Wolfe, Chen Dun, Yuxin Tang, Anastasios Kyrillidis, and Chris Jermaine. Distributed learning of fully connected neural networks using independent subnet training. *Proceedings of the VLDB Endowment*, 15(8):1581–1590, 2022.

[41] Hyokun Yun, Hsiang-Fu Yu, Cho-Jui Hsieh, SVN Vishwanathan, and Inderjit Dhillon. Nomad: Non-locking, stochastic multi-machine algorithm for asynchronous and decentralized matrix completion. *arXiv preprint arXiv:1312.0193*, 2013.

[42] Yong Zhuang, Wei-Sheng Chin, Yu-Chin Juan, and Chih-Jen Lin. A fast parallel SGD for matrix factorization in shared memory systems. In *Proceedings of the 7th ACM conference on Recommender systems*, pages 249–256, 2013.