nature portfolio

https://doi.org/10.1038/s43588-021-00039-6

Supplementary information

Random sketch learning for deep neural networks in edge computing

In the format provided by the authors and unedited

Supplementary Information: 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]

I. Comparison Against Related Work

Recent works on the network pruning or model compression can be classified into three categories, i.e. (1) network pruning requiring a computational pre-training [1], [2]; (2) model pruning requiring a simple training – one-shot pre-training [3]; and (3) the methods based on *dynamical* network architecture searching, namely network architecture transform (NAT) [4]–[6]. To sum up, our method would differ sharply from such related works.

A. Lottery Ticket Hypothesis and PruneTrain

First, both lottery ticket hypothesis (LTH) and PruneTrain methods start from the computational training of high-dimensional dense network, and require multiple pretraining rounds. LTH aims to identifies a small sub-network with the sparse structure, which plays an essential rule and may achieve the comparable accuracy. PruneTrain relies on the grouped Lasso regularization to find another small sub-network, via the structured pruning [2]. Both them follows a popular

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. Centre for Autonomous and Cyberphysical Systems, Cranfield University, MK43 0AL, UK.

4. The Alan Turing Institute, 96 Euston Road, London NW1 2DB, 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.

lightweight deep learning framework, i.e. first training in a high-dimensional parameter space, and then extracting a small structure in low dimensional space.

We are particularly impressed by the attained compression ratio of the LTH method. By iteratively and gradually pruning a large network, LTH is capable of identifying a very small sub-network that can preserve the accuracy, after removing most of model weights. When this LTH scheme runs iteratively, without considering the time complexity and the storage size of multi-round pretraining/retaining, it would acquire the best performance. I.e., this LTH scheme would attain the even lower compression ratio, compared to our method.

However, we are more interested to the performance comparatione of LTH and Rosler, in specific application scenarios – *tiny AI* on low-cost edge devices. It is easily noted that, when it comes to on-device learning subject to the restricted hardware resources (as for the emerging federated learning), the LTH method would be infeasible due to its massive storage size and highly computational complexity. For one thing, as for most pruning methods, LTH starts from the training of large dense networks in a high-dimensional parameter space, thereby calling for an impractically huge storage to run it. For another, it repeats training-pruning for many times (e.g. 40 rounds), and hence incurs a very high computation burden. Taking a pruning rate of 0.1 for example (in each round LTH keeps 90% weight and discards 10% weights), the total time complexity equals to carry out $(1 - 0.9^{40})/(1 - 0.9) \simeq 10$ times pre-training rounds, which is formidably complex to low-cost hardware (e.g. in on-device federated learning).

To even things up, we compare LTH and Rosler in the resource-restricted tiny AI scenarios. We assume the LTH method adopts the *one-shot mode*, i.e. only performing pre-training and retraining in one round to balance the training complexity. That is to say, LTH directly removes $(1-\beta)\%$ model weights after the pre-training of large network (β is the target compression ratio). And then, the remained structure is initialized by the original weights and is further retrained. As demonstrated, in this case LTH acquires the less attractive performance compared to Rosler, even if it still involves the computational pretraining process (see Figure 3 in the main text). When the number of iteration was increased, LTH attains a lower compression ratio but is still inferior to our method (e.g. 4-rounds, Supplementary Figure 5).

Last but not least, LTH would obtain a *non-structured* sparse model, which tends to be ineffective for hardware implementation. For one thing, in order to record the positions (i.e. the row and column indexes) of each non-zero elements, a sparse weight matrix requires $3\times$ memory

size, compared to one dense matrix with the same number of non-zero elements. For another, since the general hardware has no efficient algorithm to support sparse matrix multiplication, the related time complexity is much higher (e.g. $5 \sim 10\times$) than that of dense matrices (with the same non-zero elements), see Supplementary Figure 1. As such, even the LTH method would achieve the same compression ratio (via multiple rounds with a higher computational complexity), our method would be more efficient for hardware storage and computation in tiny AI, since it learns an ultra-compact structured model (i.e. each small sketch in a BUFF structure represents one dense matrix).

B. SNIP

Second, the other SNIP method develops another way to obtain a small network. Unlike a classical lightweight framework "pre-training + post-compression", SNIP adopts the *one-shot pruning* before the network training, relying on a concept of connectivity score, which effectively simplifies the computational pretraining procedure. As reported, it attains the very promising performance, by learning the sparse representation with a largely reduced computation.

Although SNIP avoids the computational pre-training process, it starts from the exploration in a high-dimensional parameter space. Recall that, SNIP needs to compute all the connectively scores of the whole high-dimensional weights. Thus, despite a new initialization (via one-shot training), SNIP may still follow a widely-accepted lightweight deep learning framework (i.e. a high-dimensional parameter space would help to improve the likelihood of escaping from local optimums and containing the better solutions). From this theoretical perspective, our method for the first time breaks through such a classical DNN compression framework, which directly learns in a low-dimensional space to find a tiny sketched network, by entirely excluding the high-dimensional space exploration.

The second difference is that our method learns directly one ultra-compact *structured* network involving only dense matrices, whilst SNIP focuses on the *non-structured* sparse matrices. As demonstrated, SNIP can obtain the comparable compression ratio as our method, in both MLP and CNN (see Figure 3-a, 3-c, 3-e and 3-g). Even so, when it comes to the concerned tiny AI on edge devices, Rosler should be more efficient in both hardware storage and computation.

(1) In order to store the positions of non-zero sparse elements (e.g. the column and row indexes), SNIP requires the larger memory size of $> 3\times$, compared to our approach (see Figure

3-b, 3-d, 3-f and 3-h, Figure 4-e and 4-f).

(2) Since the general hardware do not support the highly efficient multiplication of sparse matrices, SNIP is more computational than Rosler (in both training and inference). Although we indeed observe its time complexity grows as the number of non-zeros elements, i.e. $\mathcal{O}\{\text{nnz}(\mathbf{W})\}$, it may consume $5 \sim 10\times$ computation (see also Figure 3-b, 3-d, 3-f and 3-h, Figure 4-e and 4-f; and Supplementary Figure 2).

(3) Moreover, SNIP is inherently data dependent – the one-shot pruning relies on the data-dependent connectivity score. In real-world applications, especially the emerging data-privacy federated learning with *non i.i.d* local datasets, each trained model at local device may potentially maintain *different* sparse structures, which would increase the network parameters of a global model (e.g. accompanying the communication cost and computation complexity). In contrast, our method is data independent.

C. NAT

Third, we would like to clarify that Rosler emphasizes a direct network compression in a systematical and `static` manner, while the current NAT focuses on the iterative architecture transform via the `dynamical` searching. As such, although the dynamical network architecture transform (NAT) methods would identify one small structure, they would be rarely applicable to tiny AI on low-cost edge devices, due to the extremely high computation of the iterative search [4]–[6].

One interesting observation is that our method may constitute a novel *non-iterative* way to achieve computational efficient NAT; recall that it transforms a L-layer large fat network to another $3L$-layer small thin network, by simultaneously stretching its depth and shrinking its width. So, one unique merit of such static NAT method is that it avoids a highly computational network searching, and is more suitable to tine AI on edge devices. Also, our Rosler can be readily combined with current NAT methods, e.g. by taking the hardware resource constraints [6] to develop more efficient models for tiny AI.

D. Randomized Low-rank Approximation

Our method develops one computational efficient sketch learning approach for lightweight DNN, by leveraging random matrix sketching technique. As we noted, random matrix sketching

has been studied in linear algebra [7]–[9] and signal processing [10], [11], which yet concentrates on static matrix compression rather than dynamical training before this work. As in most low-rank approximation (LRA) models [12]–[15], an intuitive way is thus to decompose a pre-trained weight $\mathbf{W}$ into this BUFF structure, by applying randomized matrix sketching [7]–[11] to current lightweight models. While the pretrained model may be reduced by this randomized LRA (RLRA), the compression ratio seems to be largely limited (Supplementary Figure 6); and more importantly, the computational pre-training still seriously hinders low-cost devices from the on-device learning, e.g. in privacy/latency critical federated learning at local devices whereby massive raw data was observed [1], [16], [17].

Thus, our method differs sharply from current LRA models. First, LRA/RLRA must rely on an *exact* and *high-dimensional* representation – a pre-trained model [12], [13] which fundamentally bounds the attainable compression region (Figure 3-a, 3-c, 3-e and 3-g). In Rosler, small sketches use only *approximate* and *low-dimensional* information, getting entirely off a pre-trained $\mathbf{W}$. Second, a direct training of multiple decomposed matrices was found impossible [15], evidencing a pre-trained model was a prerequisite to LRA [12], [13] or RLRA [7], [8]. As a core innovation, the sub-layers are directly trained via the designed aRes-BP, i.e. without a heavy pre-training, which transcends the limits of current DNN models in the computational-efficient tiny AI.

E. Summary

Based on the above detailed comparison, our method breaks through a classical lightweight deep learning framework, and for the first time demonstrates a new path which starts from a tiny deep network to train it directly in a low-dimensional space. That is to say, we exclude the exploration of a high-dimensional parameter space (as for LTH, PruneTrain, SNIP and NAT), although it was rarely succeeded before this work. As widely accepted, a high-dimensional parameter training was an important necessity, given the improved probability of escaping from local optimums and containing the better local optimums in high-dimensional space. Contrary to this common rule, Rosler now establishes a new lightweight deep learning framework. The success of our method is attributed that, despite a low-dimensional BUFF structure, it actually explores another high-dimensional space with a low-rank constraint, as done by the approximated rank-restricted back propagation (aRes-BP) algorithm. Most importantly, it directly learns a ultra-compact *structured* model (rather than *non-structured* sparse models) that is inherently hardware

efficient. As validated by comprehensive experiments (both computer simulations and hardware implementations), it significantly reduces the memory storage, the hardware computation (see Figure 3-b, 3-d, 3-f and 3-h, Figure 4-d to 4-f) as well as the energy consumption. In this regard, our method bridges the computational gap between DNN and low-cost devices, thus making tiny AI available to many industrial/medical applications.

REFERENCES

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

[2] S. Lym, C. Esha, Z. Siavash, W. Wen, S. Sujay, and E. Mattan, "PruneTrain: fast neural network training by dynamic sparse model reconfiguration," *In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 1–13, 2019.

[3] N. Lee, A. Thalaiyasingam, and P. H. Torr, "SNIP: Single-shot network pruning based on connection sensitivity," *in Prof. of International Conference on Learning Representations (ICLR)*, 2019.

[4] X. Dong and Y. Yang, "Network pruning via transformable architecture search.," *in Prof. of Neural Information Processing Systems (NIPS)*, pp. 760–771, 2019.

[5] Y. Guo, Z. Yin, M. Tan, Q. Chen, J. Chen, P. Zhao, and J. Huang, "NAT: Neural architecture transformer for accurate and compact architectures," *in Prof. of Neural Information Processing Systems (NIPS)*, pp. 737–748, 2019.

[6] Q. Lu, W. Jiang, X. Xu, Y. Shi, and J. Hu, "On neural architecture search for resource-constrained hardware platform," *arXiv preprint arXiv:1911.00105*, 2019.

[7] P. Drineas, M. W. Mahoney, and S. Muthukrishnan, "Relative-error CUR matrix decompositions," *SIAM Journal on Matrix Analysis and Applications*, 2008.

[8] S. Wang and Z. Zhang, "Improving CUR matrix decomposition and the nyström approximation via adaptive sampling," *Journal of Machine Learning Research*, vol. 14, no. 1, pp. 2729–2769, 2013.

[9] S. Wang, Z. Zhang, and T. Zhang, "Towards more efficient SPSD matrix approximation and CUR matrix decomposition," *Journal of Machine Learning Research*, vol. 17, no. 1, pp. 7329–7377, 2016.

[10] B. Li, S. Wang, and et.al., "Fast-MUSIC for Automotive Massive-MIMO Radar," *ArXiv 1911.07434*, pp. 1–14, 2019.

[11] B. Li, S. Wang, J. Zhang, X. Cao, J. Zhang, and C. Zhao, "Randomized Approximate Channel Estimator in Massive-MIMO Communication," *IEEE Communications Letters*, vol. 24, no. 10, pp. 2314–2318, 2020.

[12] X. Yu, T. Liu, X. Wang, and D. Tao, "On Compressing Deep Models by Low Rank and Sparse Decomposition," *in Prof. of International Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 67–76, 2017.

[13] T. Zhou and D. Tao, "GoDec: Randomized low-rank & sparse matrix decomposition in noisy case," *in Prof. of International Conference on Machine Learning (ICML)*, pp. 33–40, 2011.

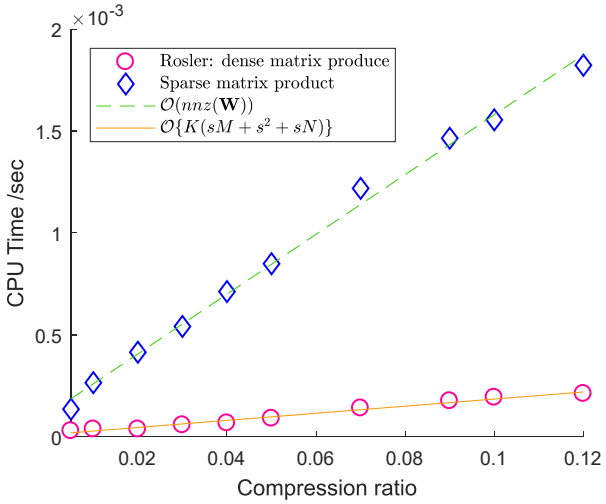
[14] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up convolutional neural networks with low rank expansions," *arXiv: Computer Vision and Pattern Recognition*, 2014.

[15] M. Denil, B. Shakibi, L. Dinh, M. Ranzato, and N. De Freitas, "Predicting parameters in deep learning," *in Prof. of Neural Information Processing Systems (NIPS)*, pp. 2148–2156, 2013.

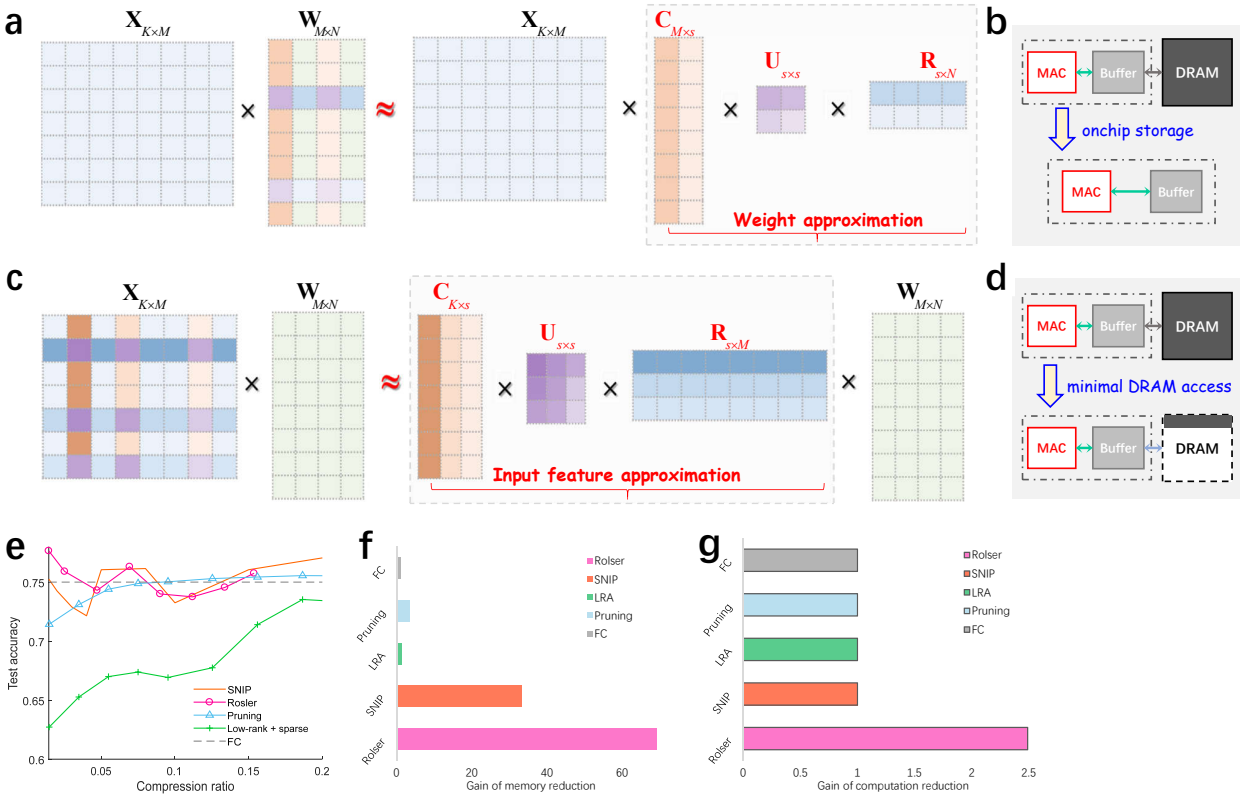
[16] P. Jihong, S. Samarakoon, B. Mehdi, and M. Debba, "Wireless network intelligence at the edge," *Proceedings of the IEEE*, vol. 107, no. 11, pp. 2204–2239, 2019.

[17] C. Sebastian, K. Jakub, H. B. McMahan, and A. Talwalkar, "Expanding the reach of federated learning by reducing client resource requirements," *arXiv preprint arXiv:1812.07210*, 2018.

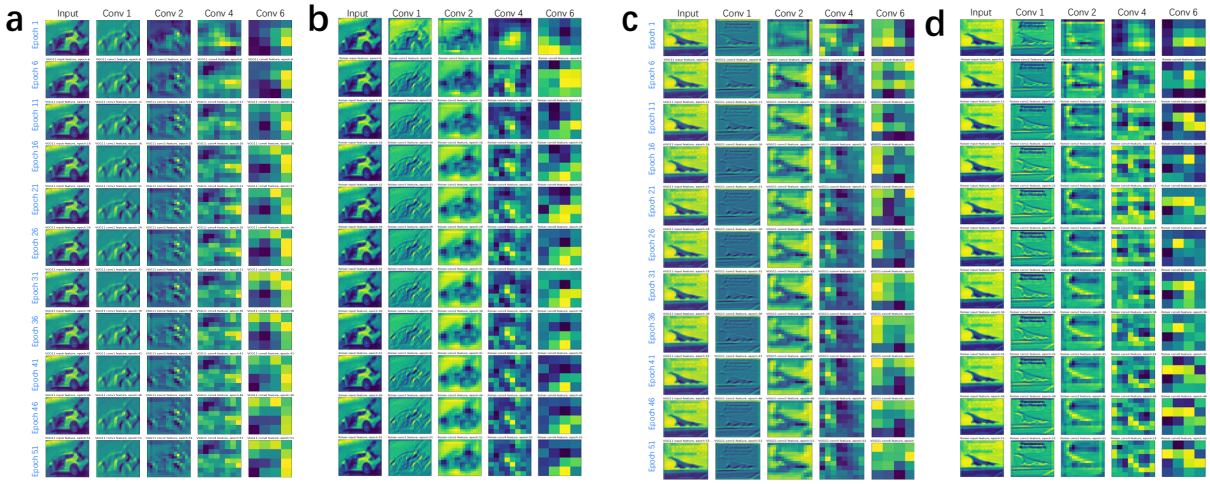
SUPPLEMENTARY FIGURE & TABLE



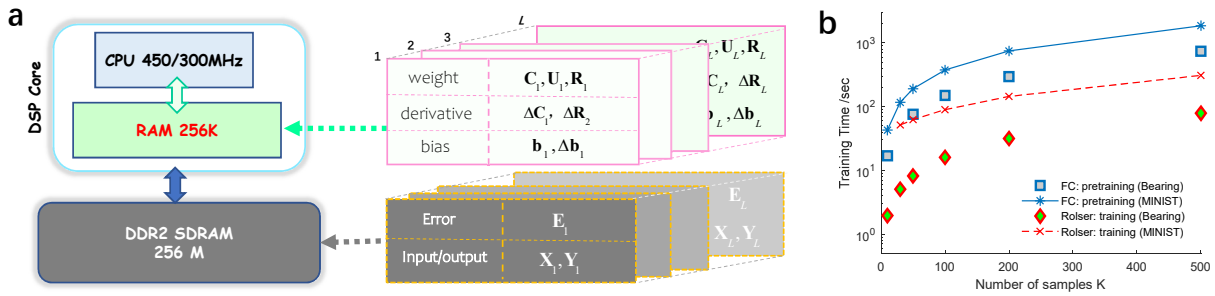
Supplementary Figure 1. **Computational complexity of sparse and dense matrix multiplication**. We evaluate the time complexity of the 1st layer in the MLP model (MINIST dataset); the weight matrix is $\mathbf{W} \in \mathbb{R}^{784\times512}$ ($M = 782,\ N = 512$), and the dense input matrix is $\mathbf{X} \in \mathbb{R}^{784\times100}$ (a mini-batch size K is100). For a sparse model, the compression ratio equals to the sparsity ratio. The optimized method in Matlab (R20170a) is directly used to compute sparse matrix multiplication.



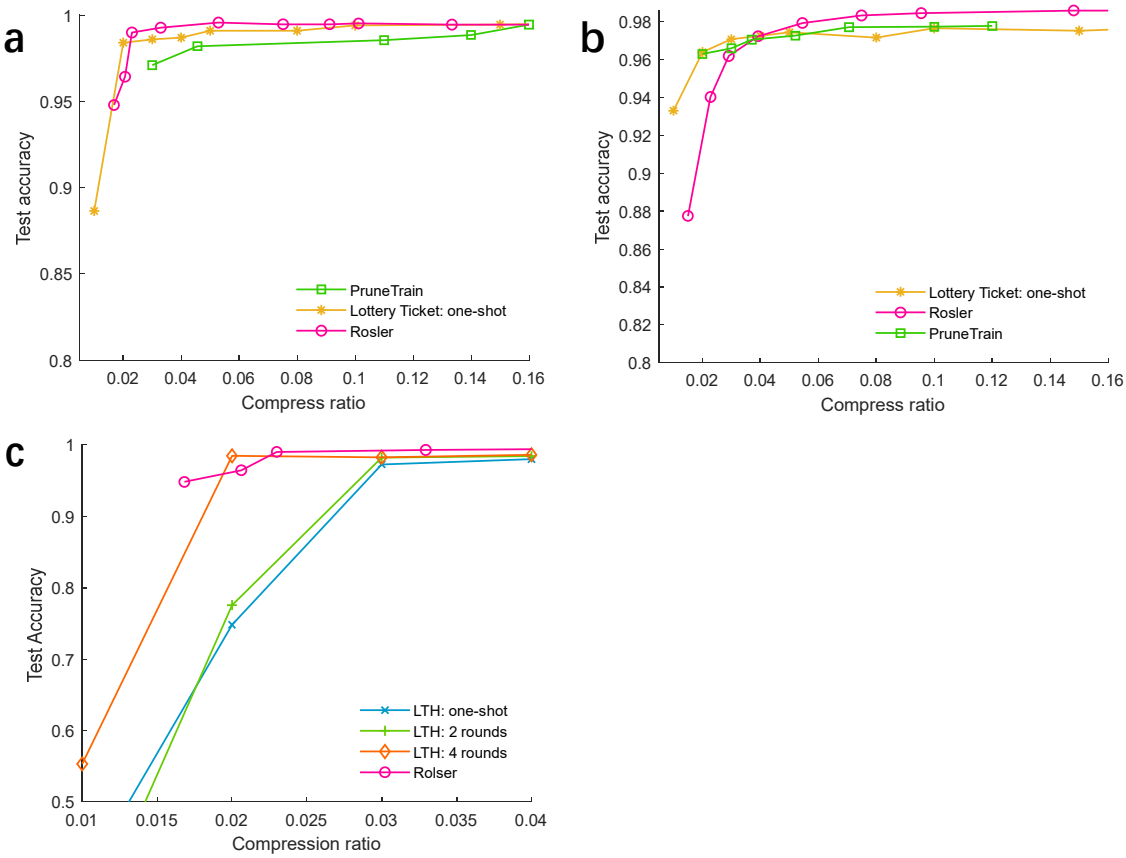
Supplementary Figure 2. **Sketch learning of CNN**. A kernel weight matrix (after the im2col unfolding) is represented by three sketches **(a)**, which is directly learned via the aRes-BP algorithm. A reduced model can be stored in on-chip RAM **(b)**. In Rosler, an input feature is also approximated by 3 sketches **(c)**, by minimizing the accessing of off-chip DRAM **(d)**. **(e)** Test accuracy, the Cat/Dag dataset (input image size 150×150); CNN with 4 convolution layers (3×3 kernel, max-pooling) and 3 FC layers (the numbers of channels in 4 convolution layers are 32, 64, 128 and 128; the numbers of nodes of 3 FC layers are 1024, 512 and 1). **(f)** Gain of memory reduction. **(g)** Gain of computation reduction (float-point).



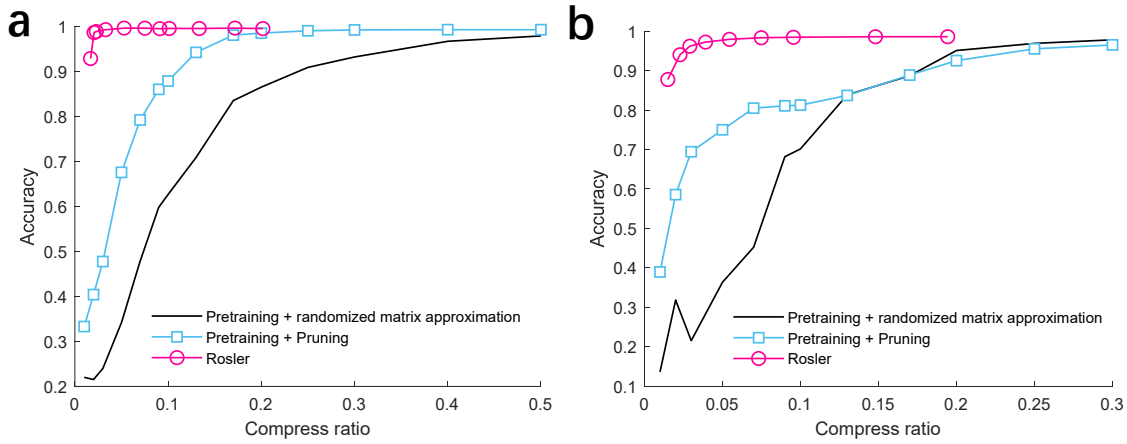
Supplementary Figure 3. **Learned feature maps on CIFAR-10 dataset**. Feature maps of an automobile; **(a)** VGG-11 model and **(b)** the tiny sketched model (the compression ratio is 0.0387). Feature maps of an airplane; **(c)** VGG-11 model and **(d)** the tiny sketched model (the compression ratio is 0.0387).



Supplementary Figure 4. **On-device training with low-cost hardware. (a)** Experimental setting of edge training on low-cost platforms. When a batch size is relatively large ($K > 100$), the input/output of each layer and the program are stored in off-chip DRAM. Other network parameters (including weights, derivatives of weight and bias vectors) of Rosler are put in the on-chip RAM. In the pre-training of FC network, such network parameters are all moved to the off-chip DRAM. **(b)** Time complexity of on-device training (20 iterations).



Supplementary Figure 5. **Performance of PruneTrain and Lottery Ticket Hypothesis methods**. **(a)** Test accuracy on the bearing data, 4-layer FC network (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). **(b)** Test accuracy on the MINIST data, 4-layer FC network (the number of nodes of input layer is 784, the number of nodes of 2 hidden layers are 512 and 256, the numbers of nodes of output layer is 10). For the LTH method, we adopt the one-shot mode. For the PruneTrain method, a similar one-time compression was used, i.e. the epochs for pre-training and re-training are equivalent. **(c)** Test accuracy of LTH under various iterative training rounds; the bearing data, 4-layer FC network as in **(a)** .



Supplementary Figure 6. **Randomized low-rank approximation based model compression**. Relying on a pre-trained model, randomized low-rank approximation (RLAR) was directly used for model compression, i.e. we directly approximated a pre-trained weight via three small matrices. **(a)** Test accuracy of various methods, the bearing dataset, 4-layer FC network (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). **(b)** Test accuracy, the MINIST dataset, 4-layer FC network (the number of nodes of input layer is 784, the number of nodes of 2 hidden layers are 512 and 256, the numbers of nodes of output layer is 10).

Supplementary Table I
REDUCTION OF NETWORK WEIGHTS, FEATURE MAPS AND COMPUTATIONAL COMPLEXITY

Layer #	Network Weights			Feature Maps			Computational Complexity		
	Full Net	Rosler	Ratio	Full Net	Rosler	Ratio	Full Net	Rosler	Ratio
Conv 1	864	864	1	59.1 M	33 M	0.559	1.89 G	1.05 G	0.557
Conv 2	18.4 K	10.6 K	0.576	149.3 M	26.3 M	0.176	9.55 G	1.68 G	0.176
Conv 3	73.7 K	42.2 K	0.573	66.6 M	8.48 M	0.127	8.52 G	1.06 G	0.125
Conv 4	147.5 K	76.8 K	0.521	25.9 M	2.4 M	0.093	3.32 G	282 M	0.085
FC 1	29.5 M	1.49 M	0.051	-	-	-	2.95 G	149 M	0.051
FC 2	524 K	47 K	0.09	-	-	-	54.4 M	4.7 M	0.09
FC 3	512	512	1	-	-	-	51.2 K	51.2 K	1
Total	30.3 M	1.67 M	**0.055**	300.9 M	70.27 M	**0.233**	26.28 G	4.23 G	**0.161**

Here, we consider the Cat-Dog classification task (input image size 150×150), CNN with 4 convolution layers (3×3 kernel, max-pooling) and 3 FC layers.

Supplementary Table II
FEDERATED LEARNING ON VGG-11 MODEL, CIFAR-10 DATASET

VGG-11 Model	**Test Accuracy**	**Memory Cost** (MB)	**Communication Cost** (MB)	**Computation Cost** (G flops)
FC DNN	0.851	500	500	0.172
SNIP	0.838	77.25 ($6.47\times$)	77.25 ($6.47\times$)	0.0664 ($2.59\times$)
Rosler	0.838	28.65 ($17.45\times$)	28.65 ($17.45\times$)	0.0418 ($4.11\times$)

[1] In federated learning, 3 clients are assumed and the local training epoch is 5. At a center entity, the averaged global model is obtained via multiple sketches. Both the BUFF structure in Rosler and the sparse structure in SNIP are determined in the initialization stage, which remain unchanged during the whole learning process. The compression ratio of Rosler is 0.0573, and the compression ratio of SNIP is 0.0515; CIFAR-10 dataset.

[2] Both SNIP and Rosler are implemented on low-cost device (DSP C6478) for edge inference; the input batch size K is 12; the computation time of FC DNN is 112 sec, and the computation time of Rosler is 25.7 sec (the weights are stored in off-chip DRAM).