

Fast Server Learning Rate Tuning for Coded Federated Dropout

Giacomo Verardo, Daniel Barreira, Marco Chiesa, Dejan Kostic, Gerald Q. Maguire Jr.

KTH Royal Institute of Technology

{verardo, barreira, mchiesa, dmk, maguire}@kth.se

Abstract

In cross-device Federated Learning (FL), clients with low computational power train a common machine model by exchanging parameters via updates instead of potentially private data. Federated Dropout (FD) is a technique that improves the communication efficiency of a FL session by selecting a *subset* of model parameters to be updated in each training round. However, compared to standard FL, FD produces considerably lower accuracy and faces a longer convergence time. In this paper, we leverage *coding theory* to enhance FD by allowing different sub-models to be used at each client. We also show that by carefully tuning the server learning rate hyper-parameter, we can achieve higher training speed while also achieving up to the same final accuracy as the no dropout case. For the EMNIST dataset, our mechanism achieves 99.6% of the final accuracy of the no dropout case while requiring $2.43\times$ less bandwidth to achieve this level of accuracy.

1 Introduction

In cross-device Federated Learning (FL) [McMahan *et al.*, 2017], the parameter server broadcasts a global machine learning (ML) model to the devices (clients), which in turn perform training over their own datasets. The resulting model updates are sent from the clients to the server, which aggregates the results and may start another FL round. Even in the case of highly heterogeneous client datasets, it has been demonstrated that the model converges [Li *et al.*, 2020]. As the size of a model could be hundreds of MB [Chollet, 2015], Federated Dropout (FD) has been used to both reduce the size of the model and the size of the updates, reducing both communication costs and computational costs at the clients.

Splitting a common, global model between clients and training it collaboratively has become imperative to reduce both memory and computational demands of an FL session. In FD, a global model is divided into sub-models which are trained locally and then merged into an updated global model. FD is orthogonal to message compression techniques, such as quantization [Alistarh *et al.*, 2017] or sparsification [Alistarh

et al., 2018], which do not reduce the computational power and memory needed at the client side.

Reaping the benefits of FD is not easy as it entails solving two main challenges:

- **Low accuracy.** FD learning may result in *lower accuracy* than traditional FL [Caldas *et al.*, 2018]. Intuitively, partially overlapping sub-models per client may improve performance. However, it is not clear how sub-models should be selected, nor which is the orthogonality level which produce the optimal accuracy. Moreover, merging the sub-models into a new global model is a challenging task since it depends on their overlapping.
- **Slow convergence.** Although FD requires less bandwidth per round compared to traditional (no-dropout) FL, there is no guarantee that FD will converge rapidly. If FD requires significantly more rounds than FL to achieve high accuracy, then the promised bandwidth benefits would vanish.

We propose novel techniques to improve the accuracy of FD *without* losing the inherent bandwidth savings offered by FD (*i.e.*, by improving convergence speed). To tackle the above challenges, we explore the following two ideas:

- **Applying coding theory for sub-model selection.** Existing results show that sending random sub-models rather than the same model to each client performs better [Wen *et al.*, 2021]. Building upon the idea of sending different models to different clients during one FD round, we deterministically compute sub-models and then examine whether they perform better than random sub-models. We draw inspirations from *coding theory* (specifically, from the Code Division Multiple Access (CDMA) problem where orthogonal codes enable simultaneous communication channels *without* interference). We employ Gold codes [Gold, 1967] and Constant Weight Codes (CWC) [Brouwer *et al.*, 1990] as masks to drop units and create different sub-models. The intuition is that selecting sub-models using these mechanisms will produce higher accuracy than random selection.
- **Adaptive server learning rate.** We experimentally observe that the convergence speed of an FD session depends on a critical parameter called *server learning rate*, which determines how the weight updates from the clients are incorporated into the trained model. Thanks to the inherent bandwidth savings of FD, we propose to search for the best server learning rate at the beginning of an FD session.

Based on the above ideas, we design a mechanism called Coded Federated Dropout (CFD), which we incorporate alongside existing state-of-the-art FL systems, such as FedAdam [Reddi *et al.*, 2020] and FedAvg [McMahan *et al.*, 2017]. Based on an evaluation with the EMNIST62 dataset, we show that CFD increases the final accuracy of the trained models while preserving the bandwidth savings of FD.

In summary, our contributions are:

- We are the first to leverage coding theory to carefully select the sub-models used in each FL round.
- We show that the optimal server learning rate in a traditional FL session differs from that of an FD session.
- We design a technique to *quickly* search for the best server learning rate. Our evaluation shows that we can identify good server learning rates in just hundreds of rounds.
- We show that CFD with Gold Codes achieves *comparable accuracy* to no-dropout FL with $2.43\times$ *less bandwidth* on the EMNIST dataset.
- We show that minimizing the “cross-correlation” metric in Gold Codes produces better final accuracy than maximizing the “minimal distance” metric of CWC codes.

2 Background

Federated Learning An FL session is composed of one parameter server and multiple clients. At the beginning of an FL round, the server broadcasts a common global model $w_k^{(t)}$ to a fraction of the clients. At round t , each client k trains for a customizable number of epochs E and returns the update $\Delta w_k^{(t)}$ from the previously received weights:

$$\hat{w}_k^{(t)} = w_k^{(t)} - \eta_t \nabla_w L(w, D_k) \quad (1)$$

$$\Delta w_k^{(t)} = \hat{w}_k^{(t)} - w_k^{(t)} \quad (2)$$

where $L(w, D_k)$ is the loss function, which is a function of the model weights and the client dataset D_k . When employing Federated Averaging (FAVG, [McMahan *et al.*, 2017]), the originally proposed aggregation method, the parameter server computes new weights by averaging the updates and adding them to the previous global model:

$$w^{(t+1)} = w^{(t)} + \eta \sum_{j \in S(t)} p_j \Delta w_j^{(t)} \quad (3)$$

$$p_j = \frac{|D_j|}{\sum_{j \in S(t)} |D_j|} \quad (4)$$

For FAVG, η is set to 1.0, whereas it can be different for other aggregation mechanisms.

Federated Dropout One of the major issues of FL is the communication overhead. FD improves bandwidth efficiency by randomly dropping connections between adjacent neural network layers. Unlike standard dropout [Srivastava *et al.*, 2014], FD is not employed as a regularisation tool. Instead, FD keeps a fixed percentage α of activations, thus producing a sub-model with a $(1 - \alpha)^2$ parameters fraction of fully connected networks. The sub-model is trained at the client-side, while the aggregation procedure only involves those nodes

that have been kept. Moreover, the required computational power and memory at the client are reduced. However, although the benefits in bandwidth efficiency are remarkable, the same set of weights is trained at each client per round.

[Bouacida *et al.*, 2020] suggest adapting the selection of dropped nodes based on the loss for each client; however, they state that this approach is unsuitable for FL since it wastes too much memory at the server. Therefore, they propose an alternative technique which employs a single sub-model for all clients, which unfortunately degrades convergence rate and final accuracy. In contrast to random dropout [Wen *et al.*, 2021], we propose the use of coding theory as a tool for mask selection to enhance sub-models’ orthogonality by producing different dropping masks for each client. Hence, we allow partially disjoint sub-models per clients per round to improve both convergence time and final accuracy for the same α .

Code Division Multiple Access Multiple access techniques address the problem of having multiple users communicate via a shared channel. Time and frequency division multiple access respectively splits the channel in time and frequency between users. In contrast, CDMA assigns a different code to each user and allows each of them to use the whole channel. If the codes are sufficiently orthogonal, the inter-user interference is low and transmission occurs with negligible error rate. CDMA has been extensively used in satellite [Taaghoul *et al.*, 1999] and mobile communication [Lee, 1991]. Gold [Gold, 1967] and Kasami [Kasami, 1966] codes are examples of families of sequences designed for orthogonality.

Adaptive Federated Optimization [Reddi *et al.*, 2020] have proven that using a different learning rate for each parameter during aggregation can greatly improve FL model convergence. They propose 3 aggregation methods (FedAdam, FedAdagrad, and FedYogi), which replace Eq.3. Here we describe only the FedAdam algorithm, which performs well in all the datasets:

$$\Delta^t = \beta_1 \cdot \Delta^{t-1} + (1 - \beta_1) \cdot \sum_{j \in S(t)} p_j \Delta w_j^{(t)} \quad (5)$$

$$v^{(t)} = \beta_2 \cdot v^{(t-1)} + (1 - \beta_2) \cdot \Delta^{(t)^2} \quad (6)$$

$$w^{(t+1)} = w^{(t)} + \eta \frac{\Delta^{(t)}}{\sqrt{v^{(t)}} + \tau} \quad (7)$$

where β_1, β_2 and τ are hyper-parameters. The key insight is that training variables which have been trained less in the previous rounds will improve convergence. For this reason, v_t stores an indication of how much variables have been trained and is used to independently scale each component of the next update $\Delta^{(t)}$. However, the proposed optimization techniques require expensive hyper-parameters tuning and therefore a considerable amount of time.

Other adaptive mechanisms have been proposed to correct the client drift due to the statistical heterogeneity in the clients datasets. [Karimireddy *et al.*, 2021] employ variance reduction, but this requires too much information to be stored server-side. [Li *et al.*, 2021] propose a trade-off between fairness and robustness of the global model.

3 Methodology

This section describes the proposed Coded Federated Dropout (CFD) method which performs both tuning of the server learning rate η (Sect. 3.1) and the selection of the sub-models sent to the clients (Sect. 3.2).

3.1 Fast server learning rate adaptation

Similarly to centralized ML, increasing the server learning rate may lead to faster convergence, but further increasing the learning rate causes the objective function to diverge [Zeiler, 2012]. Fig. 1(a) empirically confirms that this is also the case for no-dropout FL where a high server learning rate of $\eta = 3$ exhibits worse convergence than with $\eta = 2$. This result is based on the EMNIST62 dataset with more details in Sect. 4. Interestingly, Fig. 1(b) shows that in FD with random sub-models increasing the server learning rate to 3 leads to faster convergence. This shows that the “best” server learning rate for FD may differ from the no-dropout case.

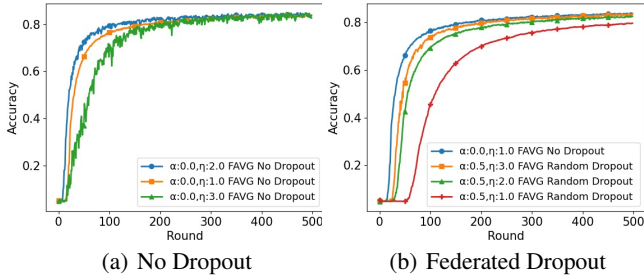


Figure 1: Increasing the server learning rate η to 3.0 is beneficial for random FD with different sub-models per client, while it is detrimental for the no dropout case

We propose a fast server learning rate adaptation method, which can also be extended to other parameters. At the beginning of training, we run Algorithm 1, which requires n_a adaptation steps. In each step (line 2), multiple FL sessions are launched in parallel from the same parameter server with different server learning rates \mathbb{H} and, in general, different clients subsets per round. We start our search using three η values during the first adaptation step (line 1) and reduce it to two server learning rates in the following adaptation steps (line 14). The goal of this search (lines 4–12) is to find the server learning rate that reaches a preconfigured accuracy target γ (lines 8–9) in the minimum number of rounds r^* (line 5). More specifically, in each round r , the server collects both the gradient update Δw_k^t and the accuracy of the model for each training client in the round (line 6). Then, it computes the average of the median training accuracy $\bar{\gamma}$ in the last q FL rounds (line 7). The median operation is performed in order to avoid the impact of outliers (*i.e.*, clients with too high or low training accuracies), when the average over the last rounds avoids sudden spikes. If one server learning rate $\bar{\gamma}$ is higher than a predefined threshold γ^* , then we have found a new optimal server learning rate $\eta^* = \eta$ that requires the new minimum number of rounds $r^* = r$ to achieve the target accuracy (lines 8–9). For the next adaptation step, the new

tentative server learning rates \mathbb{H} are chosen near η^* (lines 15–16) and the next adaptation step is performed. An adaptation step may also end when all the FL sessions produce $r \geq r^*$ (line 5). Worth noting is that the search at lines 3–13 can be done in parallel to improve convergence speed. This algorithm reduces the number of rounds compared to testing all possible server learning rates using full FL sessions. In particular, since sessions are aborted when $r \geq r^*$, the overhead introduced by each adaptation step is limited. Assuming the parallel search is synchronized round-by-round, the additional overhead in number of rounds of our algorithm is:

$$3 \cdot r_0^* + 2 \sum_{i=1}^{n_a} r_i^* - r^* \quad (8)$$

where r_i^* is the minimum number of rounds at the end of the adaptive step i . The first term accounts for the first adaptive step, the second terms for the following adaptive steps, and the third term for the spared training rounds when running the full simulation with $\eta = \eta^*$.

Algorithm 1 selects the optimal η^* in terms of number of rounds to reach the target accuracy. The Round function at line 6 represents the underlying FL mechanism being used to compute the trained model, for instance, FAVG or FedAdam.

We argue that running full simulations to achieve the same level of granularity takes $T \times$ the number of rounds per simulation, where T is the number of tried server learning rates. This value is strictly greater than the bound provided by equation 8.

Algorithm 1 Fast server learning rate adaptation

Input: $w^0, \{D_k \forall k \in \{1, \dots, T\}\}$

Parameter: $\gamma^*, q, n_a, \eta_0, \Delta\eta$

Output: η^*

```

1:  $\mathbb{H} \leftarrow \{\eta_0, \eta_0 - \Delta\eta, \eta_0 + \Delta\eta\}$ 
2: for  $s = 0$  to  $n_a$  do
3:   for  $\eta \in \mathbb{H}$  parallel do
4:      $r \leftarrow 0$ 
5:     while  $r < r^*$  do
6:        $\bar{\gamma}^t, w^{(t+1)} \leftarrow \text{Round}(\{D_k\}, w^{(t)})$ 
7:        $\bar{\gamma} \leftarrow \frac{1}{q} \cdot \sum_{i=0}^{q-1} \bar{\gamma}^{t-i}$ 
8:       if  $\bar{\gamma} \geq \gamma^*$  then
9:          $r^* \leftarrow r, \eta^* \leftarrow \eta$ 
10:        wait for parallel search to end; go to line 15
11:      end if
12:       $r \leftarrow r + 1$ 
13:    end while
14:  end for
15:   $\Delta\eta \leftarrow \frac{\Delta\eta}{2}$ 
16:   $\mathbb{H} \leftarrow \{\eta^* - \Delta\eta, \eta^* + \Delta\eta\}$ 
17: end for
```

3.2 Coded Federated Dropout

We reduce the size of the model by dropping weights from each layer by associating to each client k and model layer i in the FL round a binary mask vector $c_{ij}^k \in \mathbf{R}^{N_i}$. A unit is dropped or kept when the component c_{ij}^k is equal to 0 or 1

respectively. For adjacent fully connected layers (Fig. 2) the dropped weights can be straightforwardly obtained by eliminating rows and columns corresponding to the dropped units from the previous and following layer respectively. As in standard FD, we only drop a fraction α of nodes per layer i , which produces the same model size for all clients. For instance, in Fig. 2 we have $N_i = N_{i+1} = 5$ and $\alpha = 2/5$ and therefore only $N_i \cdot N_{i+1} \cdot (1 - \alpha)^2 = 25 \cdot \frac{9}{25} = 9$ weights should be transmitted instead of $N_i \cdot N_{i+1} = 25$.

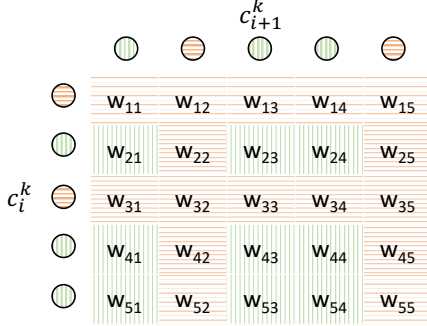


Figure 2: Federated Dropout for masks $c_i^k=[0,1,0,1,1]$ and $c_{i+1}^k=[1,0,1,1,0]$ reduces the weights to be trained from 25 to 9

The problem to be solved is to obtain a matrix C_i per layer i with the following properties:

- each row of C_i is a codeword $c_i^k \in \mathbf{R}^{N_i}$;
- the Hamming weight (*i.e.*, the number of ones in the codeword) of each row in C_i is equal to $N_i \cdot (1 - \alpha)$; and
- the number of rows in C_i is greater or equal than the number of clients per round M .

We consider 4 methods to compute C_i : (i) same random codeword for each client (baseline Federated Dropout), (ii) different random codeword for each client (proposed contemporaneously by [Wen *et al.*, 2021]), (iii) Gold sequences, and (iv) CWCs. While the first two are straightforward, the other two provide different levels of orthogonality between the dropped models. Since sub-models are trained independently and then aggregated, having partially non overlapping sub-models reduces the impact of heterogeneous updates.

In CFD we exploit the different masks per client. At the beginning of each training round, we compute one matrix C_i for each layer i . Client k is assigned the k -th row from each C_i and the correspondent weights are extracted from the global model. If such a matrix is burdensome to be computed or if it would be the same after the generation process, then the rows and columns of the matrix are shuffled instead. In that way, the excess codewords can be employed when the number of rows of C_i is greater than M .

Applying coded masks to neural networks is not straightforward, since it requires different approaches according to the kind of employed layers. For *fully connected* layers, it is sufficient to apply dropout as in Fig. 2. For *convolutional* layers, we experimentally observed that dropping entire filters rather than individual weights achieves better performance.

Long Short-Term Memory layers [Hochreiter and Schmidhuber, 1997], which are composed of multiple gates, require that the same mask is employed for each set of weights.

After each sub-model has been trained, each weight in the global model is updated by averaging the contribution of each sub-model which contained that weight.

Gold codes

Codewords orthogonality may be defined by means of cross-correlation. The correlation between two real binary sequences u^1 and u^2 of length L_u is a function of the shift l :

$$R(u_1, u_2, l) = \frac{1}{L_u} \sum_{j=1}^{L_u} u_j^1 \cdot u_{(j+l) \bmod L_u}^2 \quad (9)$$

$$u_j^1, u_j^2 \in \{-1, 1\} \quad \forall j = 1, \dots, L_u$$

Gold codes are generated from two Linear Feedback Shift Registers (LFSR) with suitable feedback polynomials and initial conditions. The LFSRs produce two m-sequences, which are then circularly shifted and element-wise xored to produce all the sequences in the family. The size n_{LFSR} of the LFSR determines the code length $2^{n_{\text{LFSR}}} - 1$, the number of codewords in the set $2^{n_{\text{LFSR}}} + 1$, and the upper bounds for the maximum cross-correlation in the set:

$$\max_l |R(c_i^{k_1}, c_i^{k_2}, l)| = 2^{\lfloor (n_{\text{LFSR}} + 2)/2 \rfloor} + 1 \quad (10)$$

$$\forall k_1, k_2 \in \{1, \dots, (2^{n_{\text{LFSR}}} + 1)\}, \quad k_1 \neq k_2$$

After computing the sequences, we concatenate them as row vectors in matrix C_i , which will then have $2^{n_{\text{LFSR}}} + 1$ rows and $2^{n_{\text{LFSR}}}$ columns. Although Gold codes provide orthogonality, they have constraints on the size of C_i and α value, which can only be 50% since most Gold sequences are balanced (*i.e.*, Hamming weight $2^{n_{\text{LFSR}} - 1}$). Please refer to [Sarwate and Pursley, 1980] for details on constructions and properties of Gold codes and LFSR-generated sequences.

Constant weight codes

Another metric is the Hamming distance between two codewords u^1 and u^2 , which is the number of ones in $u^1 \oplus u^2$:

$$d_h(u^1, u^2) = \sum_{j=1}^{L_u} u_j^1 \oplus u_j^2 \quad (11)$$

We provide a method to create a matrix C_i with size $M \times N_i$ where, in order to improve orthogonality, the minimum Hamming distance between rows is maximized. CWC are a family of non-linear codes where each sequence has a fixed Hamming weight (*i.e.*, number of ones). CWC are flexible: they can provide sets of codewords with any cardinality, sequence length, and Hamming weight. We devise a variant of the algorithm from [Montemanni and Smith, 2009] to generate M codewords with length N_i and weight $(1 - \alpha) \cdot N_i$ by maximizing the minimum distance. Starting from the maximum possible minimum distance, we iteratively compute a codewords set of size M . If this is not feasible, we reduce the required minimum distance and repeat the process. The algorithm details can be found in [Verardo *et al.*, 2022].

4 Evaluation

We run our code in vanilla TensorFlow [Abadi *et al.*, 2016] and Python3, since the major frameworks for FL do not support FD. In particular, although TensorFlow Federated [tensorflow.org, 2017] allows FD with the same dropping masks, it does not allow broadcast of different models to the clients (which we require for CFD). The generation algorithm for CWC was implemented in Matlab. Training was performed on the EMNIST dataset [Cohen *et al.*, 2017] with convolutional model C [Springenberg *et al.*, 2015]. Preprocessing and full model description are provided in [Verardo *et al.*, 2022]. We use four kinds of codes for CFD with dropout fraction $\alpha = 0.5$: random with same sub-model for each client (baseline FD), random with different sub-models, Gold and CWC.

The fast server learning rate tuning algorithm achieves consistent optimal η values across many simulations. We run 10 training sessions for each coded approach with target accuracy $\gamma^* = 20/62$ (which is 20 times the random accuracy) and showcase the results in Table 1. Whereas for FAVG the selected η is the same for all simulations (except for the baseline FD), FedAdam produces higher variability. However, the differences between the different selected η^* amounts to a maximum of 0.5 in log scale for each code.

Table 1: Selected η values for 10 simulations and different codes and aggregation algorithms

Server Learning Rate (Log10)	FedAdam					FAVG	
	-2.25	-2	-1.75	-1.5	0.25	0.5	
$\alpha : 0.0$ Fedadam No Dropout	0%	30%	60%	10%	100%	0%	
$\alpha : 0.5$ Random Fedadam	0%	20%	30%	50%	0%	100%	
$\alpha : 0.5$ CWC FedAdam	0%	10%	40%	50%	0%	100%	
$\alpha : 0.5$ Gold FedAdam	0%	0%	80%	20%	0%	100%	
$\alpha : 0.5$ Fedadam + Baseline FD	50%	50%	0%	0%	70%	30%	

The optimal server learning rate in a traditional FL session is greater than that of an FD session, especially when a different sub-model per client is employed. We experiment with 10 simulations with our tuning algorithm and keep track of the number of rounds to reach γ^* for each experimented η . Fig 3 shows the average number of rounds for each η for both FedAdam and FAVG and makes evident that the η producing the minimum number of rounds is greater for the coded approaches compared to the no dropout. Moreover, although the no dropout case is still the fastest one, the coded approaches achieve up to $1.5\times$ speedup to reach γ^* compared to the dropout baseline, thus improving convergence time.

Our tuning mechanism saves communication resources compared to running full FL sessions with different values of η . We compute the number of additional rounds as in Eq. 8 and report the results in Table 2. In FedAdam₁₀, γ^* is 10 times the random accuracy (10/62) instead of 20. The overhead is directly dependent on the convergence speed of the model. Consequently, the no dropout case requires the least overhead and the baseline FD the most. Still, the additional number of rounds is much lower than running multiple full FL sessions. For both FedAdam and FAVG, our tuning algorithm tests 10 η values. Therefore, running full sessions would require $500 \cdot (10 - 1)$ additional rounds. We point out

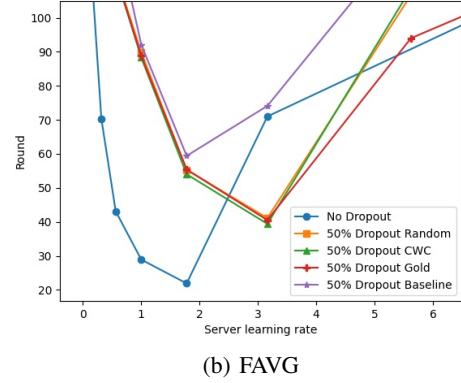
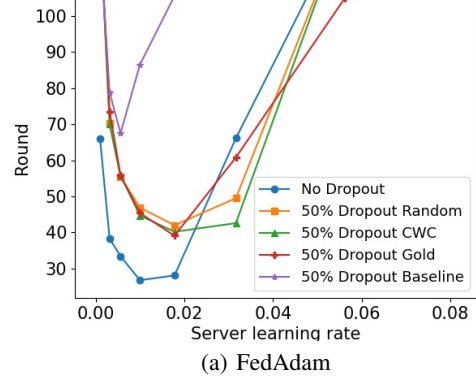


Figure 3: The dropout approaches require a higher η in order to reach the minimal number of rounds for the target accuracy

that decreasing the accuracy threshold γ^* to 10/62 notably reduces the additional number of rounds (FedAdam₁₀). The best selected η is only slightly influenced by changing γ^* , thus demonstrating that tuning the threshold value is much easier than tuning η directly.

Table 2: Average number of additional rounds for different codes

	No Drop	Rand	CWC	Gold	FD
FedAdam₂₀	154.9	262.5	248.6	259.3	456.3
FAVG₂₀	166.4	393.0	383.1	387.9	479.3
FedAdam₁₀	140.7	219.2	225.6	211.9	351.2

Gold codes outperform other FD approaches for FedAdam and achieve 99.6% of the final accuracy of the no dropout case while saving $> 2\times$ bandwidth. Fig. 4 shows the average test accuracy of simulations run with the previously selected η^* values. While the benefits of using Gold codes or CWC is negligible in terms of final accuracy compared to random for FAVG, FedAdam plus Gold codes produces higher convergence speed. Moreover, Gold FedAdam reaches 99.6 % of the final accuracy of the no dropout case while saving almost $1 - (1 - \alpha)^2 = 75\%$ of the bandwidth per round. The best result is achieved

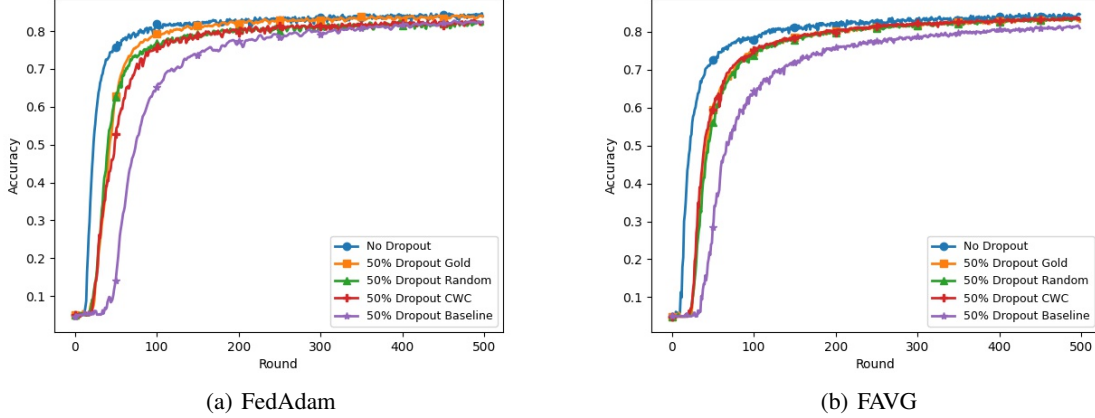


Figure 4: Average test accuracy of FedAdam and FAVG for 5 simulations with the selected η^* from Table 1. Coded Federated Dropout with Gold codes and FedAdam improves convergence rate and final accuracy compared to random, CWC and baseline FD approaches.

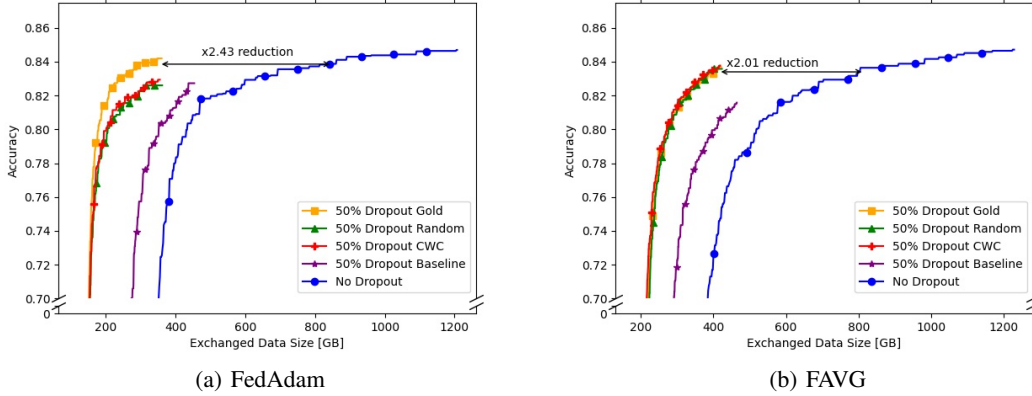


Figure 5: Reachable test accuracy for FedAdam and FAVG given the amount of exchanged bytes in a FL session. Coded Federated Dropout with Gold codes and FedAdam reaches the same level of final accuracy as no dropout while reducing bandwidth usage by $2.43\times$.

by FAVG without dropout (84.1% accuracy) when averaging over the last 100 rounds, while Gold codes achieve 83.8% for FedAdam, which is the 99.6%. Conversely, CWC does not perform well for FedAdam, achieving even worse performance than random codes. Regarding the overall bandwidth, we measure the size of the exchanged sub-models and compute the amount of gigabytes needed to reach a certain test accuracy. Fig.5 shows the reduction in overall bandwidth when CFD is employed with the selected η^* values. Gold codes plus FedAdam reduces the bandwidth needed to reach the maximum test accuracy by $2.43\times$ compared to no dropout, while CFD plus FAVG by $2.01\times$.

Minimizing cross-correlation instead of maximizing minimal distance provides greater sub-model orthogonality. Figures 4 and 5 show that Gold codes always outperform CWC. Therefore, codes built by minimizing cross-correlation produce higher final accuracy and convergence rate than the ones obtained by maximizing the minimum distance. Nev-

ertheless, optimizing any of the two metrics outperforms the baseline for federated dropout for both FedAdam and FAVG.

5 Conclusion and future works

We have presented a fast server learning rate tuning algorithm for Federated Dropout and shown considerable reduction on the number of rounds to assess the optimal η^* . Moreover, we have shown that convergence rate and final accuracy of models trained in a FL session are improved when using coding theory to carefully perform Federated Dropout. Specifically, CFD with Gold sequences paired with an optimization mechanism such as FedAdam can achieve up to the same accuracy of the no dropout case, with $2.43\times$ bandwidth savings. However, Gold codes have specific lengths and Hamming weights, so they are not flexible enough, while CWC does not improve performance compared to random dropout. Hence, future work will investigate further sequences from coding theory for FD.

References

- [Abadi *et al.*, 2016] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283, 2016.
- [Alistarh *et al.*, 2017] Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. Qsgd: Communication-efficient sgd via gradient quantization and encoding, 2017.
- [Alistarh *et al.*, 2018] Dan Alistarh, Torsten Hoefer, Mikael Johansson, Sarit Khirirat, Nikola Konstantinov, and Cédric Renggli. The convergence of sparsified gradient methods, 2018.
- [Bouacida *et al.*, 2020] Nader Bouacida, Jiahui Hou, Hui Zang, and Xin Liu. Adaptive federated dropout: Improving communication efficiency and generalization for federated learning, 2020.
- [Brouwer *et al.*, 1990] A.E. Brouwer, J.B. Shearer, N.J.A. Sloane, and W.D. Smith. A new table of constant weight codes. *IEEE Transactions on Information Theory*, 36(6):1334–1380, 1990.
- [Caldas *et al.*, 2018] Sebastian Caldas, Jakub Konečný, H. Brendan McMahan, and Ameet Talwalkar. Expanding the reach of federated learning by reducing client resource requirements. *CoRR*, abs/1812.07210, 2018.
- [Caldas *et al.*, 2019] Sebastian Caldas, Sai Meher Karthik Duddu, Peter Wu, Tian Li, Jakub Konecny, H. Brendan McMahan, Virginia Smith, and Ameet Talwalkar. Leaf: A benchmark for federated settings, 2019.
- [Chollet, 2015] François Chollet. Keras. <https://github.com/fchollet/keras>, 2015.
- [Cohen *et al.*, 2017] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and André van Schaik. Emnist: an extension of mnist to handwritten letters, 2017.
- [Gold, 1967] R. Gold. Optimal binary sequences for spread spectrum multiplexing (corresp.). *IEEE Transactions on Information Theory*, 13(4):619–621, 1967.
- [Hochreiter and Schmidhuber, 1997] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
- [Karimireddy *et al.*, 2021] Sai Praneeth Karimireddy, Satyen Kale, Mehryar Mohri, Sashank J. Reddi, Sebastian U. Stich, and Ananda Theertha Suresh. Scaffold: Stochastic controlled averaging for federated learning, 2021.
- [Kasami, 1966] Tadao Kasami. Weight distribution formula for some class of cyclic codes. *Coordinated Science Laboratory Report no. R-285*, 1966.
- [Lee, 1991] W.C.Y. Lee. Overview of cellular cdma. *IEEE Transactions on Vehicular Technology*, 40(2):291–302, 1991.
- [Li *et al.*, 2020] Xiang Li, Kaixuan Huang, Wenhao Yang, Shusen Wang, and Zhihua Zhang. On the convergence of fedavg on non-iid data, 2020.
- [Li *et al.*, 2021] Tian Li, Shengyuan Hu, Ahmad Beirami, and Virginia Smith. Ditto: Fair and robust federated learning through personalization. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 6357–6368. PMLR, 18–24 Jul 2021.
- [McMahan *et al.*, 2017] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data, 2017.
- [Montemanni and Smith, 2009] Roberto Montemanni and Derek H. Smith. Heuristic algorithms for constructing binary constant weight codes. *IEEE Transactions on Information Theory*, 55(10):4651–4656, 2009.
- [Reddi *et al.*, 2020] Sashank Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečný, Sanjiv Kumar, and H. Brendan McMahan. Adaptive federated optimization, 2020.
- [Sarwate and Pursley, 1980] D.V. Sarwate and M.B. Pursley. Crosscorrelation properties of pseudorandom and related sequences. *Proceedings of the IEEE*, 68(5):593–619, 1980.
- [Springenberg *et al.*, 2015] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net, 2015.
- [Srivastava *et al.*, 2014] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, January 2014.
- [Taaghool *et al.*, 1999] P. Taaghool, B.G. Evans, E. Buracchini, G. De Gaudinaro, Joon Ho Lee, and Chung Gu Kang. Satellite umts/imt2000 w-cdma air interfaces. *IEEE Communications Magazine*, 37(9):116–126, 1999.
- [tensorflow.org, 2017] tensorflow.org. Tensorflow federated. <https://www.tensorflow.org/federated>, 2017.
- [Verardo *et al.*, 2022] Giacomo Verardo, Daniel Barreira, Marco Chiesa, and Dejan Kostic. Fast server learning rate tuning for coded federated dropout, 2022.
- [Wen *et al.*, 2021] Dingzhu Wen, Ki-Jun Jeon, and Kaibin Huang. Federated dropout – a simple approach for enabling federated learning on resource constrained devices, 2021.
- [Zeiler, 2012] Matthew D. Zeiler. Adadelta: An adaptive learning rate method, 2012.

6 Appendix

6.1 Notation and parameter values

The main system parameters employed throughout the paper and the related notation are summarised in Table 3.

6.2 Employed polynomials for Gold codes

To generate Gold sequences, we employ the generation mechanism from Sec. 3.2 with the preferred polynomials pairs listed in Table 4. The table also contains the correspondent length of the generated gold sequences. In order to have a suitable length for the model layers, we usually pad each resulting sequence with a 0 value after the longest run of zeros. Also note that Gold sequences with multiple of 4 degrees are not supported. Fig. 6 provides an example of the LFSR used to generate Gold codes of length 31.

6.3 Generation Algorithm for Constant Weight Codes

CWC are a non-linear class of codes. Non-linearity implies that the generation of the sequences will be partially randomized, since a new codeword to be added in the set does not depend on the previously selected codewords. Hence, we follow the indications from [Montemanni and Smith, 2009] and develop Algorithm 2. Starting from a random codeword, we iteratively select new sequences with a fixed distance from the current set. The sequences are added in lexicographic order. If the final set is not complete (i.e., a set of M sequences with weight $(1 - \alpha) \cdot N_i$ and length N_i does not exist for the given minimum distance), the required minimum distance is decremented and the selection procedure is performed again. The algorithm is described in 2, where with an abuse of notation we identify C_i as a set of codewords instead of a matrix.

Algorithm 2 Constant Weight Code Generation

Input: N_i, M, α, t_{max}

Output: C_i

- 1: Let $t = 0$.
 - 2: $d_{min} = N_i, C_i = \{\}$
 - 3: $F_i = \{f \in \{0, 1\}^{N_i} : w_h(f) = (1 - \alpha) \cdot N_i\}$, f added in lexicographic order
 - 4: Add random codeword from F_i to C_i
 - 5: **while** $t < t_{max}$ **do**
 - 6: $F_i = \{f \in F_i : d_h(f, c) \geq d_{min} \forall c \in C_i\}$
 - 7: **if** $|F_i| \neq 0$ **then**
 - 8: Add first sequence from F_i to C_i
 - 9: **else**
 - 10: $d_{min} = d_{min} - 2, C_i = \{\}$
 - 11: $F_i = \{f \in \{0, 1\}^{N_i} : w_h(f) = (1 - \alpha) \cdot N_i\}$, f added in lexicographic order
 - 12: Add random codeword from F_i to C_i
 - 13: **end if**
 - 14: **if** $|C_i| = M$ **then**
 - 15: **return** C_i
 - 16: **end if**
 - 17: **end while**
 - 18: **return** C_i
-

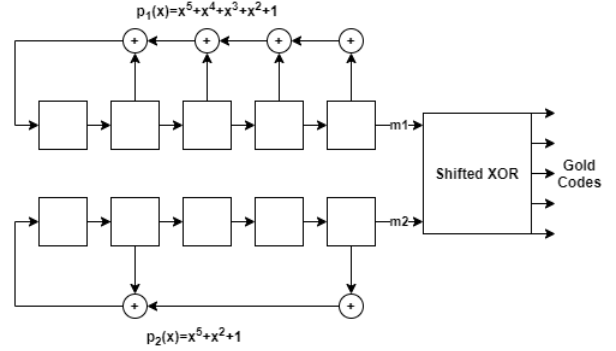


Figure 6: Gold code generation for preferred feedback polynomials $p_1(x) = x^5 + x^4 + x^3 + x^2 + 1$ and $p_2(x) = x^5 + x^2 + 1$. The output is a set of $2^5 + 1 = 32$ sequences of length $2^5 - 1 = 31$

6.4 Datasets and models

EMNIST62

We use the EMNIST character and digits recognition dataset provided by [Caldas *et al.*, 2019; tensorflow.org, 2017], which includes grey scale images of size 28x28. We normalize each image in the $[0, 1]$ interval and use a batch size of 10. We provide a description of the employed model (with total number of parameters of 6,603,710) in Table 5. Each client optimize the sparse categorical crossentropy loss with the SGD optimizer. We train the model for 500 rounds.

6.5 Practical notions about Federated Dropout

As in [Caldas *et al.*, 2018], we perform federated dropout by dropping units (for dense layers) or filters (for convolutional layers) on each layer except the first and the last one.

Table 3: Adopted notations and principal symbols

Symbol	Description	Value
N_i	Number of units in model layer i	See Tab. 5
T	Number of total clients	3400
M	Number of clients per round	35
α	Dropout fraction for FD	0.5
η_l	Client learning rate	0.035
E	Training epochs per client	1
η	Server learning rate	Optimized
β_1	Momentum parameter for FedAdam	0.90
β_2	Momentum parameter for FedAdam	0.99
τ	Adaptivity degree for FedAdam	0.001
$w^{(t)}$	Server weights at round t	-
$w_k^{(t)}$	Initial client k weights at round t	-
$\hat{w}_k^{(t)}$	Final client k weights at round t	-
D_k	Dataset for client k	-
$\{\xi\}_k^{(t)}$	Set of batches for client k at round t	-
$L(\cdot)$	Loss function	-
$S(t)$	Set of clients selected at round t	-
c_i^k	Binary mask for client k and layer i	-
c_{ij}^k	Component j of c_i^k	-
C_i	Codes Matrix per layer i	-
n_{LFSR}	Size of the LFSR	See Tab.4
$R(u_1, u_2, l)$	Correlation between u_1 and u_2 for shift l	-
$d_h(u^1, u^2)$	Hamming distance between u_1 and u_2	-
$w_h(u)$	Hamming weight of u	-
γ^*	Target accuracy	$\frac{20}{62}$ or $\frac{10}{62}$
γ_k^t	Training accuracy for client k at round t	-
$\bar{\gamma}^t$	Median training accuracy at round t	-
$\bar{\gamma}$	Average of $\bar{\gamma}^t$ at round t	-
q	Rounds number to compute $\bar{\gamma}$	-
n_a	Number of adaptation steps	3
η^*	Best server learning rate	See Tab.1
$\Delta\eta$	Log distance between tentative η values	-
$\Delta\eta_0$	Initial $\Delta\eta$	1
\mathbb{H}	Set of tentative η values	-
r	Current round number	-
r^*	Best round number to reach γ^*	-
r_i^*	Best round number to reach γ^* at step i	-

Table 4: Preferred polynomial pairs for Gold codes generation

Degree	Sequence length	Polynomial 1	Polynomial 2
5	31	$1 + x^2 + x^5$	$1 + x^2 + x^3 + x^4 + x^5$
6	63	$1 + x^6$	$1 + x^1 + x^2 + x^5 + x^6$
7	127	$1 + x^3 + x^7$	$1 + x^1 + x^2 + x^3 + x^7$
9	511	$1 + x^4 + x^9$	$1 + x^3 + x^4 + x^6 + x^9$
10	1023	$1 + x^3 + x^{10}$	$1 + x^2 + x^3 + x^8 + x^{10}$
11	2047	$1 + x^2 + x^5 + x^8 + x^{11}$	$1 + x^2 + x^{11}$

Table 5: EMNIST62 model summary

Layer Type	Output Shape	Param #	Activation	Hyper-parameters
Conv2D	(-, 28, 28, 32)	832	Relu	Num filters: 32 Kernel size: (5,5) Padding: Same
MaxPooling2D	(-,14,14,32)	0	-	Pool size: (2,2) Padding: Valid
Conv2D	(-, 14, 14, 64)	51264	Relu	Num filters: 64 Kernel size: (5,5) Padding: Same
MaxPooling2D	(-,7,7,64)	0	-	Pool size: (2,2) Padding: Valid
Flatten	(-, 3136)	0	-	
Dense	(-, 2048)	6424576	Relu	Num units: 2048
Dense	(-, 62)	127038	-	Num units: 62