Nature Computational Science ,Volume 1, March 2021, pp. 221–228
DOI: 10.1038/s43588-021-00039-6

Random Sketch Learning for Deep Neural Networks in Edge Computing

Bin Li[1,2✉], Peijun Chen[1], Hongfu Liu[1], Weisi Guo[3,4], Xianbin Cao[5✉], Junzhao Du[6], Chenglin Zhao[1], Jun Zhang[2,5]

***Abstract*—Despite the great potential of deep neural networks (DNN), they require massive weights and huge computational resources, creating a vast gap when deploying artificial intelligence (AI) at low-cost edge devices. Current lightweight DNN, achieved by high-dimensional space pre-training and post-compression, presents challenges when covering the resources deficit, making tiny AI hard to be implemented. Here, we report an architecture named random sketch learning, or Rosler, for computational-efficient tiny AI. We build a universal compressing-while-training framework, which, for the first time, learns directly a compact model and, most importantly, enables computational efficient on-device learning. As validated on different models and datasets, it attains substantial memory reduction of 50∼90× (16-bits quantization), compared to full-connected DNN. We demonstrate it on low-cost hardware, whereby the computation is accelerated by >180× and the energy consumption is saved by ∼10×. Our method paves the way for deploying tiny AI in many scientific and industrial applications.**

I. Introduction

Deep learning is a powerful tool for solving complex problems [1], whereby the analytical models are not sufficiently representative to describe real world complexities [2, 3]. Deploying deep neural network (DNN) on edge devices that are remote from a center server is critical for many scientific/industrial applications [4, 5, 6], e.g. remote observing [7, 8], autonomous instruments [9, 10], and mission critical diagnostics [11, 12]. In such tiny artificial intelligence (AI), a large number of distributed devices (e.g. edge analytic, smart sensors) have limited power budget (e.g. tens of milliwatt) and storage size (∼100 KB on-chip memory) [13, 14, 15, 16, 17]; whilst the cloud server would be inaccessible [4]. Thus, the radical incompatibility between the computation-intensive DNN (e.g. >2 MB weights, >400 mW power even for a moderate DNN in handwriting recognition [18]) and the restricted memory/energy resources presents a substantial challenge [14, 19, 20]. In this respect, there exists a fundamental gap in energy efficiency and on-chip memory (related also to the complexity) when loading AI to low-cost hardware.

1. School of Information and Communication Engineering, Beijing University of Posts and Telecommunications, Beijing, 100876, China.
2. School of Information and Electronics, Beijing Institute of Technology, Beijing, 100081, China.
3. The Alan Turing Institute, 96 Euston Road, London NW1 2DB, UK.
4. Centre for Autonomous and Cyberphysical Systems, Cranfield University, MK43 0AL, UK.
5. School of Electronic and Information Engineering, Beihang University, Beijing, 100191, China.
6. The 6th Research Institute of China Electronics Corporation, Beijing, 102209, China.
Corresponding authors: Bin Li (Binli@bupt.edu.cn) and Xianbin Cao (xbcao@buaa.edu.cn).

To tackle this challenge, one potential way is to combine co-efforts from hardware design (e.g. near-data processing [21], non-von Neumann architectures [22, 23]) and algorithm development (e.g. new compression methods [20]). As reported [24], the Moore's law-based hardware scaling was largely blocked, owing to the movement of large weights between central processing unit (CPU) and off-chip memory [19]. Thus, the computation capacity and energy efficiency of low-cost devices cannot be rapidly improved to keep up with the explosive increase of DNN. Recently, lightweight deep learning has received great attention [20, 19], aiming to reduce a large model via: (1) network pruning [25, 26, 27, 28, 29], (2) low-rank approximation (LRA) [30, 31, 32, 33], (3) weight quantization [34, 27], and (4) network architecture transform (NAT) [35, 36]. Parallel to the hardware advances [20], such algorithmic innovations boost the widespread use of DNN.

But still and all, current lightweight DNNs are far from computational-efficient for tiny AI. First, almost all methods follow a classical framework, i.e. high-dimensional space pre-training + post-compression (Figure 1-a, 1-d) [32, 33, 25, 26, 27]. Even if a heavy network was reduced, the effective compression ratio (or the memory reduction ratio) is severely restricted by a pre-trained model [32, 33, 37], which may be inadequate to cover the deficit in hardware resources. So, how to break the current limit in model compression, thus maximally ease the storage/computation burden, remains one open question. Second, the computational pre-training/fine-turning is challenging to low-cost hardware [33, 28]. Thus, how to directly find a compact network, e.g. beyond the bondage of pre-training, is another fundamental unsolved problem, which is critical to the on-device federated learning in many privacy/latency sensitive scenarios [38, 39, 40].

Here, we report an architecture named random sketch learning, or Rosler, for hardware-friendly tiny AI. We build a universal compressing-while-training framework, which, for the first time, learns directly the tiny representation by removing computational pre-training/fine-tuning. To achieve this, we develop an approximate rank-restricted back-propagation (*aResBP*) algorithm. The attainable compression ratio is extended substantially by Rosler (Figure 3), after its depth is stretched via a butterfly-unfolding (BUFF) structure, which represents each dense layer of DNN with three cascading layers. Each cascading layer thus corresponds to a small weight matrix (referred also as sketch), see Figure 1-e.

The learned tiny model has the lower rank and higher equivalent sparsity, compared to one pretrained large DNN. As tested on different models/datasets, it substantially reduces

the storage of model weights by $25 \sim 45\times$ (float-point; Figure 2-e, 2-f, 3-f and 3-h), which suffices to cover the hardware deficit. We demonstrate our method for machinery diagnosis on low-cost hardware. The computation is accelerated by $> 180\times$ (16 bits fixed-point; $\sim 50\times$ for float-point), and the consumed energy is saved by $> 10\times$; whilst its accuracy is only degraded by $\sim 1\%$. Most importantly, it allows for the computational-efficient on-device training that was a challenging task, thus enabling the privacy-sensitive and low-latency federated learning[41]. As such, our method makes tiny AI available to the resource-constrained platforms.

II. Results

A. Random Sketch Learning

In principle, Rosler seeks for one tiny representation of each layer in DNN, by identifying multiple small sketches. Here, a sketch refers to one random sampling version of the weight $\mathbf{W} \in \mathbb{R}^{M\times N}$; M and N are the input and output sizes of the layer. For example, a column sampling sketch is $\mathbf{C} = \mathbf{W}(:, \mathcal{S}_c) \in \mathbb{R}^{M\times s}$, whilst a row sampling sketch is $\mathbf{R} = \mathbf{W}(\mathcal{S}_r, :) \in \mathbb{R}^{s\times N}$; $\mathcal{S}_c$ and $\mathcal{S}_r$ are two indexing sets of the sampled columns and rows. For clarity, we assume $|\mathcal{S}_c| = |\mathcal{S}_r| = s$, with $s \ll \min\{M, N\}$. Then, each large weight is represented by one BUFF structure of 3 sub-layers – a left-sketch $\mathbf{C}$, a central-body $\mathbf{U} \in \mathbb{R}^{s\times s}$ and a right-sketch $\mathbf{R}$, i.e. we approximate the full-connected (FC) layer with $\mathbf{W} \simeq \mathbf{CUR}$ (Figure 1-a, 1-e).

We thus directly learn the tiny stretched model (Figure 1-f), by removing computational pre-training and post-compression. Although it was difficult for classical back-propagation (BP) method[30], this can be achieved by our sketch learning algorithm (Figure 2-b, 2-c, see details in the Method section A), which constitutes a universal *approximate rank-restricted BP (aRes-BP)*. That means, the rank of a BUFF structure is restricted in each training iteration, i.e. $\text{rank}(\mathbf{CUR}) \leq s$. A distinctive feature of aRes-BP is that three small sketches are firstly updated as a whole, and then trained one by one (Figure 2-c). The convergence of this random sketch learning is also demonstrated numerically (Figure 2-d and 2-e).

B. Application to Multi-layer Perceptron

We start from the learning of tiny sketched model for multi-layer perceptron (MLP). Here, we consider a machinery diagnose problem [12], i.e. using the recorded time series to predict five operation states of industrial bearing. The public data of Case Western Reserve University is used [12]; each input contains 500 randomly truncated data samples from the time series. In the benchmark FC DNN, the number of nodes of input layer is 500; the number of nodes of 2 hidden layers are 300 and 100; the numbers of nodes of output layer is 5; 3000 samples for training and 2000 for test, resulting in a test accuracy 0.996. We evaluate various sparse pruning methods, e.g. weight pruning [25], single-shot pruning (SNIP) [29] and lottery ticket hypophysis (LTH) [28]; as well as low-rank compression methods [32, 33], see Figure 3-a. For a slightly degraded accuracy 0.99, the compression ratio of Rosler is 0.022 (the learned tiny model has 8 layers; the number of nodes of input layer is 500; the numbers of nodes of 6 hidden layers are 4, 4, 300, 3, 3, 200; the number of nodes of output layer is 5). Similar results are attained in the MINIST data for handwritten numeral recognition (Figure 3-c, 3-d). In the FC network, the numbers of nodes of input layer, 2 hidden layers and output layers are 784, 512, 256 and 10, respectively.

With the compressed model, one immediate result is that the on-chip memory of edge device will be reduced. Compared to the FC DNN, a classical pruning method saves the memory by $\sim 3.4\times$ (Figure 3-b, float-point; see the Method section D). The LTH method [28] reduces the storage by $\sim 6.5\times$ (its memory size is $3\times$ of the number of non-zero weights in order to record the row/column indexes of sparse elements; one-shot mode was used to balance the training complexity, see the Method section D). Another SNIP method [29] reduces the memory size by $\sim 16.7\times$ (attaining a compression ratio 0.02). In comparison, our method reduces the on-chip storage by $> 40\times$. Incorporating a novel BUFF structure, Rosler allows to store the model weights ($\sim$ 20 KB) in on-chip random-access memory (RAM), which is more efficient for data movement. As reported [20], the accessing of large dynamical RAM (DRAM) consumes orders of magnitude higher energy than small on-chip RAM of a few kilobytes.

Accompanied by our tiny model, the computational complexity is reduced and edge inference would be substantially accelerated, see details in the Method section B. When inferring the bearing states at edge devices, the required computation is reduced by $\sim 38.5\times$ (Figure 3-b). Focusing on the sparse structures, both SNIP and LTH may incur the much higher computation cost. For example, without the hardware-inefficient sparse matrix computation, Rosler would be faster than SNIP by $> 4\times$ (Supplementary Figure 1), even if they acquire the same compression ratio (Figure 3-b and 3-d).

Most importantly, by removing the computational pre-training with a huge memory/power burden, the complexity of Rolser in edge training is also reduced, see details in the Method section B. Taking the machinery diagnose problem for example, the time complexity of on-device training would be reduced by $>20\times$, compared to the FC DNN. Although the popular SNIP method also simplifies the computational pre-training [29], its sparse structure is incompatible to efficient hardware storage/computation. Thus, our method makes edge learning at low-cost devices computationally efficient, when learning in the place where the data was observed.

C. Application to Convolution Neural Network

For another convolution neural network (CNN), the kernel weights have both sparse and low-rank properties [33]. Fortunately, our method enables a unified sketch learning, no matter what the underlying deep learning model is, e.g. MLP or CNN. It reconciles two different aspects of deep representation. For one thing, the learned BUFF model is low ranked; and for another, the equivalent weight $\tilde{\mathbf{W}} = \mathbf{CUR}$ would be sparse.

We examine the learned tiny model with both convolution and FC layers (the aRes-BP algorithm is the same as for MLP, when computing convolution via matrix multiplication; Supplementary Figure 2-a). We consider the CNN for MINIST,

with 3 convolution layers (kernel 3×3, max-pooling) and 1 FC layers (the numbers of channels in 3 convolution layers are 32, 64 and 128; while the number of nodes of output layer is 10). When a test accuracy is 0.98 (the benchmark accuracy of classical CNN is 0.99), the attained compression ratio of Rosler is 0.039 (Figure 3-e), with regards to a classical CNN. Accordingly, it reduces the memory size by $\sim 16.4\times$, and meanwhile accelerates the computation by $\sim 18.7\times$ (Figure 3-f). As such, the model weights can be also stored in the on-chip RAM (Supplementary Figure 2-b).

We further evaluate our method in large CNN, i.e. VGG-A model (8 convolution layers + 3 FC layers) [42]. Since we focus on tiny AI on edge devices (whereby the size of feature maps would be largely limited), we consider the popular CIFAR-10 dataset for the image recognition of 10 objects [29, 43, 28]. From Figure 3-g, the compact tiny model learned by our method again achieves a low compression ratio of 0.0387, while achieving a slightly degraded accuracy 0.858 (a classical CNN has an accuracy 0.868; the learned feature maps are illustrated in Supplementary Figure 3). Although the sparsity-based approach, e.g. SNIP [29], attains a comparable compression ratio, Rosler is more efficient in both storage and computation (Figure 3-h).

In addition to a compression of model weights, Rosler also enables the reduction of large feature maps that are ineffective for data movement (Supplementary Figure 2-c, 2-d; see the Method section C). We evaluate the CNN on another Cat-Dog dataset for image recognition of 2 objects (cat and dog); 4 convolution layers (3×3 kernel) with 3 FC layers (the numbers of channels in 4 convolution layers are 32, 64, 128 and 128; while the numbers of nodes of 3 FC layers are 1024, 512 and 1). As shown, the accessing of off-chip DRAM can be reduced by $4.3\times$ (Supplementary Table 1). Unlike classical pruning methods focusing only on model weights, Rosler is capable of reducing the time/space complexity of both model weights and feature maps in CNN.

D. Application in On-device Federated Learning

In many scenarios of data analytics, e.g. remote monitoring [4, 8], internet of things (IoT) for intelligent sensing [9, 11] and digital twin [44], the centralized machine learning will be barely feasible [38]. First, the local user data would become privacy sensitive, especially for medial and industrial data [40, 45, 39]. Second, it needs to respond to real-time events in many latency-critical applications [4, 14, 44]. Third, the communication cost of massive raw data is expensive, or even impractical [41]. To address such problems, federated learning at local devices whereby data was generated presents a promising new way [38].

Our method can be directly applied to computational and communication efficient on-device training (e.g. at low-cost hardware). We compare Rosler with FC DNN and SNIP in the context of federated learning for industrial IoT [44]; the bearing data is used [12]; 4 layer FC DNN (the numbers of nodes of input layer, 2 hidden layers and output layer are 500, 300, 100 and 5); the number of local clients is 3; the number of local training epochs in each round is 5. By removing the computational pre-training, our method enables the lightweight deep learning at local devices, thus forwarding a trained model ($\mathbf{C}^t$, $\mathbf{R}^t$) rather than massive data to a central entity (Figure 4-a, 4-b). Meanwhile, it significantly reduces the model weights ($\sim 37\times$, Figure 4-e), and alleviates the clients-server communication cost (Figure 4-d and 4-e). Most importantly, in contrast to existing pruning methods, Rosler enables the cooperatively *parallel* update of each local model via two processors/nodes (Figure 4-c), further reducing the computation by $2\times$. I.e. two nodes update small sketches $\mathbf{C}^t$ and $\mathbf{R}^t$ respectively, by exchanging the latest results ($\mathbf{C}^{t-1}$ and $\mathbf{R}^{t-1}$) via proximate communication. As seen, Rosler reduces the computation complexity of on-device training by $\sim7\times$ (Supplementary Figure 1), and the total communication cost by $\sim2\times$, even compared to the state-of-the-art SNIP method. For SNIP, although its communication cost of each round is comparable to our method (Figure 4-f), the total epochs of model aggregation is $\sim2\times$ of Rosler (Figure 4-d). Similar results are attained on a large VGG model (Supplementary Table 2, CIFAR-10 dataset).

E. Hardware Demonstration of Edge Inference & Learning

We implement our method on low-cost digital signal processor (DSP) platform – CPU 375 MHz, on-chip RAM 256 KB, off-chip DRAM 256 MB (Figure 5-f). Enabled by the computational-efficient on-device learning, our tiny AI would excite the widespread interest in scientific/industrial applications. Here, we consider again the machinery diagnose problem – it represents a family of low latency computing tasks in industrial IoT [8, 11, 12].

The experiment setting of edge inference for industrial machinery diagnosis is illustrated in Figure 5-a. In Rosler, its network parameters ($\sim$20 KB, compression ratio ~ 0.023) are stored in on-chip memory, which can be efficiently accessed by the multiplication-and-accumulation (MAC) unit. While for the dense DNN, its large weights ($\sim$1 MB) can be only put in off-chip DRAM. As found, Rosler accelerates the hardware inference by $\sim50\times$ (Figure 5-b, float-point). If further combined with fixed-point computation, the hardware inference of Rosler would be accelerated by $>180\times$ (16 bits quantization, Figure 5-b; the program is stored in on-chip RAM). In this case, the average latency in analyzing the sensor data (500 samples) is around 300 μs; whilst a dense DNN requires around 60 ms which would be inadequate for many industrial applications emphasizing the real-time response ($<$1ms in the low-latency remote control).

The energy consumption of Rosler is then evaluated. From Figure 5-c, the full-load instantaneous power of Rosler and FC network are comparable, P= 426.9 mW (see details in the Method section E). Even so, the full-load time of Rosler is around $\sim 1/50$ of FC network (float-point; T_{Rosler}=1.205 ms and T_{FC}=61.92 ms; K=1). From Figure 5-d, the averaged power of Rosler is reduced by $\sim 9.6\times$ (float-point); an interrupt sleep mode is used and the stand-by power is 36.3 mW. Thus, the averaged power consumption of edge device is greatly reduced in Rosler (~ 40 mW).

Finally, we examine on-device training at low-cost hardware, as in the emerging federated learning [38, 39, 41]. For

Rosler, when a batch size is relatively small ($K < 20$), the input/output of each layer $(\mathbf{X}_l, \mathbf{Y}_l)$, the network weights and the program are all stored in on-chip RAM (Figure 5-e). For the pre-training of FC network, however the weights and K input samples can be only put into off-chip DRAM. We find the training latency of Rosler is reduced by $> 20\times$ (float-point; $K = 10$), even compared to a pre-training of dense FC network. Meanwhile, the consumed energy is reduced by $\sim 8\times$ (Figure 5-f). If further taking the computational fine-tuning (of sparse models) into account, the whole latency would be shortened by $> 30\times$. When a batch size is relatively large (K>100), the input/output of each layer and the program may be moved to off-chip DRAM; in this case the training latency is still greatly reduced (Supplementary Figure 4, bearing and MINIST data).

III. Discussion

We report a computational-efficient deep learning framework for resource-constrained data analytic, which directly learns a compact model without complex pre-training. Compared to state-of-the-art pruning methods, e.g. SNIP [29] and LTH [28], it attains a comparable compression ratio on various models/datasets, yet subject to a greatly reduced training complexity. Different from most current methods emphasizing sparse structure, our compact tiny model involves only dense matrices, which is hence more efficient for hardware storage and computation. Some recent methods, e.g. PruneTrain [43], obtain also dense weights (e.g. by invoking Lasso regularization); however they call for computational pretraining and can be hardly applied to on-device learning on low-cost hardware.

The success of our method is attributed that, for one thing, the designed aRes-BP would learn in another high-dimensional space with a low rank constraint (Figure 2-c Top); and for another, it actually implements the network architecture transform (NAT), by unfolding an original L-layer fat DNN into another $3L$-layer thin DNN. Despite the recently great interest on NAT [35, 36], an iterative search of network structures incurs the extremely high computation. Our method, in contrast, constitutes another way for implementing the computational efficient NAT, by adapting the depth and width in a systematic manner.

At the current stage, one limitation of our method is that the setting of the sampling lengths in different layers is less flexible, compared to sparsity-based pruning methods (e.g. given the compression ratio). To overcome this, a promising solution is to combine the sketch learning with a dynamic search of the sampling lengths, as in NAT. For example, starting from one compact tiny model, the network width can be further refined iteratively. Another open question is that, although our method directly learns the compact tiny model, whether it reaches the limit of deep model compression remains unknown, which may deserve further assessment.

Author contributions

B. Li conceived the idea. B. Li, P. J. Chen and H. F. Liu designed and implemented the source code. B. Li, P. J. Chen, H. F. Liu, W. S. Guo and X. B. Cao analyzed the data. All the authors together interpreted the findings and wrote the paper. P. J. Chen and H. F. Liu are contributed equally.

Acknowledgement

This work was supported by the Major Scientific Instrument Development Plan of National Natural Science Foundation of China (NSFC) under Grant No. 61827901, NSFC under Grant No. U1805262, Major Research Plan of NSFC under Grant No. 91738301, and Project of Basic Science Center of NSFC under Grant No. 62088101.

Reference

[1] Yann Lecun, Yoshua Bengio, and Geoffrey E Hinton. "Deep learning". In: *Nature* 521.7553 (2015), pp. 436–444.

[2] David Silver et al. "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529.7587 (2016), pp. 484–489.

[3] Markus Reichstein et al. "Deep learning and process understanding for data-driven Earth system science". In: *Nature* 566.7743 (2019), pp. 195–204.

[4] Park Jihong et al. "Wireless network intelligence at the edge". In: *Proceedings of the IEEE* 107.11 (2019), pp. 2204–2239.

[5] Doyu Hiroshi and Morabito Roberto. *TinyML as-a-Service: What is it and what does it mean for the IoT Edge?* https://www.ericsson.com/en/blog/2019/12/tinyml-as-a-service-iot-edge.

[6] O Vaughan. "Working on the edge". In: *Nature Electronics* 2 (2019), pp. 2–3.

[7] Benjamin Burger et al. "A Mobile Robotic Chemist". In: *Nature* 583 (2020), pp. 237–241.

[8] Jinjiang Wang et al. "Deep learning for smart manufacturing: Methods and applications". In: *Journal of Manufacturing Systems* 48.C (2018), pp. 144–156.

[9] Frederik J. Simons et al. "On the potential of recording earthquakes for global seismic tomography by low-cost autonomous instruments in the oceans". In: *Journal of Geophysical Research: Solid Earth* 114.B5 (2009), pp. –.

[10] B Ravi Kiran et al. "Deep Reinforcement Learning for Autonomous Driving: A Survey". In: *arXiv* (2020), pp. 1–18.

[11] B A Weiss et al. "Measurement Science Roadmap for Prognostics and Health Management for Smart Manufacturing Systems". In: *National Institute of Standards and Technology* (2016, http://dx.doi.org/10.6028/NIST.AMS.100-2).

[12] Wade A Smith and R B Randall. "Rolling element bearing diagnostics using the Case Western Reserve University data: A benchmark study". In: *Mechanical Systems and Signal Processing* 64 (2015), pp. 100–131.

[13] D. Hiroshi, M. Roberto, and Jan Höller. "Bringing Machine Learning to the Deepest IoT Edge with TinyML as-a-Service". In: *IEEE Internet of Things (IoT) Newsletter* (May, 2020).

[14] Doyu Hiroshi and Morabito Roberto. *TinyML as a Service and the challenges of machine learning at the edge*. https://www.ericsson.com/en/blog/2019/12/tinyml-as-a-service.

[15] Sally Ward-Foxton. *Adapting the Microcontroller for AI in the Endpoint*. https://www.eetimes.com/adapting-the-microcontroller-for-ai-in-the-endpoint/.

[16] Mike Loukides. *TinyML: The challenges and opportunities of low-power ML applications*. https://www.oreilly.com/radar/tinyml-the-challenges-and-opportunities-of-low-power-ml-applications/.

[17] Vijay Janapa Reddi. "Enabling Ultra-low Power Machine Learning at the Edge". In: *Presented in tinyML Summit 2020* (February 12-13, 2020).

[18] Gregor Koehler. *MNIST Handwritten Digit Recognition in Keras*. https://nextjournal.com/gkoehler/digit-recognition-with-keras. 2020.

[19] Xiaowei Xu et al. "Scaling for edge inference of deep neural networks". In: *Nature Electronics* 1.4 (2018), pp. 216–222.

[20] Vivienne Sze et al. "Efficient Processing of Deep Neural Networks: A Tutorial and Survey". In: *Proceedings of the IEEE* 105.12 (2017), pp. 2295–2329.

[21] Mingyu Gao et al. "TETRIS: Scalable and Efficient Neural Network Acceleration with $3D$ Memory". In: *in Proc. of the 22 International Conference on Architectural Support for Programming Languages and Operating Systems* 45.1 (2017), pp. 751–764.

[22] Can Li et al. "Analogue signal and image processing with large memristor crossbars". In: *Nature Electronics* 1.4 (2018), pp. 52–59.

[23] Mirko Prezioso et al. "Training and operation of an integrated neuromorphic network based on metal-oxide memristors". In: *Nature* 521.7550 (2015), pp. 61–64.

[24] "NVIDIA TESLA P100 (NVIDIA, 2017)". In: *www.nvidia.com/object/tesla-p100.html* ().

[25] Song Han et al. "Learning both weights and connections for efficient neural networks". In: *in Prof. of Neural Information Processing Systems (NIPS)* (2015), pp. 1135–1143.

[26] Wei Wen et al. "Learning Structured Sparsity in Deep Neural Networks". In: *in Prof. of Neural Information Processing Systems (NIPS)* (2016), pp. 2074–2082.

[27] Song Han, Huizi Mao, and William J Dally. "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding". In: *in Prof. of International Conference on Learning Representations (ICLR)* (2015).

[28] Jonathan Frankle and Michael Carbin. "The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks". In: *in Prof. of International Conference on Learning Representations (ICLR)* (2018).

[29] Namhoon Lee, Ajanthan Thalaiyasingam, and Philip HS Torr. "SNIP: Single-shot network pruning based on connection sensitivity". In: *in Prof. of International Conference on Learning Representations (ICLR)* (2019).

[30] Misha Denil et al. "Predicting Parameters in Deep Learning". In: *in Prof. of Neural Information Processing Systems (NIPS)* (2013), pp. 2148–2156.

[31] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. "Speeding up Convolutional Neural Networks with Low Rank Expansions". In: *arXiv: Computer Vision and Pattern Recognition* (2014).

[32] Tianyi Zhou and Dacheng Tao. "GoDec: Randomized Low-rank & Sparse Matrix Decomposition in Noisy Case". In: *in Prof. of International Conference on Machine Learning (ICML)* (2011), pp. 33–40.

[33] Xiyu Yu et al. "On Compressing Deep Models by Low Rank and Sparse Decomposition". In: *in Prof. of International Conference on Computer Vision and Pattern Recognition (CVPR)* (2017), pp. 67–76.

[34] Edward H Lee et al. "LogNet: Energy-efficient neural networks using logarithmic computation". In: (2017), pp. 5900–5904.

[35] Xuanyi Dong and Yi Yang. "Network pruning via transformable architecture search." In: *in Prof. of Neural Information Processing Systems (NIPS)* (2019), pp. 760–771.

[36] Yong Guo et al. "NAT: Neural architecture transformer for accurate and compact architectures". In: *in Prof. of Neural Information Processing Systems (NIPS)* (2019), pp. 737–748.

[37] Davis W Blalock et al. "What is the State of Neural Network Pruning". In: *arXiv: Learning* (2020).

[38] Q. Yang et al. "Federated machine learning: Concept and applications". In: *ACM Transactions on Intelligent Systems and Technology* 10.2 (2019), pp. 1–19.

[39] K. Bonawitz et al. "Practical Secure Aggregation for Federated Learning on User-Held Data". In: *in Prof. of Neural Information Processing Systems (NIPS)* (2016).

[40] Santiago Silva et al. "Federated Learning in Distributed Medical Databases: Meta-Analysis of Large-Scale Subcortical Brain Data". In: *Proc. of IEEE International Symposium on Biomedical Imaging*. 2019.

[41] H. Brendan Mcmahan et al. "Communication-Efficient Learning of Deep Networks from Decentralized Data". In: *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS)*. 2017, pp. 1–11.

[42] K. Simonyan and A. Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *International Conference on Learning Representations (ICLR)*. 2015.

[43] Sangkug Lym et al. "PruneTrain: fast neural network training by dynamic sparse model reconfiguration". In: *In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (2019), pp. 1–13.

[44] Yunlong Lu et al. "Low-latency Federated Learning and Blockchain for Edge Association in Digital Twin empowered 6G Networks". In: *IEEE Transactions on Industrial Informatics* 10.1109/TII.2020.3017668 (2020), pp. 1–10.

[45] Theodora. S. Brisimi et al. "Federated learning of predictive models from federated Electronic Health Records". In: *International Journal of Medical Informatics* 112 (2018), pp. 59–67.

Fig. 1. **Rosler directly learns one compact tiny model**. In current lightweight DNN, a full-connection network **(a)** is pre-trained firstly. **(b)** This pre-trained model is low-ranked (x-axis is the index number of singular values, y-axis is the amplitude of singular values), **(c)** and/or sparse (x-axis is the amplitude of pretrained weights, and y-axis gives their histogram). Model compression (e.g. network pruning), then attains a reduced model **(d)**, which is further iteratively fine-tuned. In Rosler, a larger layer is represented by a BUFF structure **(e)**, consisting of 3 sub-layers – a left flank $\mathbf{C}$, a central body $\mathbf{U}$ and a right flank $\mathbf{R}$. After a direct training, a compact tiny network is obtained **(f)**, by removing computational pre-training/post-compression.

Fig. 2. **Computational-efficient model training**. In contrast to a classical BP **(a)**, our method first updates three cascading layers as a whole **(b)**; $\mathbf{W}^t$ and $\Delta\mathbf{W}^t$ are the weight and the gradient of a large layer in the t-th epoch; $\{\mathbf{C}^t, \mathbf{U}^t, \mathbf{R}^t\}$ are the sketched weights of 3 cascading sub-layer, $\Delta\mathbf{C}^t$ and $\Delta\mathbf{R}^t$ are their gradients; $g(\mathbf{C}^t, \mathbf{R}^t)$ is one nonlinear function to compute $\mathbf{U}^t$ (see the Method section A); α is a learning step. **(c)** Relying on the designed aRes-BP, the BUFF structure is learned via the parallel training (*Top*) and the successive (*Bottom*) updating. **(d)** Convergence of test accuracy of random sketch learning (compression ratio of 0.022 and 0.053); the bearing dataset; 50 independent trials. **(e)** Convergence of loss function using the same dataset.

Fig. 3. **Test accuracy and computation/storage cost of Rosler**. **(a)** Accuracy of MLP model on bearing data; **(b)** the gains in memory reduction and hardware acceleration (float-point), with regards to FC network. **(c)** Accuracy of MLP model on MINIST data; **(d)** the gains in memory reduction and hardware acceleration (float-point). **(e)** Accuracy of CNN model on MINIST data; **(f)** the gains in memory reduction and hardware acceleration (float-point), with regards to classical CNN. **(g)** Accuracy of VGG model on CIFAR-10 data; **(h)** the attained memory reduction and hardware acceleration (float-point). Sparse pruning methods (SNIP, LTH, weight pruning) and low-rank compression methods (SVD, low-rank + sparse) are used for comparison.

Fig. 4. **On-device federated learning in industry IoT**. On-device federated learning in industrial IoT **(a)**. Each device uploads the trained model to center entity, which aggregates multiple local models and distributes a global model to edge devices for the next round training **(b)**. In Rosler, a parallel training can be implemented **(c)**, whereby two processors cooperatively update multiple sketches of a local model. **(d)** Test accuracy of different epochs of model aggregation; the compression ratio of both SNIP and Rosler is 0.027. **(e)** Communication cost (normalized by the amount of data of FC DNN, i.e. 824.23 KB; for SNIP the row/column indexes of sparse weights are not delivered). **(f)** Communication & computation reduction of Rosler (float-point).

Fig. 5. **Hardware demonstration of computational-efficient edge inference/training. (a)** Experimental setting of machinery health diagnosis on low-cost embedded platform. **(b)** Latency of the edged inference. *Green triangles*: the program is stored in off-chip DRAM; *Yellow circles*: the program is stored in on-chip RAM. **(c)** Full-load time and instantaneous power of FC network and Rosler, see details in the Method section E. The averaged power of edge inference **(d)**-*Top*, and the test accuracy **(d)**-*Bottom*. **(e)** Illustration of weights storage on DSP C6478 in edge training. **(f)** Latency and averaged power in edge training.

Method

A. Random Sketch Learning

The proposed sketch learning algorithm involves the following three stages.

1. **Initialization** At time $t = 0$, we initialize the weight of the l-th layer, $\tilde{\mathbf{W}}_l^0 \in \mathbb{R}^{M\times N}$, with the Xavier method [46]. Its low rank representation, denoted as $\mathbf{W}_l^0$, is further attained, i.e. $\text{rank}(\mathbf{W}_l^0) = s$ (s is one user-specific parameter related to the compression ratio). This is achieved by applying the singular value decomposition (SVD) on it, $\tilde{\mathbf{W}}_l^0 = \tilde{\mathbf{U}}_W\tilde{\mathbf{\Sigma}}_W\tilde{\mathbf{V}}_W^T$.

In order to construct a compact BUFF structure, we abstract two sketches $\mathbf{C}_l^0 \in \mathbb{R}^{M\times s_c}$ and $\mathbf{R}_l^0 \in \mathbb{R}^{s_r\times N}$ from $\mathbf{W}_l^0$, respectively by means of random column and row sampling,

$$\mathbf{C}_l^0 = \mathbf{W}_l^0(:, \mathcal{S}_c),\ \mathbf{R}_l^0 = \mathbf{W}_l^0(\mathcal{S}_r, :), \tag{1}$$

where $\mathcal{S}_c \subset \{1, 2, \cdots, N\}$ ($|\mathcal{S}_c| = s_c$) and $\mathcal{S}_r \subset \{1, 2, \cdots, M\}$ ($|\mathcal{S}_r| = s_r$) are two indexing sets, whereby random indexes are generated from two probability densities $\mathcal{P}_c = \{p_c(j)\}_{j=1}^N$ and $\mathcal{P}_r = \{p_r(i)\}_{i=1}^M$. For simplicity, we assume $s_r = s_c = s \ll \min\{M, N\}$ (in more general cases we have $s_r \neq s_c$). Here, a leverage score sampling is applied, i.e. the probabilities of selecting i and j are determined by

$$p_r(i) = ||\tilde{\mathbf{U}}_W(i,:)||_2^2/||\tilde{\mathbf{U}}_W||_F^2, \quad \text{for } i \in \mathcal{S}_r;$$
$$p_c(j) = ||\tilde{\mathbf{V}}_W(j,:)||_2^2/||\tilde{\mathbf{V}}_W||_F^2, \quad \text{for } j \in \mathcal{S}_c.$$

As a variation of leverage-score sampling, the indexes can be selected directly via the s largest leverage scores.

On this basis, we compute an initial central-body sketch $\mathbf{U}_l^0$, by minimizing the whole approximation error [47, 48], i.e.

$$\mathbf{U}_l^0 = \arg\min_{\mathbf{U}\in\mathbb{R}^{s\times s}} \left\|\mathbf{W}_l^0 - \mathbf{C}_l^0\mathbf{U}_l^0\mathbf{R}_l^0\right\|_F^2,$$

and one simple solution of this optimization problem is [47, 48, 49, 50]

$$\mathbf{U}_l^0 = \mathbf{W}_l^0(\mathcal{S}_r, \mathcal{S}_c)^\dagger, \tag{2}$$

where $\mathbf{X}^\dagger$ is the pseudo-inverse of $\mathbf{X}$; $||\cdot||_F^2$ is the F-norm. In the case $s_c = s_r = s$, we further have $\mathbf{U}_l^0 = \mathbf{W}_l^0(\mathcal{S}_r, \mathcal{S}_c)^{-1}$, and $\mathbf{X}^{-1}$ is the inverse of $\mathbf{X}$.

2. **Parallel training** In contrast to classical BP which updates the cascading sub-layers successively, i.e. $\mathbf{C}_{l-1}^t \to \mathbf{R}_l^t \to \mathbf{U}_l^t \to \mathbf{C}_l^t \to \mathbf{R}_{l+1}^t \to \cdots$, we tend to update $\mathbf{R}_l^t$ and $\mathbf{C}_l^t$ in a *parallel* manner (see Figure 2-b), respectively based on $\mathbf{R}_l^{(t-1)}$ and $\mathbf{C}_l^{(t-1)}$ of the previous time $(t-1)$, i.e.

$$\mathbf{C}_l^t = \mathbf{C}_l^{(t-1)} + \alpha\Delta\mathbf{C}_l^{(t-1)}, \tag{3}$$
$$\mathbf{R}_l^t = \mathbf{R}_l^{(t-1)} + \alpha\Delta\mathbf{R}_l^{(t-1)}, \tag{4}$$

where α denotes the learning step, which can be adaptively tuned, for example, via the Adam optimizer [51]. In the above, two weight gradients are calculated via:

$$\Delta\mathbf{C}_l^{(t-1)} = \mathbf{X}_l^T({\boldsymbol{\delta}'_{l+1}}^{(t-1)}\mathbf{R}_l^{(t-1)^T}\mathbf{U}_l^{(t-1)^T}) - 1/2\cdot\mathbf{S}_r\mathbf{U}_l^{(t-1)^T}(\mathbf{C}_l^{(t-1)^T}\mathbf{X}_l^T{\boldsymbol{\delta}'_{l+1}}^{(t-1)}\mathbf{R}_l^{(t-1)^T})\mathbf{U}_l^{(t-1)^T}, \tag{5}$$

$$\Delta\mathbf{R}_l^{(t-1)} = \mathbf{U}_l^{(t-1)^T}\mathbf{C}_l^{(t-1)^T}\mathbf{X}_l^T{\boldsymbol{\delta}'_{l+1}}^{(t-1)} - 1/2\cdot\mathbf{U}_l^{(t-1)^T}(\mathbf{C}_l^{(t-1)^T}\mathbf{X}_l^T{\boldsymbol{\delta}'_{l+1}}^{(t-1)}\mathbf{R}_l^{(t-1)^T})\mathbf{U}_l^{(t-1)^T}\mathbf{S}_c^T, \tag{6}$$

where $\mathbf{X}_l \in \mathbb{R}^{K\times M}$ is the input matrix (with a batch size K);

$\mathbf{S}_c \in \mathbb{R}^{N\times s}$ and $\mathbf{S}_r \in \mathbb{R}^{M\times s}$ are two equivalent sampling matrices, i.e., $\mathbf{W}\mathbf{S}_c = \mathbf{W}(:,\mathcal{S}_c)$ and $\mathbf{S}_r^T\mathbf{W} = \mathbf{W}(\mathcal{S}_r,:)$; $\boldsymbol{\delta}_l^t$ is an error matrix of the l-th layer; ${\boldsymbol{\delta}_l'}^t$ is obtained by passing $\boldsymbol{\delta}_l^t$ through the derivative of nonlinear activation function. In the L-th output layer, the loss function is cross entropy. For the other case $s_r \neq s_c$, the computation of gradients may be more complex, which however can be determined automatically in Python platform [52]. Then, the other sketch is updated by

$$\mathbf{U}_l^t = 2 \times \left[\mathbf{C}_l^t(\mathcal{S}_r,:) + \mathbf{R}_l^t(:,\mathcal{S}_c)\right]^\dagger. \tag{7}$$

3. **Successive training** In this stage, a trained BUFF structure would be further updated, as one $3L$-layer network. That is to say, three sub-layers in a BUFF structure would be updated *one by one*, as in classical BP (Figure 2-c).

Note that, for the L-th layer an output size N_L is usually small (e.g., we have $N_L = 5$ in a DNN model for bearing data), and thus the matrix sketching was not used in the 2nd stage. However, a sparsity-based pruning can be applied to the L-th layer in this stage (see details in the Method section D).

B. Computational Complexity

We consider the time complexity in the inference process. In the l-th layer ($l = 1,\cdots,L$), each input $\mathbf{x}_l \in \mathbb{R}^{1\times M_l}$ successively passes 3 sub-layers of a BUFF model. Rather than $\mathbf{y}_l = f(\mathbf{x}_l\mathbf{W}_l + \mathbf{B})$ (Figure 1-a), the output is computed by $\mathbf{y}_l = f(\mathbf{x}_l\mathbf{C}_l\mathbf{U}_l\mathbf{R}_l + \mathbf{b})$, where $f(\cdot)$ is nonlinear activation (i.e. the Rule activation is used) and $\mathbf{b}_l \in \mathbb{R}^{1\times N_l}$ is the bias vector. When measured by the number of multiplications, the time complexity of Rosler is $\mathcal{O}(KM_ls + N_ls^2 + KN_ls)$ ($s \ll \min\{M_l, N_l\}$) in the l-th layer, which is significantly lower than FC DNN with a complexity $\mathcal{O}(KM_lN_l)$.

For the training process, the time complexity comes mainly from the calculation of $\Delta\mathbf{C}_l^{(t-1)}$ and $\Delta\mathbf{R}_l^{(t-1)}$. Accordingly, the complexity of Rosler is $\mathcal{O}\{n_E[KM_ls + KN_ls + (M_l+N_l+K)s^2 + s^3]\}$; whilst for the FC network its training complexity is $\mathcal{O}(n_EKM_lN_l)$ (n_E is the number of training epochs).

C. Random Sketch Learning of CNN

1) Approximation of input feature: Given an input feature matrix $\mathbf{X}_l \in \mathbb{R}^{M\times N}$ (e.g. after unfolding a tensor to a matrix), two column/row sketches are obtained, i.e. $\mathbf{X}_{c,l} = \mathbf{X}_l\mathbf{S}_c = \mathbf{X}_l(:,\mathcal{S}_c) \in \mathbb{R}^{M\times s_c}$ and $\mathbf{X}_{r,l} = \mathbf{S}_r^T\mathbf{X}_l = \mathbf{X}_l(\mathcal{S}_r,:) \in \mathbb{R}^{s_r\times N}$. Then, the other sketch $\mathbf{X}_{u,l}$ is computed via $\mathbf{X}_{u,l} = (\mathbf{S}_r^T\mathbf{X}_l\mathbf{S}_c)^\dagger = \mathbf{X}_l(\mathcal{S}_r,\mathcal{S}_c)^\dagger$. Note that, when abstracting the column sketch $\mathbf{X}_{c,l}$, we ample s_c columns of $\mathbf{X}_l$ according to the uniform distribution. When determining the row sketch $\mathbf{X}_{r,l}$, the probability of sampling s_r rows is proportional to the row norm of $\mathbf{X}_{c,l}$, i.e. $p_r(i) \propto ||\mathbf{X}_{c,l}(i,:)||_2^2$ for $i \in \mathcal{S}_r$. Thus, we have $\mathbf{X}_l \simeq \mathbf{X}_{c,l}\mathbf{X}_{u,l}\mathbf{X}_{r,l}$.

2) Sketch learning of CNN with approximated feature maps: When the input feature and model weight are both approximated by multiple small sketches, the output of convolution layer is $\mathbf{y}_l = f(\mathbf{X}_{c,l}\mathbf{X}_{u,l}\mathbf{X}_{r,l}\mathbf{C}_l\mathbf{U}_l\mathbf{R}_l + \mathbf{b}_l)$. On this basis, the training of sketched CNN is similar to that of MLP. For example, the gradients ($\Delta\mathbf{C}_l$ and $\Delta\mathbf{R}_l$) of two sketched weights (i.e. $\mathbf{C}_l$ and $\mathbf{R}_l$) are obtained, and then $\mathbf{U}_l$ is updated. In the case of $s_c \neq s_r$, such two gradients can be computed automatically in the Python platform, based on a directed acyclic graph (DAG) of the matrix operations.

However, since an input feature map now has been replaced by multiple sketches, the propagation of the error matrix $\boldsymbol{\delta}_l$, which is related to the input matrix, would be different. For clarity, in the following analysis we denote $\tilde{\mathbf{W}}_l = \mathbf{C}_l\mathbf{U}_l\mathbf{R}_l$. After updating three sketches $\{\mathbf{C}_l, \mathbf{U}_l, \mathbf{R}_l\}$, then the error matrix $\boldsymbol{\delta}_l$ will be computed before it is propagated to the previous layer, i.e. $\boldsymbol{\delta}_l = \boldsymbol{\delta}_{c,l} + \boldsymbol{\delta}_{u,l} + \boldsymbol{\delta}_{r,l}$, whereby the three components are computed via:

$$\begin{aligned}
\boldsymbol{\delta}_{c,l} &= \boldsymbol{\delta}_{l+1}'[\tilde{\mathbf{W}}_l^T\mathbf{X}_{r,l}^T\mathbf{X}_{u,l}^T]\mathbf{S}_c^T,\\
\boldsymbol{\delta}_{r,l} &= \mathbf{S}_r[\mathbf{X}_{u,l}^T\mathbf{X}_{c,l}^T\boldsymbol{\delta}_{l+1}'\tilde{\mathbf{W}}_l^T],\\
\boldsymbol{\delta}_{u,l} &= \mathbf{S}_r\Big\{[(\mathbf{I}_{s_r\times s_r} - \mathbf{X}_{v,l}\mathbf{X}_{u,l})\mathbf{G}^T\mathbf{X}_{u,l}\mathbf{X}_{u,l}^T] +\\
&\qquad \mathbf{X}_{u,l}^T[\mathbf{X}_{u,l}\mathbf{G}^T(\mathbf{I}_{s_c\times s_c} - \mathbf{X}_{u,l}\mathbf{X}_{v,l})] - \mathbf{X}_{u,l}^T\mathbf{G}\mathbf{X}_{u,l}^T\Big\}\mathbf{S}_c^T,
\end{aligned}$$

whereby $\mathbf{X}_{v,l} \triangleq \mathbf{S}_r^T\mathbf{X}_l\mathbf{S}_c$, $\mathbf{G} \triangleq \mathbf{X}_{c,l}^T\boldsymbol{\delta}_{l+1}'\tilde{\mathbf{W}}_l^T\mathbf{X}_{r,l}^T$; $\mathbf{I}_{s\times s}$ is the $s\times s$ identity matrix; $\boldsymbol{\delta}_{l+1}'$ is obtained by passing $\boldsymbol{\delta}_{l+1}$ through the derivative of nonlinear activation function.

D. Simulation Settings

Rosler: The initial learning rate is 1×10^{-3} in Adam in the 2nd stage, and 3×10^{-4} in the 3rd stage; a mini-batch size is 100. In MLP model, we assume $s_r = s_c$. For a given sampling length of the l-th layer (i.e. s_l), the overall compression ratio is computed via $\beta = \sum_{l=1}^L M_ls_l + s_l^2 + s_lN_l)/\sum_{l=1}^L M_lN_l$ (M_l and N_l are the input and output size of the l-th layer). For CNN model, s_r may differ from s_c; in principle we may have $s_c/s_r \sim \mathcal{O}(N/M)$). In the 3rd stage, the sparse pruning can be applied to the last output layer (e.g. with a compression ratio 0.15). For the large VGG-11 model (as well as federated learning), the last stage was removed for simplicity.

LTH: We adopt the one-shot mode [28], in order to balance the training complexity. We also evaluate the 4-shot LTH, which acquires the more promising performance (Supplementary Figure 5-c), yet at the cost of the largely increased computation. An initial learning rate is 1×10^{-3} in the stochastic gradient descent (SGD) process; the mini-batch size is 100. Following the standard setting [28], in the L-th layer the pruning ratio is $(1-\beta)/2$; β is the compression ratio of the other $(L-1)$ layers. For a large VGG-11 model (Figure 3-g and 3-h), the 1st convolution layer is uncompressed.

SNIP: The connectivity score is firstly evaluated by passing a small set of samples [29], with a mini-batch size of 20. Then, n weights (corresponding to the first n largest connectivity scores) are reserved, $n = \beta\sum_l^L M_lN_l$. The initial learning rate is 1×10^{-3} in SGD; a mini-batch size 100. When applied to federated learning, one local device attains an initial model, and reports it to a center entity which then directly broadcasts it to multiple local devices, as the global initializer.

Low-rank + sparse: For the GreBdec method [33], a learning rate in SGD is 1×10^{-3} and a mini-batch size is 100. For the large VGG-11 model, the compression ratio of low-rank and sparse components are equal [33]. The compression ratio of convolution layers is $5\times$ of FC layers (where only the

sparse weights are used). In MLP model, the ratio between low-rank and sparse components is 9, in order to improve the accuracy. To even things up, we exclude the iterative retraining which incurs a high computation [32, 33].

Pruning: For a classical pruning method, we directly remove small weights based on a pretrained model [25]. The same learning rate and mini-batch size are used. Meanwhile, we assume the computational fine-tuning was not applied.

E. Measurement of Consumed Energy

In the experiment, we use the low cost embedded platform – DSP C6478. The measured operation voltage of DSP core is $v = 1.0 \sim 1.2$ voltage. To determine the effective current, a resistance of $r = 6\ \Omega$ is cascaded to the input voltage (i.e. 5 voltage). Then, the operation current is calculated by $i = \Delta V/r$; and then the instantaneous power is $P_{\text{ins}} = v \times i$.

The full-load operation time of Rosler, T_{Rosler}, is measured via the Digital Phosphor Oscilloscope (DPO). On this basis, the consumed energy of Rosler in a time duration ΔT is:

$$E_{\text{rosler}} = P_{\text{ins}} T_{\text{Rosler}} + P_{\text{sleep}}(\Delta T - T_{\text{Rosler}}).$$

Here, the interrupt sleep mode is used in the remaining time $(\Delta T - T_{\text{Rosler}})$, whereby the stand-by power is P_{sleep} (for DSP C6478, $P_{\text{ins}} = 426$ mW and $P_{\text{sleep}} = 36.3$ mW). For FC network, we similarly have $E_{\text{FC}} = P_{\text{ins}} \Delta T$, and $\Delta T = T_{\text{FC}}$ is its full-load operation time (for the bearing diagnose task, the measured duration T_{FC} is 61.92 ms, see Figure 5-c).

Data Availability

The bearing data (https://csegroups.case.edu/bearingdatacenter), the MINIST data (http://yann.lecun.com/exdb/mnist/), the CIFAR-10 data (https://www.cs.toronto.edu/ kriz/cifar.html), and the Cat-dog data (https://www.kaggle.com/c/dogsvs-cats/data) can be all downloaded from the corresponding websites. Source Data for Figures 2-5 is also available with this manuscript.

Code Availability

A Python implementation of Rosler is available in Code Ocean [52].

[46] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2010, pp. 249–256.

[47] Shusen Wang and Zhihua Zhang. "Improving CUR matrix decomposition and the Nyström approximation via adaptive sampling". In: *Journal of Machine Learning Research* 14.1 (2013), pp. 2729–2769.

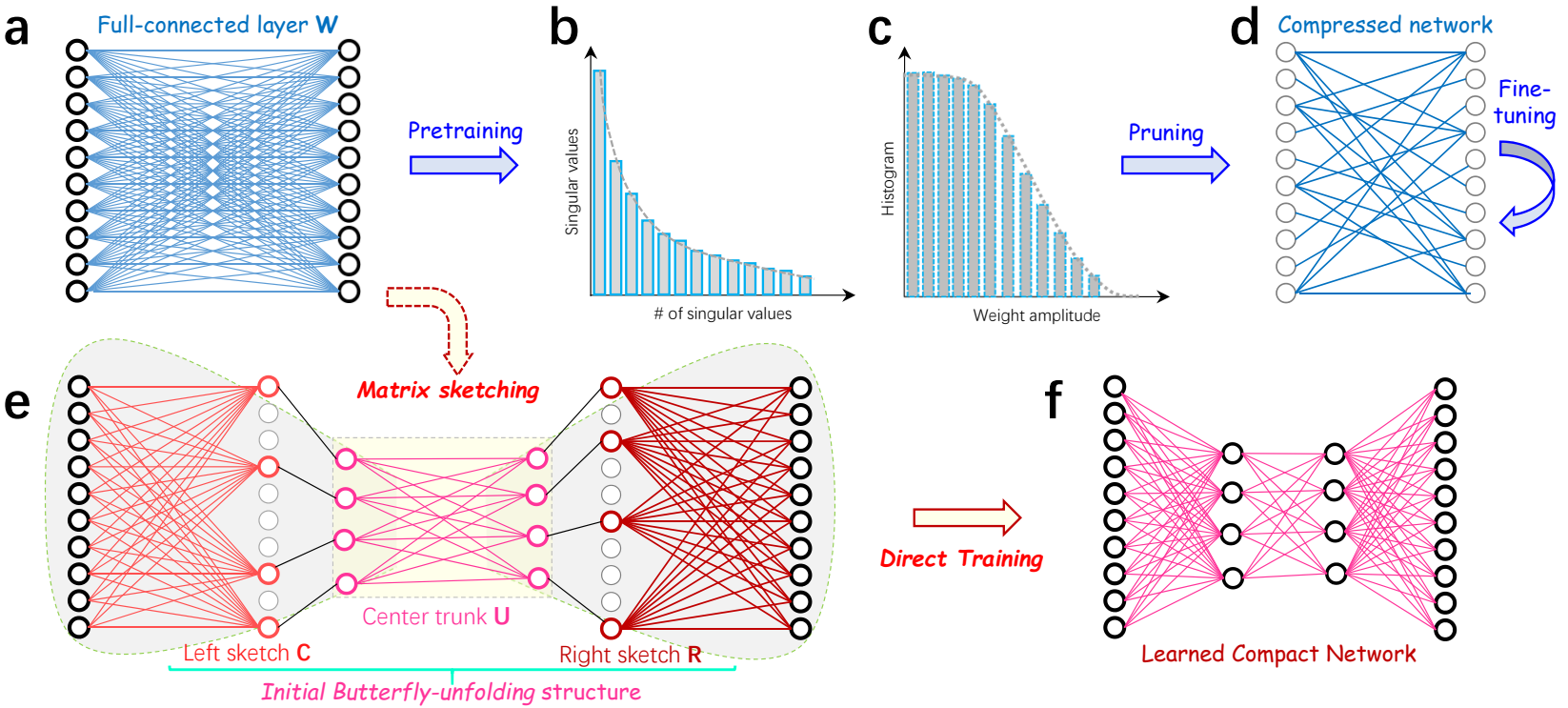
[48] Petros Drineas, Michael W. Mahoney, and S. Muthukrishnan. "Relative-Error CUR Matrix Decompositions". In: *SIAM Journal on Matrix Analysis and Applications* (2008).

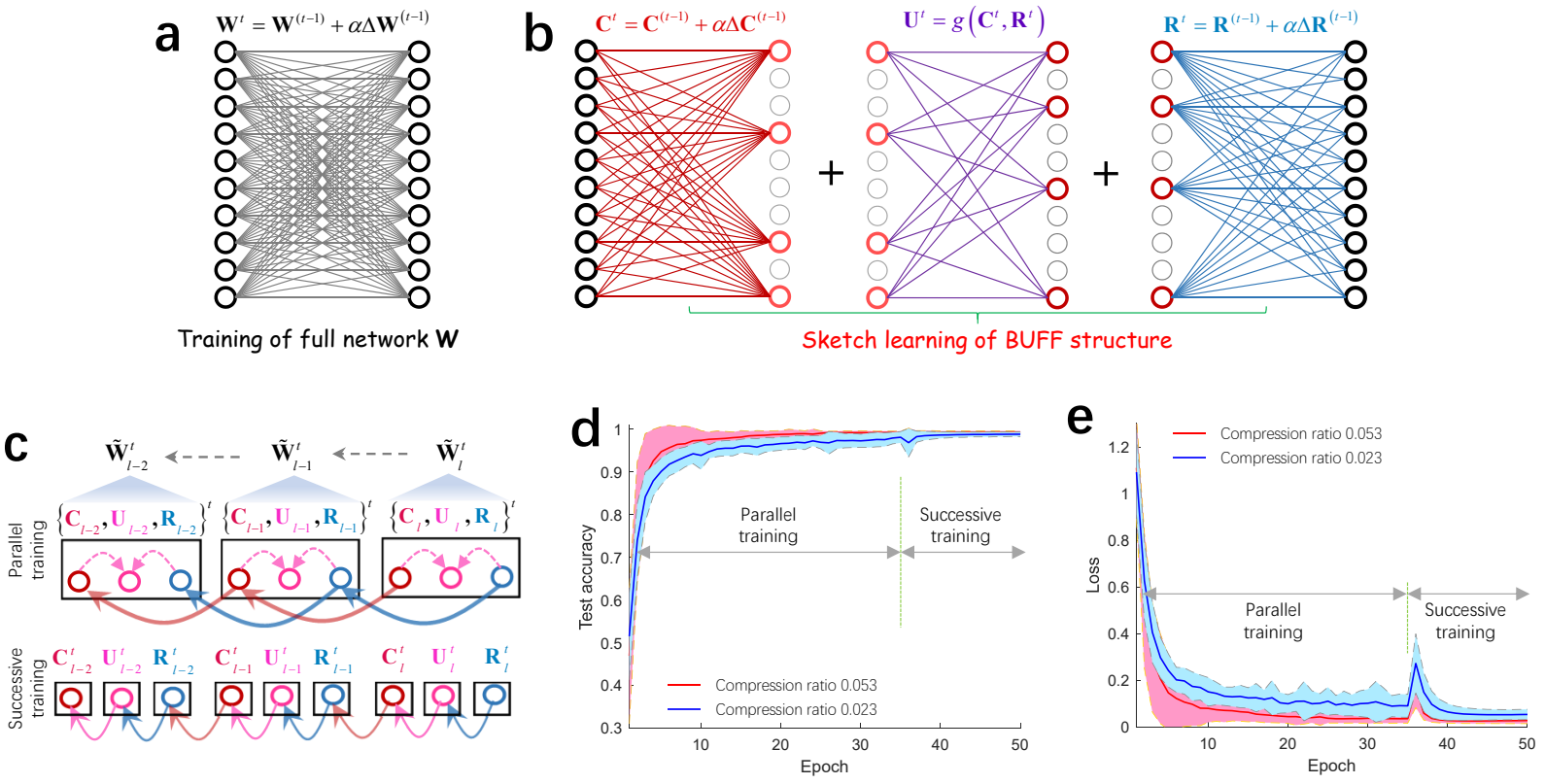
[49] Bin Li et al. "Randomized Approximate Channel Estimator in Massive-MIMO Communication". In: *IEEE Communications Letters* 24.10 (2020), pp. 2314–2318.

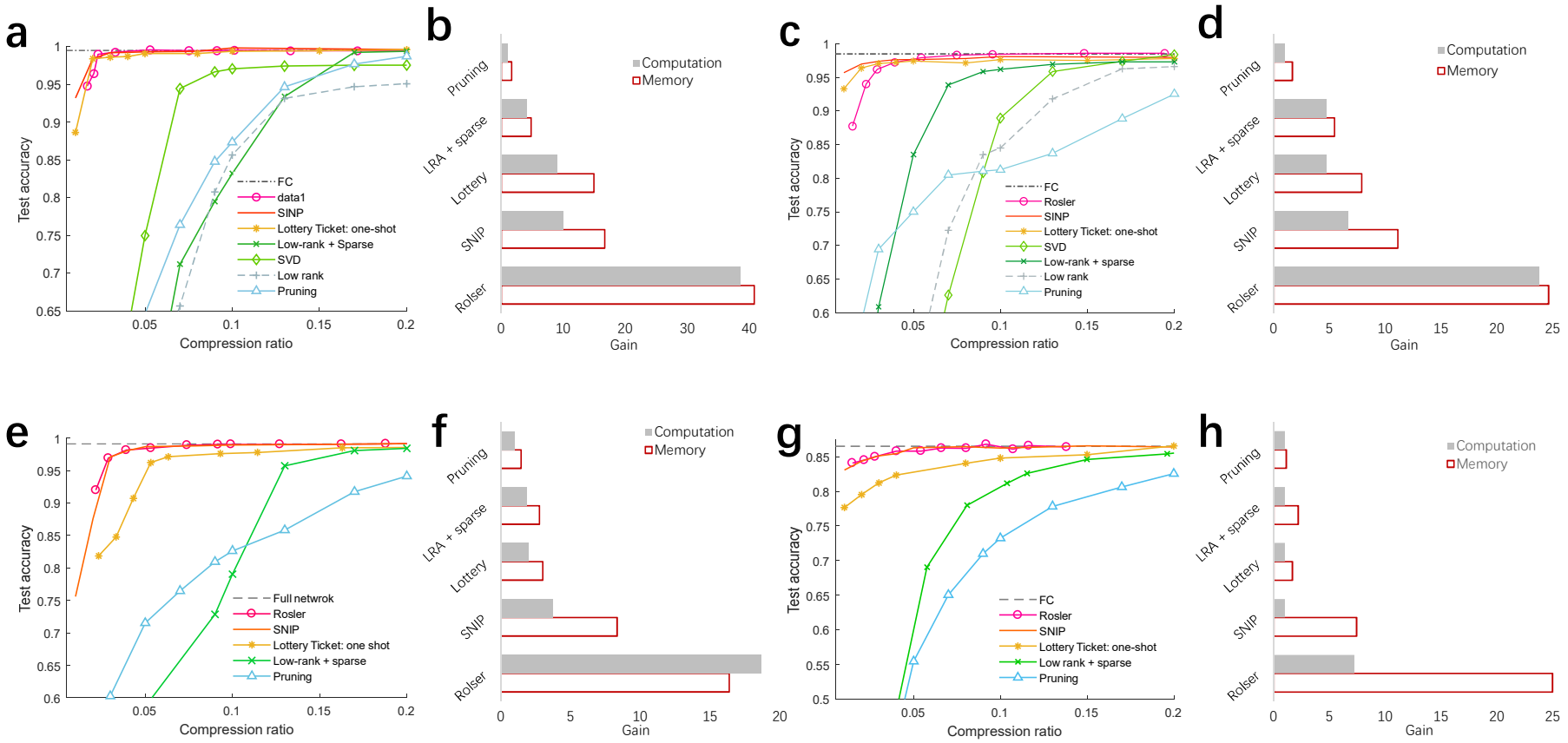
[50] Bin Li, Shuseng Wang, and et.al. "Fast-MUSIC for Automotive Massive-MIMO Radar". In: *ArXiv 1911.07434* (2019), pp. 1–14.

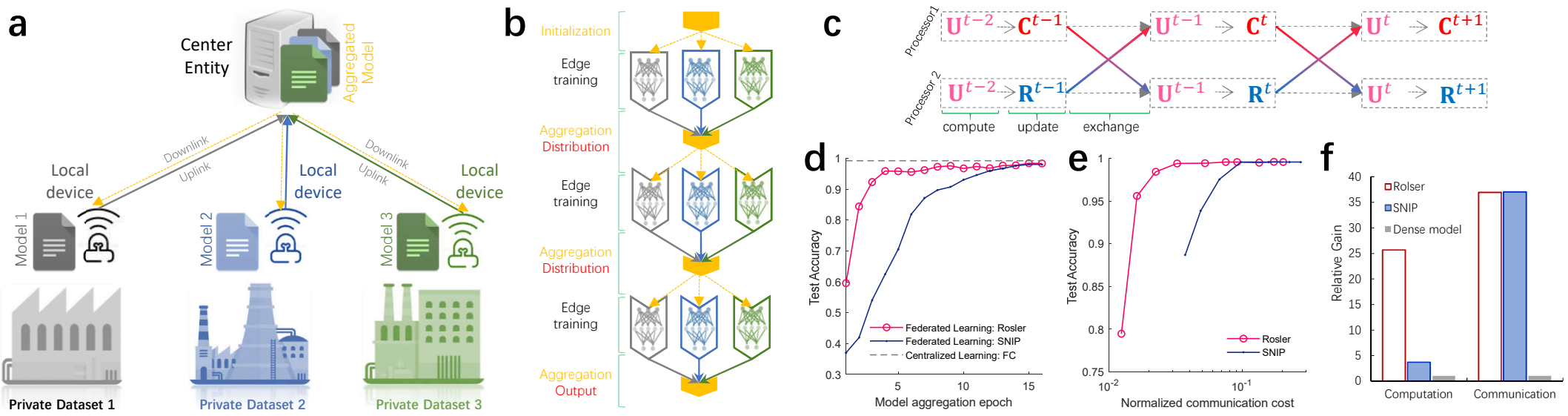
[51] Diederik P Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *in Prof. of International Conference on Learning Representations (ICLR)* (2015), pp. 1–15.

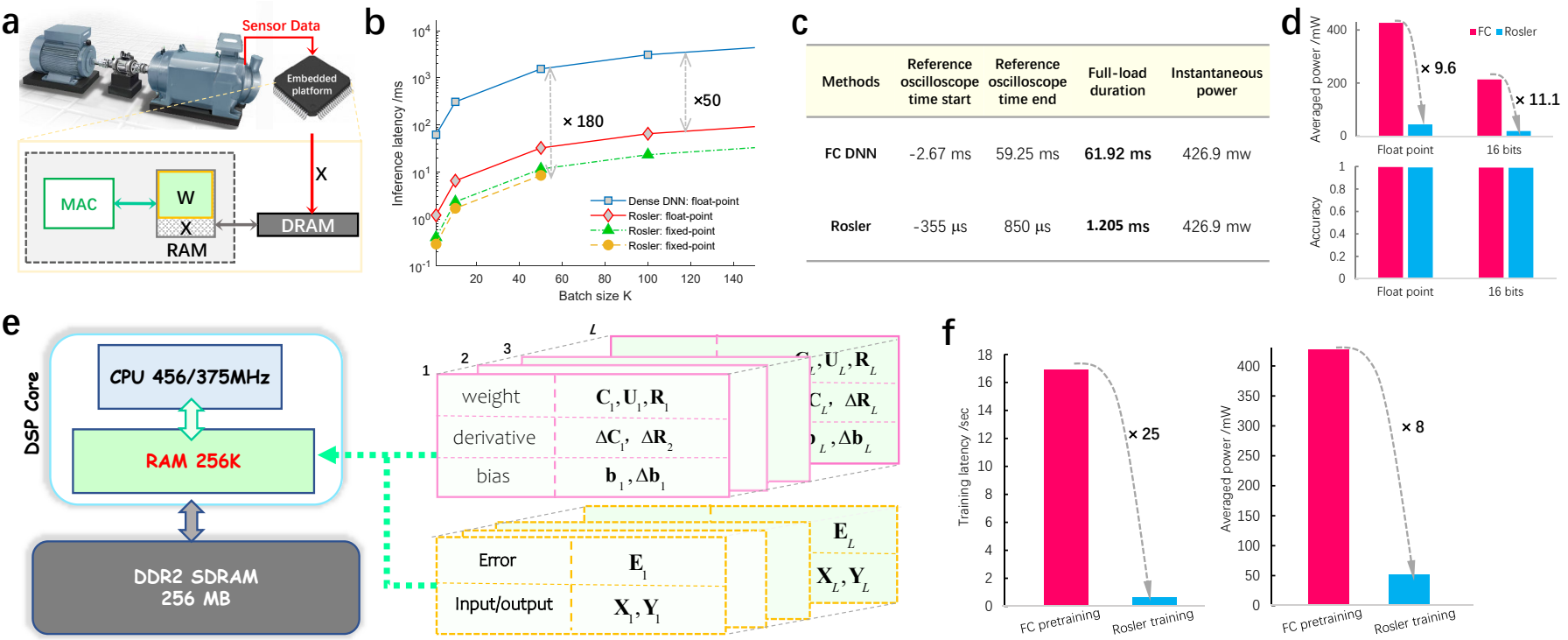
[52] Bin Li, Hongfu Liu, and Peijun Chen. "Random Sketch Learning for Tiny AI [Source Code]". In: *https://doi.org/10.24433/CO.5227764.v1* (2021).


a
Full-connected layer W
Pretraining
b
Singular values
# of singular values
c
Histogram
Weight amplitude
Pruning
d
Compressed network
Fine-tuning
e
Matrix sketching
Center trunk U
Left sketch C
Right sketch R
Initial Butterfly-unfolding structure
Direct Training
f
Learned Compact Network


a
$\mathbf{W}^t = \mathbf{W}^{(t-1)} + \alpha\Delta\mathbf{W}^{(t-1)}$
Training of full network W
b
$\mathbf{C}^t = \mathbf{C}^{(t-1)} + \alpha\Delta\mathbf{C}^{(t-1)}$
$\mathbf{U}^t = g\left(\mathbf{C}^t, \mathbf{R}^t\right)$
$\mathbf{R}^t = \mathbf{R}^{(t-1)} + \alpha\Delta\mathbf{R}^{(t-1)}$
Sketch learning of BUFF structure
c
Parallel training
Successive training
d
Test accuracy
Epoch
Parallel training
Successive training
Compression ratio 0.053
Compression ratio 0.023
e
Loss
Epoch
Parallel training
Successive training
Compression ratio 0.053
Compression ratio 0.023


a
Test accuracy
Compression ratio
FC
data1
SINP
Lottery Ticket: one-shot
Low-rank + Sparse
SVD
Low rank
Pruning
b
Computation
Memory
Pruning
LRA + sparse
Lottery
SNIP
Rolser
Gain
c
Test accuracy
Compression ratio
FC
Rosler
SINP
Lottery Ticket: one-shot
SVD
Low-rank + sparse
Low rank
Pruning
d
Computation
Memory
Pruning
LRA + sparse
Lottery
SNIP
Rolser
Gain
e
Test accuracy
Compression ratio
Full netwrok
Rosler
SNIP
Lottery Ticket: one shot
Low-rank + sparse
Pruning
f
Computation
Memory
Pruning
LRA + sparse
Lottery
SNIP
Rolser
Gain
g
Test accuracy
Compression ratio
FC
Rosler
SNIP
Lottery Ticket: one-shot
Low rank + sparse
Pruning
h
Computation
Memory
Pruning
LRA + sparse
Lottery
SNIP
Rolser
Gain


a
Center Entity
Aggregated Model
Downlink
Uplink
Local device
Model 1
Model 2
Model 3
Private Dataset 1
Private Dataset 2
Private Dataset 3
b
Initialization
Edge training
Aggregation Distribution
Edge training
Aggregation Distribution
Edge training
Aggregation Output
c
Processor1
Processor2
compute
update
exchange
d
Test Accuracy
Federated Learning: Rosler
Federated Learning: SNIP
Centralized Learning: FC
Model aggregation epoch
e
Test Accuracy
Rosler
SNIP
Normalized communication cost
f
Relative Gain
Rolser
SNIP
Dense model
Computation
Communication


a
Sensor Data
Embedded platform
MAC
W
X
RAM
X
DRAM
b
Inference latency /ms
Batch size K
× 180
×50
Dense DNN: float-point
Rosler: float-point
Rosler: fixed-point
Rosler: fixed-point
c
Methods
Reference oscilloscope time start
Reference oscilloscope time end
Full-load duration
Instantaneous power
FC DNN
-2.67 ms
59.25 ms
61.92 ms
426.9 mw
Rosler
-355 μs
850 μs
1.205 ms
426.9 mw
d
FC
Rosler
Averaged power /mW
× 9.6
× 11.1
Float point
16 bits
Accuracy
Float point
16 bits
e
DSP Core
CPU 456/375MHz
RAM 256K
DDR2 SDRAM 256 MB
weight
derivative
bias
Error
Input/output
f
Training latency /sec
× 25
FC pretraining
Rosler training
Averaged power /mW
× 8
FC pretraining
Rosler training