EFFICIENT DEEP LEARNING READING GROUP

02/26/2023

Yang Sui

SPDY: Accurate Pruning with speedup guarantees

ICML'22

Elias Frantar; Dan Alistarh

Motivation

- Unstructured Pruning (Weight Pruning)
- Previous Work:
  - Minimize the number of remaining weights
- This Work
  - Minimize the inference time
- Goal:
  - Automatically determines layer-wise sparsity
- Methods:
  - Dynamic Programming
  - Local Search

Introduction

- Pruning Methods:
  - Structured Pruning
  - Unstructured Pruning
- Runtime Side, unstructured sparsity is important:
  - Algorithms provides speedup on CPUs, GPUs or Specialized hardware.
  - Commodity CPUs, AMD models only cares sparsity instead of quantization.
- Key issue of previous unstructured pruning works:
  - Not consider the acceleration methods.

Introduction

- Contribution:
  - learned efficient Sparsity Profiles via Dynamic programming search (SPDY). Determine layer-wise sparsity to achieve a desired speedup.
  - First, optimization problem → dynamic programming solver.
  - Second, learns the layer-wise error-scores automatically, based on calibration dataset.

Optimization Problem

- Constrained Optimization Problem:

$$\min_{s_1,\ldots,s_L \in S} \sum_{\ell=1}^{L} e_\ell^{s_\ell} \quad \text{s.t.} \quad \sum_{\ell=1}^{L} t_\ell^{s_\ell} \leq T. \qquad (1)$$

- Assumption:
  - Overall execution time = Sum of the individual layer runtimes.
  - Pruning a layer ℓ to sparsity s ultimately incurs some model error $e_\ell^{s_\ell}$, which is additive.
- Integer linear program (ILP)
- However, NP-hard and requires exponential time to solve.

Efficient Solver

- Make time t as an integer-value
- Dynamic Programming
- Recursion:

$$E_\ell^t = \min_{s \in S} E_{\ell-1}^{t - t_\ell^s} + e_\ell^s \quad (2)$$

$$E_1^t = \min_{s \in S'} e_1^s \text{ if } S' = \{s \,|\, t_1^s = t\} \neq \emptyset \text{ else } \infty. \quad (3)$$

Algorithm 1 We efficiently compute the optimal layer-wise sparsity profile with execution time at most T given S, e_ℓ^s, t_ℓ^s and assuming that time is discretized, using bottom-up dynamic programming.

```
D ← L × (T + 1) matrix filled with ∞
P ← L × (T + 1) matrix
for s ∈ S do
  if e_1^s < D[1, t_1^s] then
    D[1, t_1^s] ← e_1^s;  P[1, t_1^s] ← s
  end if
end for
for ℓ = 2, ..., L do
  for s ∈ S do
    for t = t_ℓ^s + 1, ..., T do
      if e_ℓ^s + D[ℓ − 1, t − t_ℓ^s] < D[ℓ, t] then
        D[ℓ, t] ← e_ℓ^s + D[ℓ − 1, t − t_ℓ^s]; P[ℓ, t] ← s
      end if
    end for
  end for
end for
t ← argmin_t D[L, t] // return D[L, t] as optimal error
for ℓ = L, ..., 1 do
  s ← P[ℓ, t] // return s as optimal sparsity for layer ℓ
  t ← t − t_ℓ^s
end for
```

Error Metric e_ℓ^s

- Previous:
  - Weight Magnitude ; Squared Weight Magnitude ; Loss change
- Ours: “learning” or “search”

$$e_\ell^s = c_\ell \cdot \left(\frac{i}{|S|-1}\right)^2, \quad s = 1-(1-\delta)^i. \qquad (4)$$

T=10,000, |S| = 42, L = 52 for ResNet50.

- How to check optimal ”c”?

Quickly Check the Quality of a Sparsity Profile

- This database stores for each layer and each sparsity the “reconstruction” of the remaining weights after pruning.

$$\text{argmin}_{W^s} ||f_\ell(X,W) - f_\ell(X,W^s)||_2^2, \text{ for layer } \ell.$$

- First, we query the database for the corresponding reconstructed weights of each layer, each at its target sparsity.
- Second, we “stitch together” the resulting model from the reconstructed weights, and evaluate it on a given small validation set.

Determine sensitivity values C.

Algorithm 4 SPDY search for optimal sensitivity values $\mathbf{c}^*$. We use $k = 100$ and $\delta = 0.1$ in our experiments.

function $eval(\mathbf{c})$
  $e_\ell^s \leftarrow$ compute by formula (4) using $\mathbf{c}$ for all ℓ
  $s_\ell \leftarrow$ run DP algorithm with e_ℓ^s for all ℓ
  $M \leftarrow$ stitch model for s_ℓ from database
  Return calibration loss of M.

$\mathbf{c}^* \leftarrow$ sample uniform vector in $[0, 1]^L$
for k times **do**
  $\mathbf{c} \leftarrow$ sample uniform vector in $[0, 1]^L$
  if $eval(\mathbf{c}) < eval(\mathbf{c}^*)$ **then**
    $\mathbf{c}^* \leftarrow \mathbf{c}$
  end if
end for
for $d = \lceil \delta \cdot L \rceil, \ldots, 1$ **do**
  for k times **do**
    $\mathbf{c} \leftarrow \mathbf{c}^*$
    Randomly resample d items of $\mathbf{c}$ in $[0, 1]$
    if $eval(\mathbf{c}) < eval(\mathbf{c}^*)$ **then**
      $\mathbf{c}^* \leftarrow \mathbf{c}$
    end if
  end for
end for

Overall

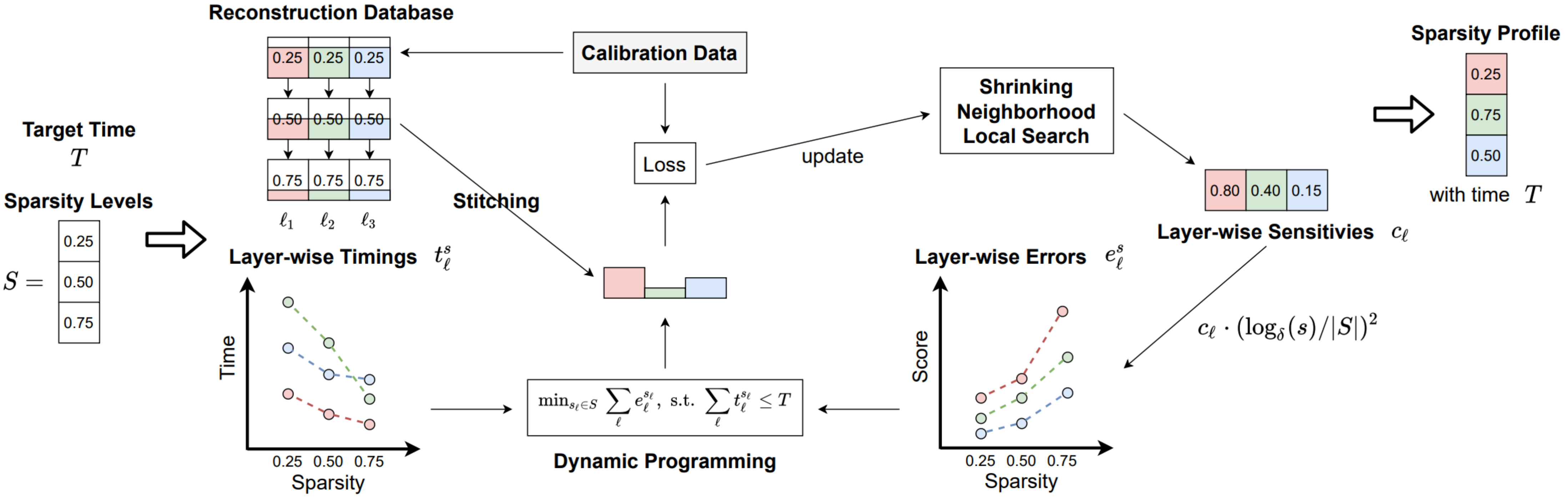


Figure 3. A visual overview of the full SPDY method.

T=10,000, |S| = 42, L = 52 for ResNet50.

Result

Model	Dense	Speed.	CPU	**SPDY**	Uni.	GMP
ResNet50	76.13	2.00×	AMD	**76.39**	76.01	75.85
ResNet50	76.13	2.50×	AMD	**75.56**	75.12	74.76
ResNet50	76.13	3.00×	AMD	**74.75**	74.02	73.44
ResNet50	76.13	3.50×	AMD	**73.06**	71.62	70.22
MobileNetV1	71.95	1.50×	Intel	**71.38**	61.33	70.63
YOLOv5s	56.40	1.50×	Intel	**55.90**	54.70	55.00
YOLOv5s	56.40	1.75×	Intel	**53.10**	50.90	47.20
YOLOv5m	64.20	1.75×	Intel	**62.50**	61.70	61.50
YOLOv5m	64.20	2.00×	Intel	**60.70**	58.30	57.20
BERT SQuAD	88.54	3.00×	Intel	**88.53**	88.22	87.98
BERT SQuAD	88.54	3.50×	Intel	**87.56**	87.23	87.22
BERT SQuAD	88.54	4.00×	Intel	**86.44**	85.63	85.13
BERT SQuAD*	88.54	4.00×	Intel	**87.14**	86.37	86.39

Table 4. Comparing accuracy metrics for sparsity profiles after gradual pruning models with respective state-of-the-art methods.

Sparse Double Descent: Where Network Pruning Aggravates Overfitting

ICML'22

Zheng He, et al.

Motivation

- Previous work:
  - Increase the model sparsity will prevents the overfitting.
- Sparse Double Descent:
  - Increase the model sparsity, test performance first gets worse (overfitting) then gets better (relieved overfitting).

Introduction

- Overparameterized DNNs are “good at” overfitting.
- In practice, DNNs often achieve higher generalization than smaller models.
- Recent, Deep Double Descent:
  - Model capacity increases, test performance first gets better then worse (overfitting) then gets better (relieved overfitting).
- Contribution:
  - Sparse Double Descent.
    - Increase the model sparsity, test performance first gets worse (overfitting) then gets better (relieved overfitting).
  - L2 learning distance
  - Contrary to the lotter ticket hypothesis.

Sparse Double Descent

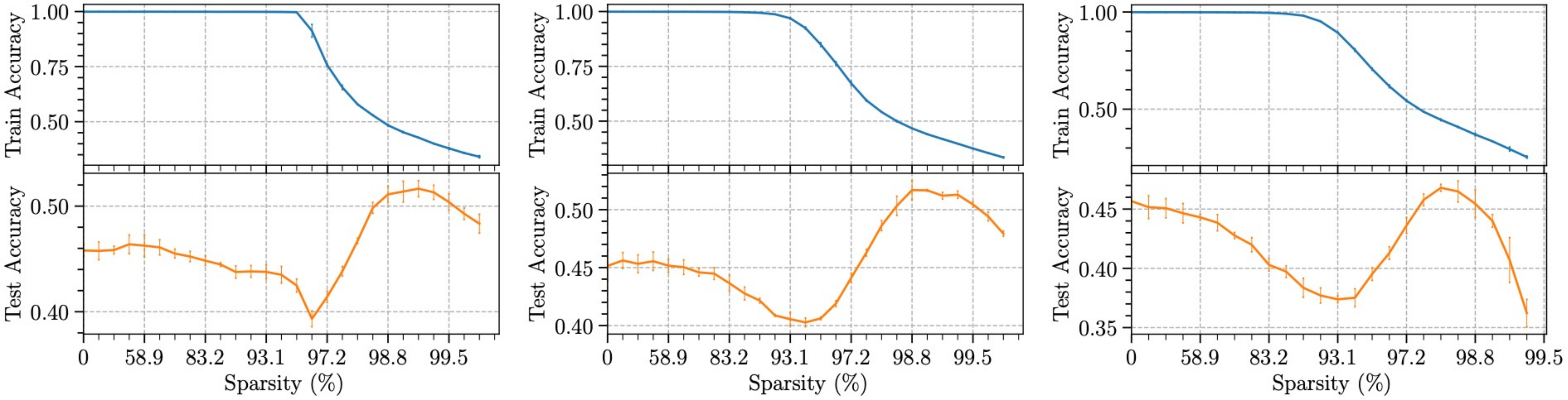


Figure 2. Sparse Double Descent of ResNet-18 on CIFAR-100 with 40% symmetric label noise, pruned using different strategies. We plot the train and test accuracy against sparsity. **Left**: Magnitude-based pruning. **Middle**: Gradient-based pruning. **Right**: Random pruning.

Four phases of model sparsity

1. Low sparsity:

   Pruned network = dense model

2. Critical phase:

   Severe Overfitting

3. Sweet Phase:

   Boosted accuracy

4. Collapsed Phase:

   Accuracy Drops

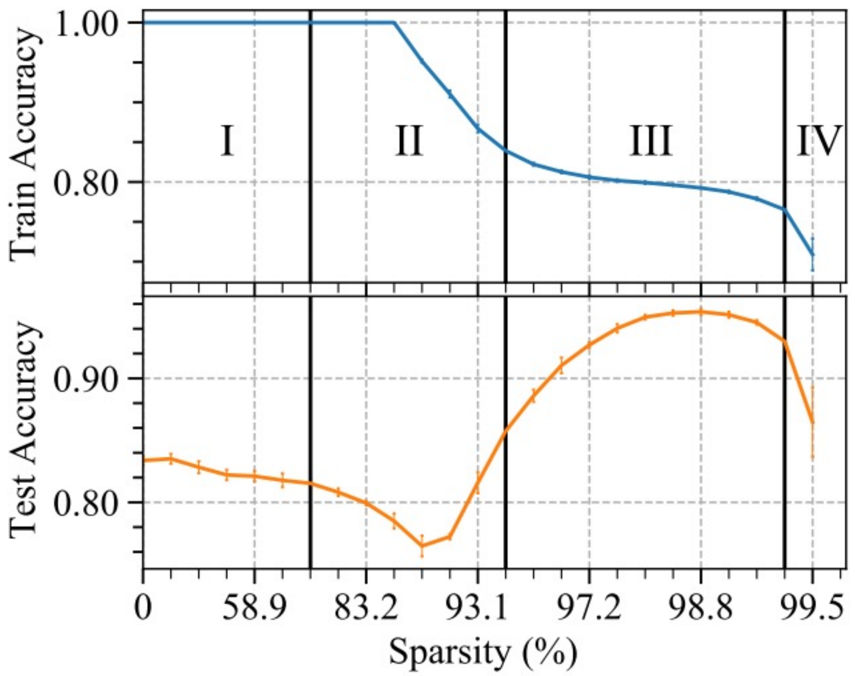


Figure 5. Illustration of four phases using the result of LeNet-300-100 on MNIST with 20% symmetric label noise. I: Light Phase. II: Critical Phase. III: Sweet Phase. IV: Collapsed Phase.

Why Sparse Double Descent Occurs

- The Learning Distance Hypothesis for Sparse Double Descent

- L2 distance:

$$D(\mathbf{w}_{init}, \mathbf{w}^i_{learned}) = ||\mathbf{w}_{init} - \mathbf{w}^i_{learned}||_2.$$

- Learning distance correlates

  the test accuracy

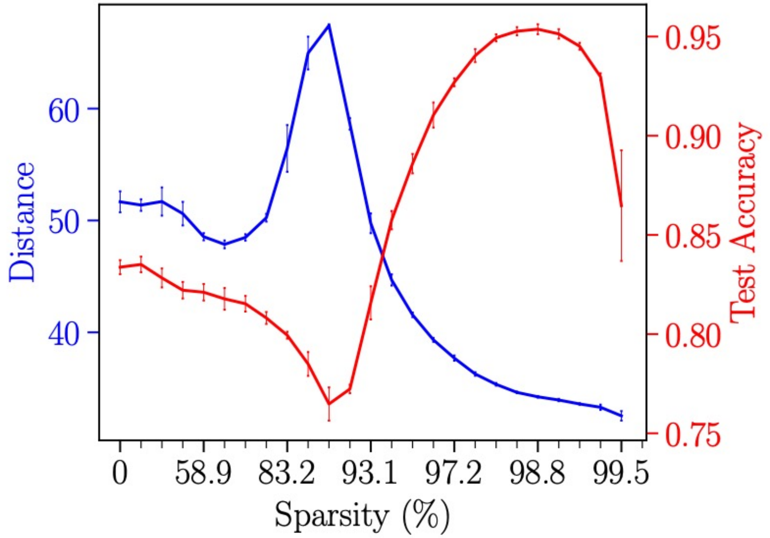


Figure 9. The curve of learning distance for LeNet-300-100 on MNIST with $\epsilon = 20\%$ may explain the double descent of test accuracy. As model sparsity increases, learning distance coincides the changes of test accuracy. The blue lines refer to ℓ_2 learning distance and the red lines are test accuracy.

Lottery tickets may not win at all time

- Reinitialized models could beat lottery ticket models at the same sparsity but different phases.

- Due to the Sparse Double Descent
- Add noise → fit noise.

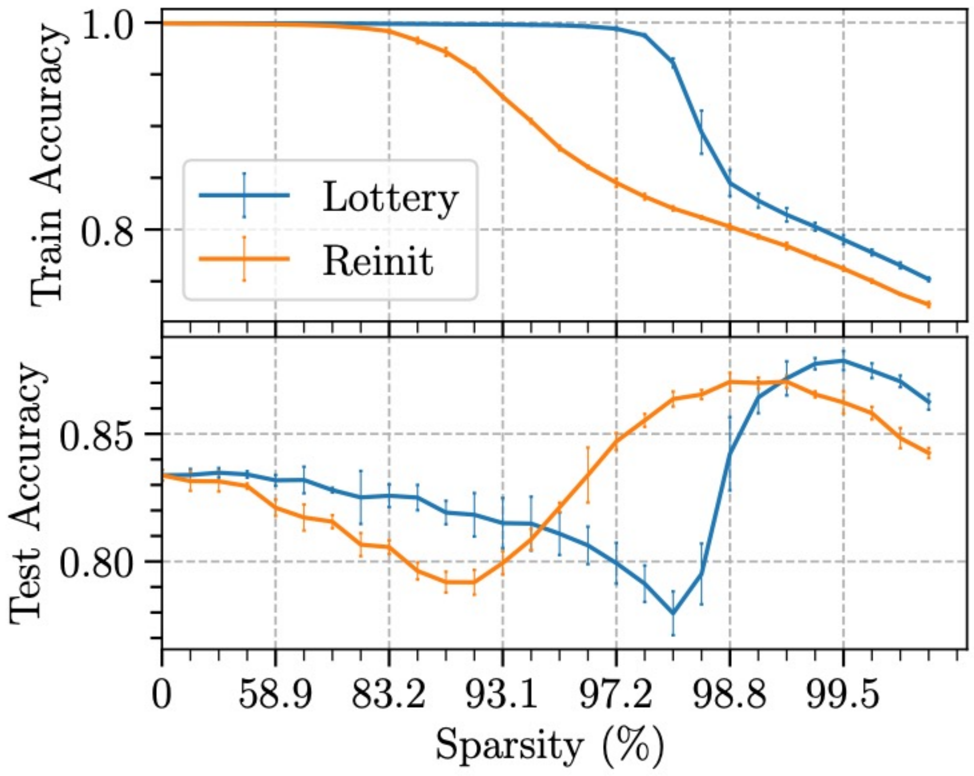


Figure 10. Performance of ResNet-18 on CIFAR-10 with $\epsilon = 20\%$ when retrained from either the original initialization (lottery tickets), or a random reinitialization. Reinitialization results sometimes surpass lottery results.

CHEX: CHannel EXploration for CNN Model Compression

CVPR'22

Zejiang Hou, et al.

Motivation

- Structured Pruning or Dense-to-Sparse training.
- Training from scratch
- Prune and regrow the channels throughout the training process.
  - tackle the channel pruning problem via a well-known column subset selection (CSS) formulation

Introduction

- Previous pruning:
  - pre-training a large model until convergence,
  - pruning a few unimportant channels by the pre-defined criterion
  - finetuning the pruned model to restore accuracy.
- long training time

- In this work, they dynamically adjust the importance of the channels via a periodic pruning and regrowing process
- allows the prematurely pruned channels to be recovered and prevents the model from losing the representation ability early in the training process.

Overview

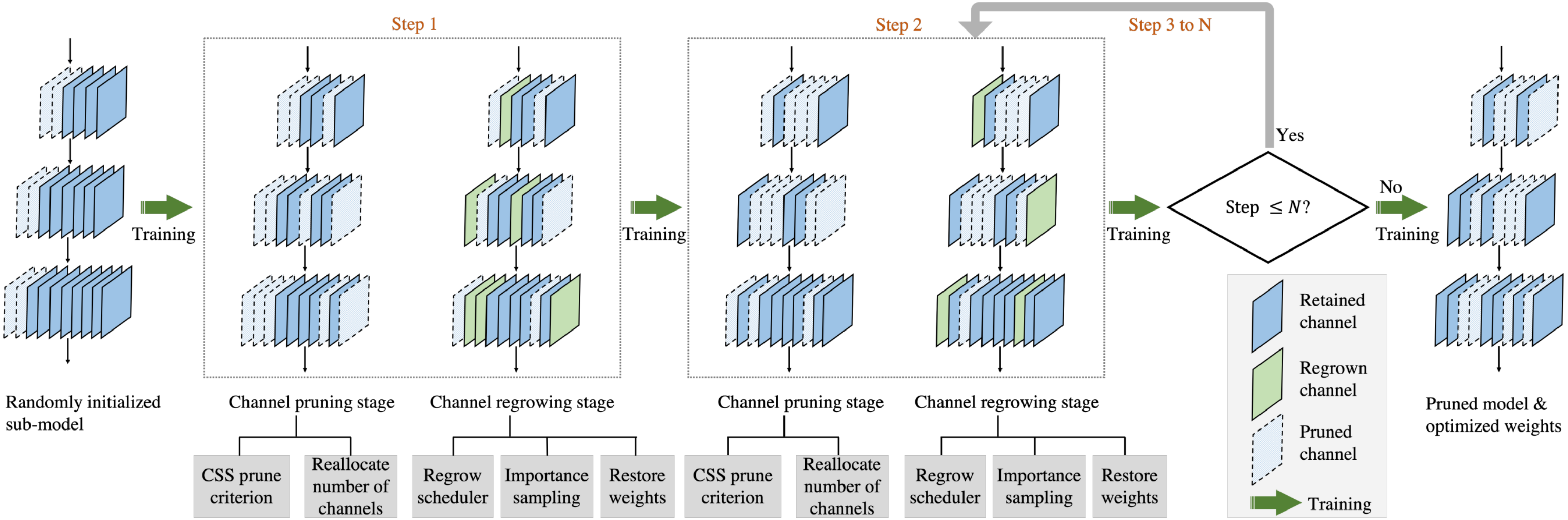


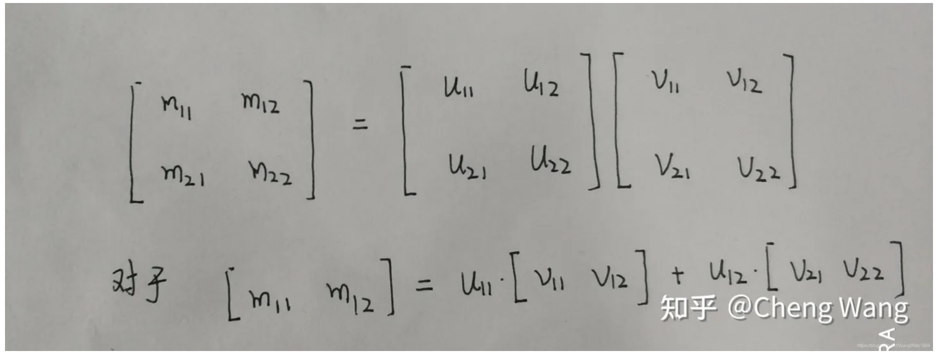
Figure 2. An illustration of our CHEX method, which jointly optimizes the weight values and explores the sub-model structure in one training pass from scratch. In CHEX, both retained and regrown channels in the sub-model are active, participating in the training iterations.

Pruning Stage1: Reallocate number of channels

- learnable scaling factors in batch normalization (BN).
- ranking all scaling factors in descending order and preserving the top $1 - S$ percent of the channels.

Pruning Stage2: CCS-Criterion

- Leverage score
- Information of N-th row.



Algorithm 2: CSS-based channel pruning.

1 **Input**: Model weights $\mathbf{w}^l$; pruning ratios κ^l ;
2 **Output**: The pruned layer l ;
3 Compute the number of retained channels
$\tilde{C}^l = \lceil (1 - \kappa^l) C^l \rceil$;
4 Compute the top $\tilde{C}^l$ right singular vectors $\mathbf{V}^l_{\tilde{C}^l}$ of $\mathbf{w}^l$;
5 Compute the leverage scores for all the channels in layer l
$\psi^l_j = \|[\mathbf{V}^l_{\tilde{C}^l}]_{j,:}\|^2_2$ for all $j \in [C^l]$;
6 Retain the important channels identified as
$\mathcal{T}^l = \text{ArgTopK}(\{\psi^l_j\}; \tilde{C}^l)$;
7 Prune channels $\{\mathbf{w}^l_{:,j}, j \notin \mathcal{T}^l\}$ from layer l ;

- U11 and U12 present the importance of the row with m11, m12.

Regrow Stage1: scheduler of number of regrown channels

- Cosine decay scheduler to gradually reduce the number of regrown channels

$$\delta_t = \frac{1}{2}\left(1 + \cos\left(\frac{t \cdot \pi}{T_{\max}/\Delta T}\right)\right)\delta_0$$

where δ0 is the initial regrowing factor, Tmax denotes the total exploration steps, and ΔT represents the frequency to invoke the pruning-regrowing steps.

Regrow Stage2: determine the channels to regrow

- Orthogonal projection formula:

$$\epsilon_j^l = \|\mathbf{w}_j^l - \mathbf{w}_{\mathcal{T}^l}^l (\mathbf{w}_{\mathcal{T}^l}^{l^T} \mathbf{w}_{\mathcal{T}^l}^l)^\dagger \mathbf{w}_{\mathcal{T}^l}^{l^T} \mathbf{w}_j^l\|_2^2. \qquad (2)$$

- A higher orthogonality value indicates that the channel is harder to approximate by others, and may have a better chance to be retained in the CSS pruning stage of the future steps.
- Be sampled with a relatively higher probability

Regrow Stage3: assign weight

- *Most recently used* (MRU) parameters, which are the last values before they are pruned

Overall algorithm

Algorithm 1: Overview of the CHEX method.

Input: An L-layer CNN model with weights $\mathbf{W} = \{\mathbf{w}^1, ..., \mathbf{w}^L\}$; target channel sparsity S; total training iterations T_{total}; initial regrowing factor δ_0; training iterations between two consecutive steps ΔT; total pruning-regrowing steps T_{max}; training set $\mathcal{D}$;

Output: A sub-model satisfying the target sparsity S and its optimal weight values $\mathbf{W}^*$;

Randomly initialize the model weights $\mathbf{W}$;

for *each training iteration* $t \in [T_{total}]$ **do**

  Sample a mini-batch from $\mathcal{D}$ and update the model weights $\mathbf{W}$;

  if $Mod(t, \Delta T) = 0$ *and* $t \leq T_{max}$ **then**

    Re-allocate the number of channels for each layer in the sub-model $\{\kappa^l, l \in [L]\}$ by Eq.(4) ;

    Prune $\{\kappa^l C^l, l \in [L]\}$ channels by CSS-based pruning in Algorithm 2 ;

    Compute the channel regrowing factor by a decay scheduler function ;

    Perform importance sampling-based channel regrowing in Algorithm 3 ;

Results

Method	PT	FLOPs	Top-1	Epochs	Method	PT	FLOPs	Top-1	Epochs
ResNet-18					***ResNet-50***				
PFP [45]	Y	1.27G	67.4%	270	GBN [82]	Y	2.4G	76.2%	350
SCOP [71]	Y	1.10G	69.2%	230	LeGR [4]	Y	2.4G	75.7%	150
SFP [24]	Y	1.04G	67.1%	200	SSS [35]	N	2.3G	71.8%	100
FPGM [26]	Y	1.04G	68.4%	200	TAS [9]	N	2.3G	76.2%	240
DMCP [16]	N	1.04G	69.0%	150	GAL [48]	Y	2.3G	72.0%	150
CHEX	N	1.03G	**69.6%**	250	Hrank [46]	Y	2.3G	75.0%	570
ResNet-34					Taylor [62]	Y	2.2G	74.5%	-
Taylor [62]	Y	2.8G	72.8%	-	C-SGD [6]	Y	2.2G	74.9%	-
SFP [24]	Y	2.2G	71.8%	200	SCOP [71]	Y	2.2G	76.0%	230
FPGM [26]	Y	2.2G	72.5%	200	DSA [63]	N	2.0G	74.7%	120
GFS [79]	Y	2.1G	72.9%	240	CafeNet [69]	N	2.0G	76.9%	300
DMC [12]	Y	2.1G	72.6%	490	**CHEX-1**	N	2.0G	**77.4%**	250
NPPM [11]	Y	2.1G	73.0%	390	SCP [37]	N	1.9G	75.3%	200
SCOP [71]	Y	2.0G	72.6%	230	Hinge [44]	Y	1.9G	74.7%	-
CafeNet [69]	N	1.8G	73.1%	300	AdaptDCP [89]	Y	1.9G	75.2%	210
CHEX	N	2.0G	**73.5%**	250	LFPC [23]	Y	1.6G	74.5%	235
ResNet-101					ResRep [8]	Y	1.5G	75.3%	270
SFP [24]	Y	4.4G	77.5%	200	Polarize [88]	Y	1.2G	74.2%	248
FPGM [26]	Y	4.4G	77.3%	200	DSNet [41]	Y	1.2G	74.6%	150
PFP [45]	Y	4.2G	76.4%	270	CURL [56]	Y	1.1G	73.4%	190
AOFP [7]	Y	3.8G	76.4%	-	DMCP [16]	N	1.1G	74.1%	150
NPPM [11]	Y	3.5G	77.8%	390	MetaPrune [52]	N	1.0G	73.4%	160
DMC [12]	Y	3.3G	77.4%	490	EagleEye [40]	Y	1.0G	74.2%	240
CHEX-1	N	3.4G	**78.8%**	250	CafeNet [69]	N	1.0G	75.3%	300
CHEX-2	N	1.9G	**77.6%**	250	**CHEX-2**	N	1.0G	**76.0%**	250

(a)