Model Pruning Enables Efficient Federated Learning on Edge Devices

A Brief Summary

Rómulo Condori

Institute of Computing

r204185@dac.unicamp.br

Agenda

Introduction

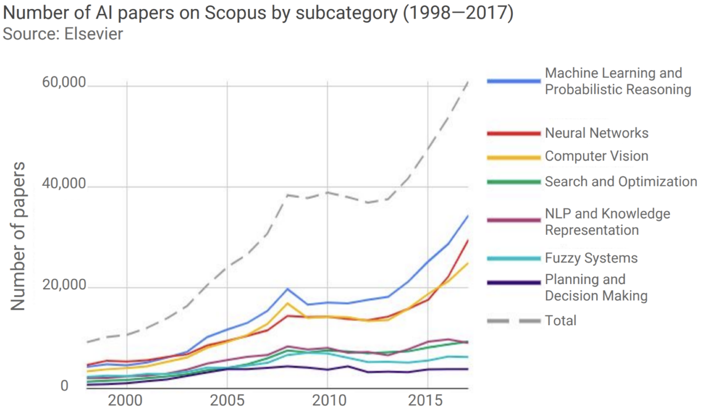


Figure: Growing in machine learning publications ([1])

Data and Deeep Learning

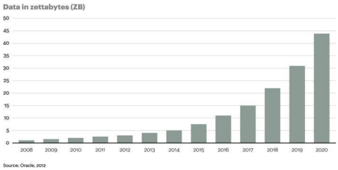


Figure: Data volume growth by year in zettabytes ([2])

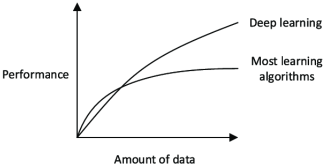


Figure: Comparison Amount data and the performance ([4])

Federated Learning (Fl)

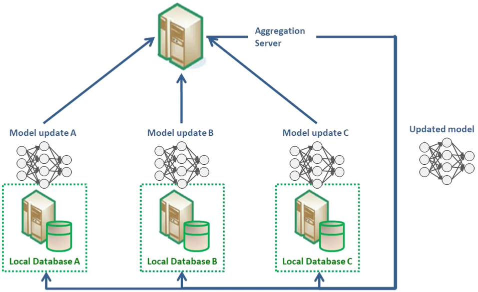


Figure: Standard architecture Federated Learning ([3])

Federated Learning (Fl)

How can we perform FL efficiently so that a model is trained within a reasonable amount of time and energy?

Types of Prunning

In machine learning, pruning is removing unnecessary neurons or weights.

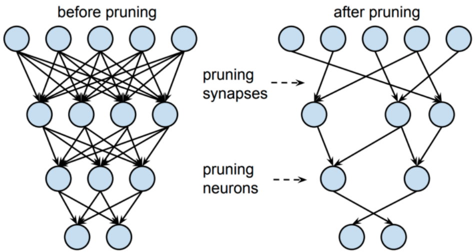


Figure: (1) Trim weights. Setting parameters to zero This would reduce model parameters while maintaining the architecture. (2) Network nodes can be removed. This would reduce network architecture while maintaining accuracy. [6].

Types of Prunning

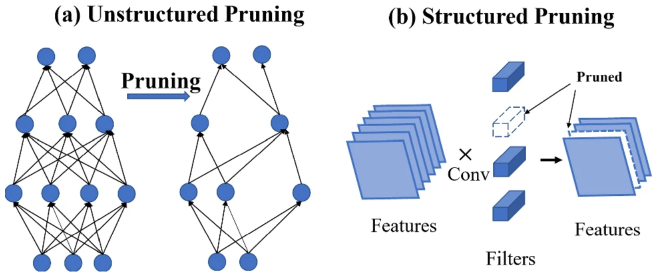


Figure: The schematic illustration of two types of network pruning methods according to granularity. a In unstructured pruning, neuron leveled connections are pruned. b A type of structured pruning on the right shows a example that filters are pruned [7].

The Lottery Ticket Hypothesis

A randomly-initialized, dense neural network contains a subnetwork that is initialized such that—when trained in isolation—it can match the test accuracy of the original network after training for at most the same number of iterations.

Federated Learning Formulation

Let FL a federated system with N clients.

$$n \in [N] := \{1, 2, \ldots, N\},$$
$$F_n(w) := \frac{1}{D_n} \sum_{i \in \mathcal{D}_n} f_i(w), \text{Local Empirical Risk (LER)}$$

Where $f_i(w)$ is the lost function and $D_n := |\mathcal{D}_n|$
The system tries to find a parameter w that minimizes the global empirical risk (GER), w is a model parameter vector

$$\min_w F(w) := \sum_{n \in [N]} p_n F_n(w) \tag{1}$$

Where $p_n > 0$ are weights such that $\sum_{n \in [N]} p_n = 1$
$f_i(w)$ Captures the difference between the model output and data sample i's desired result.

Federated Learning Formulation

Example

Let $\mathcal{D}_n \cap \mathcal{D}_{n'} = \emptyset$ for $n \neq n'$ and $p_n = D_n/D$ with $\mathcal{D} := \bigcup_n \mathcal{D}_n$ and $D := |\mathcal{D}|$ we have:

$$F(w) = \frac{1}{D} \sum_{i \in \mathcal{D}} f_i(w)$$

- Each client n has a local parameter $w_n(k)$ in iteration k.
- FL usually involves stochastic gradient descent (SGD) to make multiple updates of $w_n(k)$ on the scope of LER.

Main Notation

Notation	Definition
$\odot$	element-wise product of two vectors
n, N	client index, the total number of clients
k, K	iteration index, the total number of iterations
I	number of local iterations
p_n	weight for client $n(p_n > 0, \forall n, and \sum_n p_n = 1)$
$m(k)$	weight mask in iteration k (universal for all clients)
$w_n(k)$	client n's parameter in iteration k
$w(k)$	$w(k) := \sum_{n=1}^{N} p_n w_n(k)$
$w'_n(k), w'(k)$	$w'_n(k) = w_n(k)$
$g_n(w)$	client n's stochastic gradient with parameter w
$\nabla F_n(w)$	client n's expected gradient with parameter w
$\nabla F(w)$	$\nabla F(w) = \sum_{n=1}^{N} p_n F_n(w)$

Table: Main Notations

PruneFL

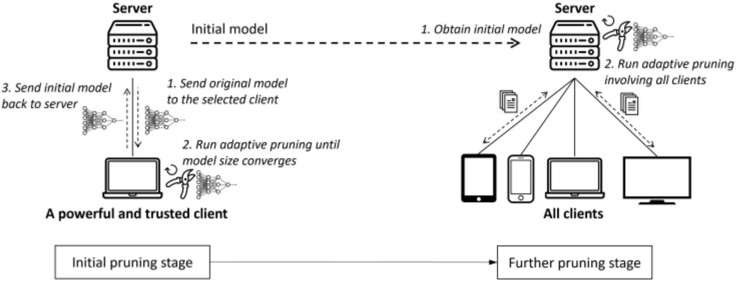


Figure: Illustration and flowchart of PruneFL ([10])

PruneFL-algorithm

- PruneFL includes Two-Stages Distributed Pruning:
  - Initial pruning at a single client with high computing capabilities can be done with biased data, and additional pruning will "remove" the bias and enhance the model.
  - Further pruning involving both the server and clients during the FL process.
- Adaptive Pruning
  - The adaptive pruning approach prunes by adding and deleting parameters.

Adaptive Pruning

Algorithm 1 Adaptive Pruning

for $k = 0, \ldots, K-1$ **do**
  Initialize the set of importance measure on each client: $\mathcal{Z}_n \leftarrow \emptyset, \forall n$;
  for *each client n, in parallel* **do**
    Compute stochastic gradient $\mathbf{g}_n(\mathbf{w}'_n(k)) := \mathbf{g}_n(\mathbf{w}_n(k) \odot \mathbf{m}(k))$;
    Update local parameters: $\mathbf{w}_n(k+1) \leftarrow \mathbf{w}'_n(k) - \eta \mathbf{g}_n(\mathbf{w}'_n(k)) \odot \mathbf{m}(k)$;
    Add importance measure $\mathbf{z}_n$ to $\mathcal{Z}_n$: $\mathbf{z}_n := \mathbf{g}_n(\mathbf{w}'_n(k)) \odot \mathbf{g}_n(\mathbf{w}'_n(k))$; $\mathcal{Z}_n \leftarrow \mathcal{Z}_n \cup \mathbf{z}_n$;
  if $I \mid k+1$ **then**
    Each client n sends $\mathbf{w}_n(k+1)$ to the server;
    Server aggregates the parameters from each client: $\mathbf{w}(k+1) \leftarrow \sum_{n=1}^{N} p_n \mathbf{w}_n(k+1)$;
    if $k+1$ *is reconfiguration iteration* **then**
      Each client sends the averaged importance measure $\overline{\mathbf{z}}_n := (\sum_{\mathbf{z}_n \in \mathcal{Z}_n} \mathbf{z}_n)/|\mathcal{Z}_n|$ to the server;
      Server aggregates the received importance measure: $\mathbf{z} \leftarrow \sum_{n=1}^{N} p_n \overline{\mathbf{z}}_n$;
      Reconfigure using Algorithm 2: $\mathbf{w}'(k+1), \mathbf{m}(k+1) \leftarrow \text{reconfigure}(\mathbf{w}(k+1), \mathbf{z})$;
      Reset: $\mathcal{Z}_n \leftarrow \emptyset, \forall n$;
    else
      No reconfiguration: $\mathbf{w}'(k+1) \leftarrow \mathbf{w}(k+1)$;
    Server sends new parameters to each client: $\mathbf{w}'_n(k+1) \leftarrow \mathbf{w}'(k+1), \forall n$;

Figure: Adaptive pruning algorithm [10]

Solving-Formulations

To solve the final optimization problem:

$$\max_{\mathcal{A} \subseteq \mathcal{P}} (\mathcal{A} \cup \bar{\mathcal{P}}) \tag{2}$$

Where $\mathcal{A}$ is the set of parameters in $\mathcal{P}$ that remain (i.e., are not pruned). The final set of all parameters in the original model. The final set of remaining parameters is then $\mathcal{M} = \mathcal{A} \cup \bar{P}$.

Algorithm 2

Input : importance measure g_j^2 and time coefficient t_j, for each parameter index j

Output: the optimal subset of parameters $\mathcal{A}$

$\mathcal{A} \leftarrow \emptyset$;

$\mathcal{S} \leftarrow \arg\text{sort}_{j \in \mathcal{P}} \frac{g_j^2}{t_j}$; // ordered set

for $j \in \mathcal{S}$ **do**

  if $\frac{g_j^2}{t_j} \geq \Gamma(\mathcal{A} \cup \overline{\mathcal{P}})$ **then**

    $\mathcal{A} \leftarrow \mathcal{A} \cup \{j\}$;

  else

    break;

return $\mathcal{A}$; // final result

Figure: Algorithm to solve the problem of optimization [10]

Assumptions

1. Smoothness:

$$\|\nabla F_n(w_1) - \nabla F_n(w_2)\| \leq \beta \|w_1 - w_2\|, \forall n, w_1, w_2$$

   where β is a positive constant.
2. Lipschitzness:

$$\|F(w_1) - F(w_2)\| \leq L \|w_1 - w_2\|, \forall w_1, w_2$$

   where L is a positive constant.
3. Unbiasedness:

$$\mathbb{E}[g_n(w)] = \nabla F_n(w) \forall n, w$$

Implementation

Using Sparse Matrixes

- In the literature, most existing implementations substitute sparse parameters by applying binary masks to dense parameters. However, this strategy increases the overhead of computation.
- For the implementation of PruneFL, dense matrices for full-sized models and sparse matrices for weights in both convolutional and fully connected layers in pruned models are used.

Implementation

Complexity Analysis

- **Storage, Memory and Communication** The implementation uses bitmap and value index tuple storage to store the sparse matrix. Bitmap adds one bit to denote zero. Incurs in $\frac{1}{32}$ extra storage and communication overhead.
- **Computation:** The multiplication of dense matrices is highly optimized. The computational complexity of the matrix multiplication between a sparse matrix S and a dense matrix D $O(n)$, where n is associated with the number of nonzero entries in S with the assumption that D is fixed.
- **Implementation Challenges** Sparse matrix computing is constrained in popular machine learning frameworks.

Implementation

Dense Matrix

1	2	31	2	9	7	34	22	11	5
11	92	4	3	2	2	3	3	2	1
3	9	13	8	21	17	4	2	1	4
8	32	1	2	34	18	7	78	10	7
9	22	3	9	8	71	12	22	17	3
13	21	21	9	2	47	1	81	21	9
21	12	53	12	91	24	81	8	91	2
61	8	33	82	19	87	16	3	1	55
54	4	78	24	18	11	4	2	99	5
13	22	32	42	9	15	9	22	1	21

Sparse Matrix

1	.	3	.	9	.	3	.	.	.
11	.	4	.	.	.	.	.	2	1
.	.	1	.	.	.	4	.	1	.
8	.	.	.	3	1	.	.	.	.
.	.	.	9	.	.	1	.	17	.
13	21	.	9	2	47	1	81	21	9
.	.	.	.	.	.	.	.	.	.
.	.	.	.	19	8	16	.	.	55
54	4	.	.	.	11	.	.	.	.
.	.	2	.	.	.	.	22	.	21

Figure: Dense vs Sparse matrix

Architecture

Architecture	Conv-2	VGG-11	ResNet-18	MobileNetV3-Small
Convolutional	32, pool, 64, pool	64, pool, 128, pool, 2×256, pool, 2×512, pool, 2×512, pool	64, pool, $2 \times [64, 64]$, $2 \times [128, 128]$, $2 \times [256, 256]$, $2 \times [512, 512]$	16, 16, 8, 16, 16, 72, 72, 24, 88, 88, 24, 96, 96, 24, 96, 40, 240, 240, 64, 240, 40, 240, 240, 64, 240, 40, 120, 120, 32, 120, 48, 144, 144, 40, 144, 48, 288, 288, 72, 288, 96, 576, 576, 144, 576, 96, 576, 576, 144, 576, 96, 576
Fully-connected	2048, 62 (input: 3136)	512, 512, 10 (input: 512)	avgpool, 100 (input: 512)	avgpool, 1280, dropout (0.2), 2 (input: 960)
Conv/FC/all params	52.1K/6.6M/6.6M	9.2M/530.4K/9.8M	11.2M/102.6K/11.3M	927.0K/592.9K/1.5M

Figure: Model Architecture

Architecture

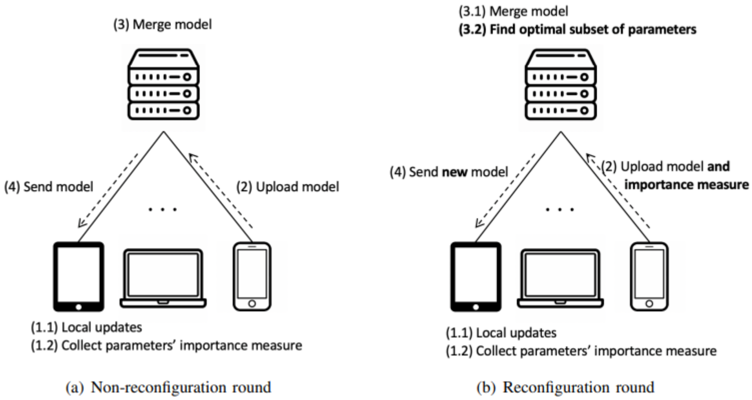


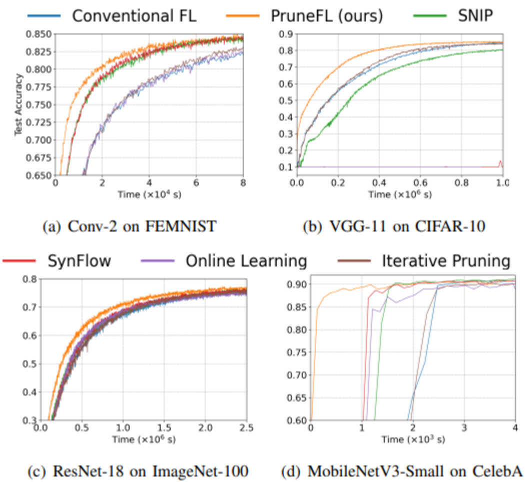
Figure: Example of adaptive pruning performed in conjunction with further pruning carried out during FL.

Experimental Setup

The evaluation of PruneFL was done with four image classification tasks:

1. Conv-2 model on FEMNIST.
2. VGG-11 model on CIFAR-10.
3. ResNet-18 model on ImageNet-100.
4. MobileNetV3-Small model on CelebA.

Test Accuracy

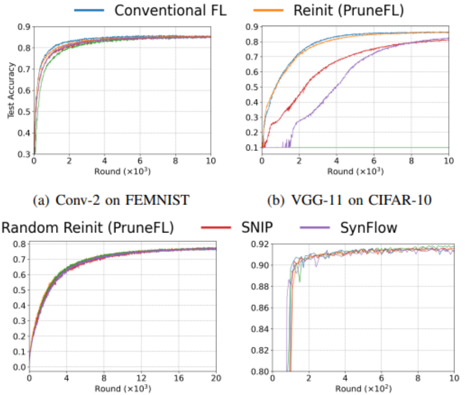


(a) Conv-2 on FEMNIST (b) VGG-11 on CIFAR-10

(c) ResNet-18 on ImageNet-100 (d) MobileNetV3-Small on CelebA

Figure: Test accuracy vs. time results of four datasets.

Lottery Ticket



(a) Conv-2 on FEMNIST (b) VGG-11 on CIFAR-10

(c) ResNet-18 on ImageNet-100 (d) MobileNetV3-Small on CelebA

Figure: Lottery ticket results of four datasets. (a) Conv-2 on FEMNIST. (b) VGG-11 on CIFAR-10. (c) ResNet-18 on ImageNet-100. (d) MobileNetV3-Small on CelebA.

Results PruneFL

Approach	Time (FLOPs) to reach 70% accuracy	Time (FLOPs) to reach 80% accuracy
Conventional FL	17,929 s (3.5 TFLOPs)	52,153 s (10.5 TFLOPs)
PruneFL (ours)	**3,187 s (1.6 TFLOPs)**	**15,009 s (6.8 TFLOPs)**
SNIP	6,801 s (3.3 TFLOPs)	22,467 s (11.7 TFLOPs)
SynFlow	7,132 s (3.6 TFLOPs)	22,327 s (12.3 TFLOPs)
Online	18,042 s (3.5 TFLOPs)	54,593 s (10.7 TFLOPs)
Iterative	17,495 s (3.5 TFLOPs)	46,521 s (10.1 TFLOPs)

Figure: TIME AND ACCUMULATED FLOPS PER CLIENT TO REACH TARGET ACCURACY (FEMINIST)

Results PruneFL

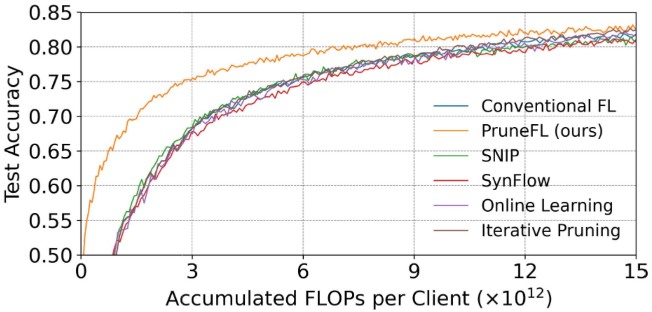


Figure: Test accuracy versus accumulated FLOPs per client (FEMNIST).

Conclusions

- PruneFl contribute with two main contributions:
  - Distributed Pruning
  - Adaptive Pruning
- The authors have proposed PruneFL for FL in edge/mobile computing contexts to minimize neural network size so resource-limited clients may train them quickly.
- PruneFL's several pruning steps increase performance.
- The low-complexity adaptive pruning strategy for efficient FL in PruneFL figures out the model's size that can match the prediction accuracy of the original model in less time.
- Raspberry Pi trials show that FL lottery tickets are found.
- Quantization can be used with the approach to decrease communication overhead.

Bibliography I

"Aman's AI Journal • CS230 • Deep Learning Intro and Applications," https://aman.ai/cs230/dl-intro-apps/, [Online; accessed 2022-10-15].

R. Jony, R. I. Rony, M. Rahman, and A. Rahat, "Big data characteristics, value chain and challenges," 05 2016.

G. P. Koudouridis, Q. He, and G. Dán, "An architecture and performance evaluation framework for artificial intelligence solutions in beyond 5g radio access networks," *EURASIP Journal on Wireless Communications and Networking*, vol. 2022, no. 1, sep 24 2022.

M. Kraus, S. Feuerriegel, and A. Oztekin, "Deep learning in business analytics and operations research: Models, applications and managerial implications," *European Journal of Operational Research*, vol. 281, 09 2019.

"A developer-friendly guide to model pruning in PyTorch," https://spell.ml/blog/model-pruning-in-pytorch-X9pXQRAAACIAcH9h, [Online; accessed 2022-10].

Bibliography II

L. Chen, Y. Chen, J. Xi, and X. Le, "Knowledge from the original network: restore a better pruned network with knowledge distillation," *Complex & Intelligent Systems*, vol. 8, no. 2, pp. 709–718, jan 10 2021.

G. P. Koudouridis, Q. He, and G. Dán, "An architecture and performance evaluation framework for artificial intelligence solutions in beyond 5g radio access networks," *EURASIP Journal on Wireless Communications and Networking*, vol. 2022, no. 1, sep 24 2022.

J.-W. Liu, J.-J. Zhou, M. S. Kamel, and X.-L. Luo, "Online Learning Algorithm Based on Adaptive Control Theory," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 6, pp. 2278–2293, 6 2018.

"Finite-time Analysis of the Multiarmed Bandit Problem."

Bibliography III

Y. Jiang, S. Wang, V. Valls, B. J. Ko, W.-H. Lee, K. K. Leung, and L. Tassiulas, "Model Pruning Enables Efficient Federated Learning on Edge Devices," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–13, 2022.

Model Pruning Enables Efficient Federated Learning on Edge Devices

A Brief Summary

Rómulo Condori

Institute of Computing

r204185@dac.unicamp.br