NeurRev: Train Better Sparse Neural Network Practically via Neuron Revitalization (ICLR 2024)

Gen Li, Lu Yin, Jie Ji, Wei Niu, Minghai Qin, Bin Ren, Linke Guo, Shiwei Liu, Xiaolong Ma

ICLR

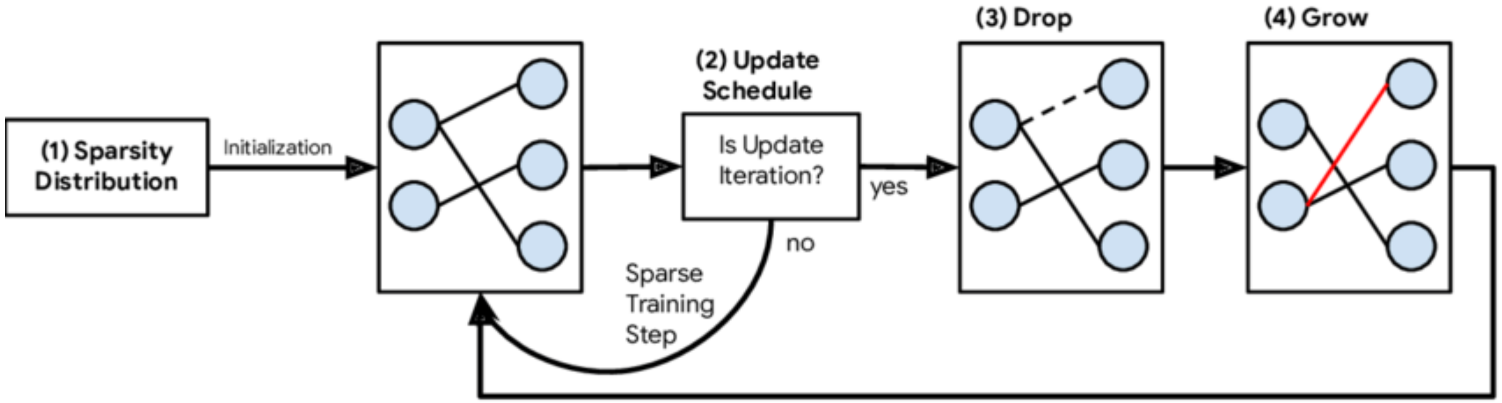
CLEMSON UNIVERSITY

Model Sparsity and Efficiency

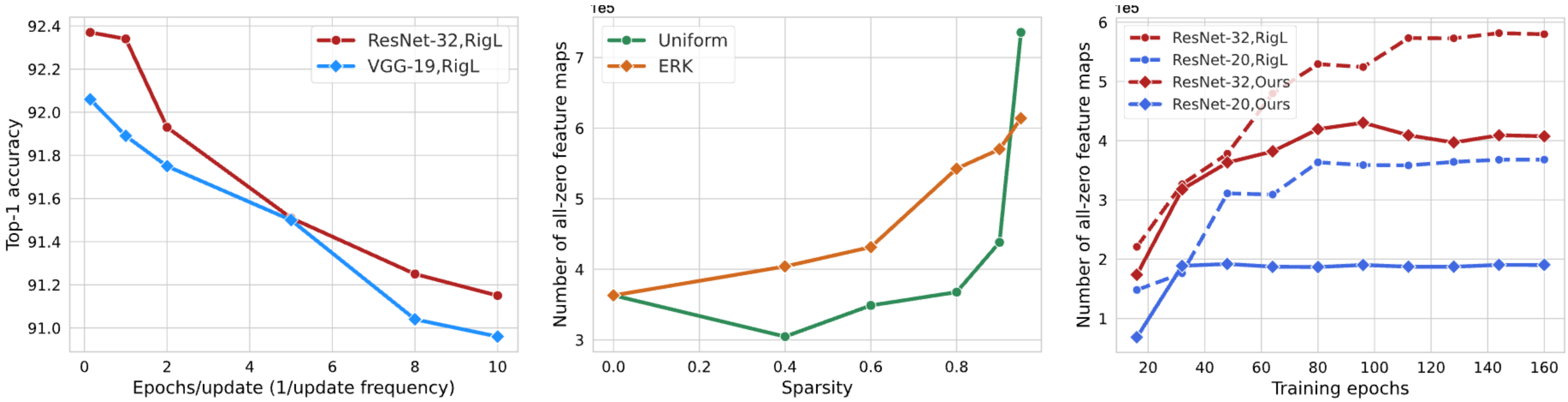
Train a Sparse Neural Network From Scratch (Sparse Training)

Static sparse training (SST) determines the structure of the sparse network at the initial stage of training by using the designed algorithm. Following that, the same sparse network structure is kept throughout the sparse training procedure.

Dynamic sparse training (DST) begins with a random selection of a sparse network structure at the starting stage of training and modifying the sparse network structure during the entire sparse training process in an effort to find a better sparse structure.



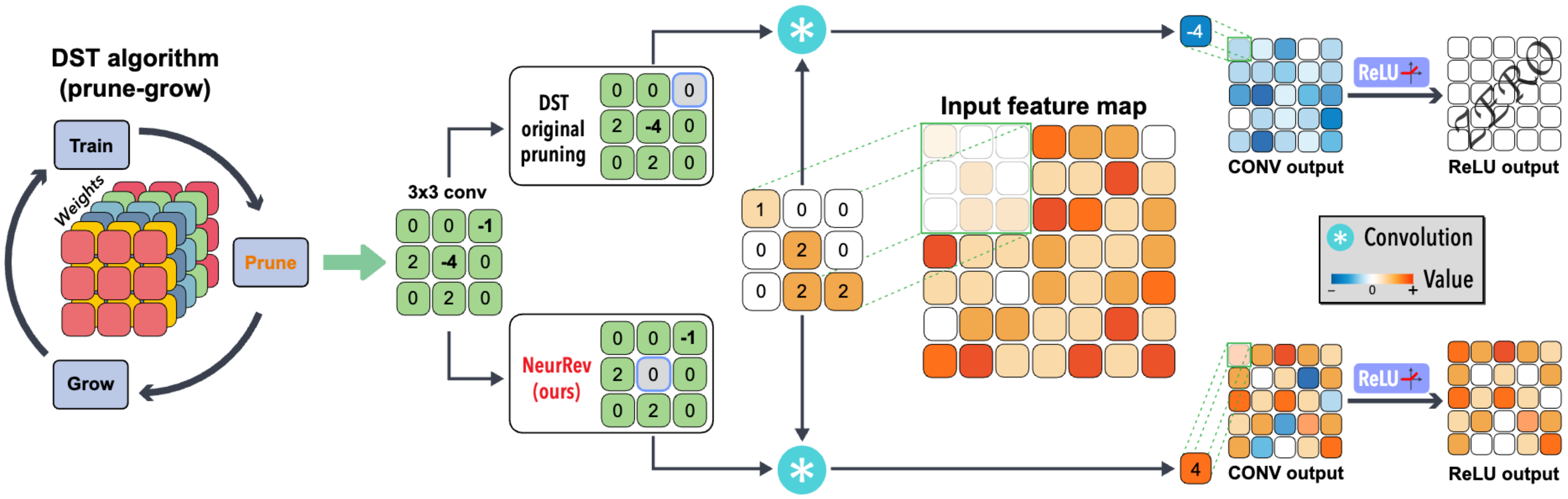
NeurRev



Finding: We identify the reason that sparsity produces **dormant neurons** in the DST process, and such dormant neurons are extremely persistent to the “prune-grow” dynamism of DST once they are generated.

CLEMSON UNIVERSITY

Overview of NeurRev



Definition: We argue that **dormant neurons** are those convolution filters whose weights contain negative values in **large magnitude**, **resulting in negative outputs** after convolution and being set to zero for post-ReLU outputs.

NeurRev

Dormant Neurons Search & Awake

Evaluating whether a neuron is a dormant neuron using layer output is computationally intensive due to the large volume of output feature maps. As we find that the gradients of dormant neurons are often zero, those neurons can hardly update weights in the training process. Based on this phenomenon, we can search for those dormant neurons according to their weight changes over a certain period.

Algorithm 1: NeurRev for DST

Input: $\boldsymbol{\theta}_s$, $\boldsymbol{\theta}_s^*$, s, ΔT_p, τ, p, τ_{stop}
Output: A sparse model satisfying the target parameter sparsity s.
Init: Initialize $\boldsymbol{\theta}_s$ according to s. $\boldsymbol{\theta}_s^* = \boldsymbol{\theta}_s$.
while $\tau < \tau_{stop}$ **do**
  if τ *mod* $\Delta T_p == 0$ **then**
    $\Delta\theta = \texttt{abs}(\boldsymbol{\theta}_s - \boldsymbol{\theta}_s^*)$
    $\boldsymbol{\theta}_s \leftarrow \texttt{Prune\&Grow}(\boldsymbol{\theta}_s, s - p)$
    $\texttt{Search\&Awake}(\boldsymbol{\theta}_s, \Delta\theta, p)$
  $\boldsymbol{\theta}_s^* = \boldsymbol{\theta}_s$
Continue sparse training from τ_{stop} to τ_{end}.

Algorithm 2: Search and Awake

Input: $\boldsymbol{\theta}$, $\boldsymbol{\theta}_s$, $\Delta\theta$, p
Output: A pruned sparse model.
Init: pruned_weights = 0
sorted_index = $\texttt{Sorted\_index}(\Delta\theta$, ascending_order)
for i **in** sorted_index **do**
  if $\boldsymbol{\theta}_s[i] < 0$ **then**
    $\texttt{Prune}(\boldsymbol{\theta}_s[i])$
    pruned_weights = pruned_weights + 1
  if *pruned_weights* $> p \cdot \|\boldsymbol{\theta}\|_0$ **then**
    `exit()`

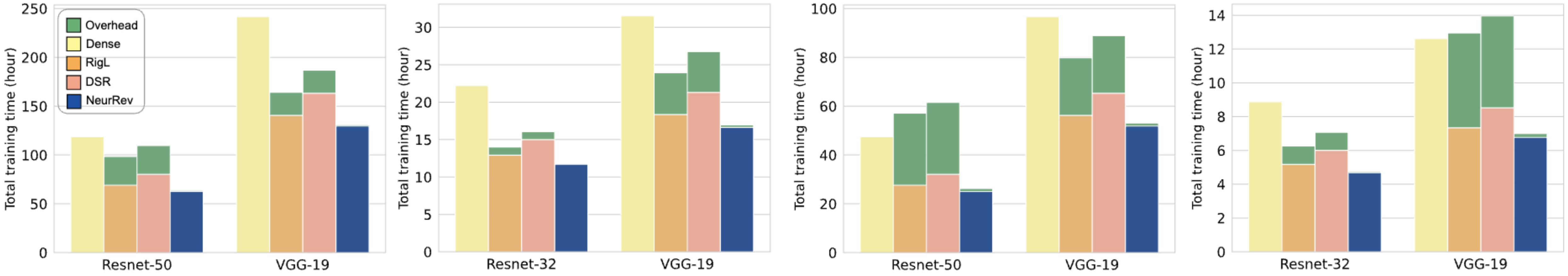
Result on CIFAR-10/100

Table 1: Test accuracy of pruned ResNet-32 on CIFAR-10/100.

Datasets	Sparsity Distribution	CIFAR-10			CIFAR-100		
Pruning ratio		90%	95%	98%	90%	95%	98%
ResNet-32	dense	94.88	94.88	94.88	74.94	74.94	74.94
LT [49]	non-uniform	92.31	91.06	88.78	68.99	65.02	57.37
SNIP [50]	non-uniform	92.59	91.01	87.51	68.89	65.02	57.37
GraSP [45]	non-uniform	92.38	91.39	88.81	69.24	66.50	58.43
Deep-R [51]	non-uniform	91.62	89.84	86.45	66.78	63.90	58.47
SET [11]	non-uniform	92.30	90.76	88.29	69.66	67.41	62.25
DSR [12]	non-uniform	92.97	91.61	88.46	69.63	68.20	61.24
RigL-ITOP [18]	uniform	93.19	92.08	89.36	70.46	68.39	64.16
RigL [15]	uniform	93.07	91.83	89.00	70.34	68.22	64.07
MEST+EM [17]	uniform	92.56	91.15	89.22	70.44	68.43	64.59
NeurRev (ours)	uniform	**93.31±0.11**	**92.18±0.14**	**89.96±0.12**	**70.87±0.08**	**68.77±0.12**	**64.91 ±0.06**
RigL [15]	ERK	93.55	92.39	90.22	70.62	68.47	64.14
RigL-ITOP [18]	ERK	93.70	92.78	90.40	71.16	69.38	66.35
NeurRev (ours)	ERK	**93.84±0.09**	**92.93±0.14**	**90.84±0.12**	**71.96±0.06**	**69.92±0.05**	**66.82 ±0.07**

Edge Training

End-to-End Training Acceleration with System Overhead

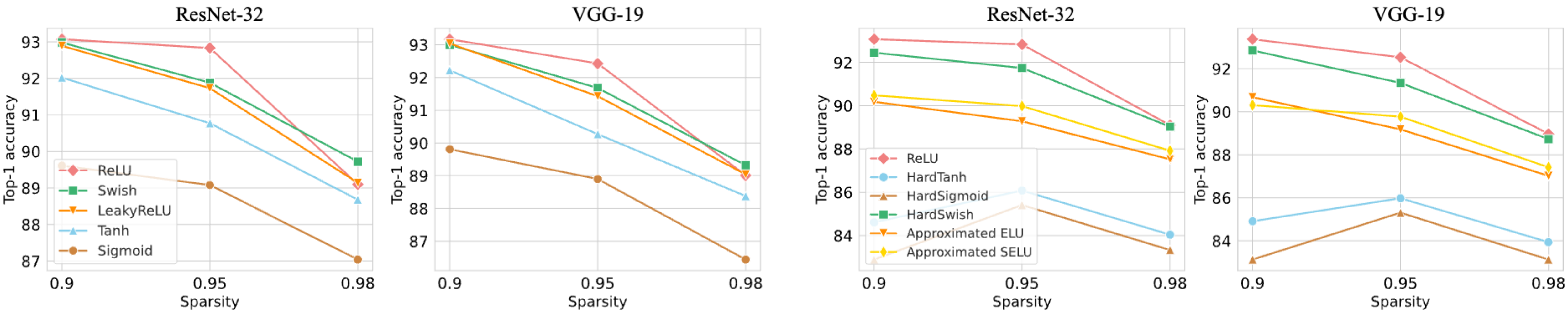


(a) ImageNet-CPU (b) CIFAR-10-CPU (c) ImageNet-GPU (d) CIFAR-10-GPU

NeurRev effectively reduces the dynamic update frequency, which enables efficient training on resource-constrained mobile devices. NeurRev (**blue bar**) achieves 1.8-2.1×better acceleration rate to dense training compared with other methods

Why ReLU?

ReLU is a versatile activation function.



(a) Software Accuracy (b) Hardware Accuracy

In (a), ReLU has an overall better performance in software accuracy. When the training is performed on a mobile device as shown in (b), most of the other activation functions need to be implemented with piecewise linear approximation. Even with specifically designed hardware versions (e.g., HardSwish, HardSigmoid, etc.), the accuracy of hardware still cannot match the ReLU-based DNN.

Thank you for your time