Efficient Training I

Sandeep Polisetty, Wenlong Zhao

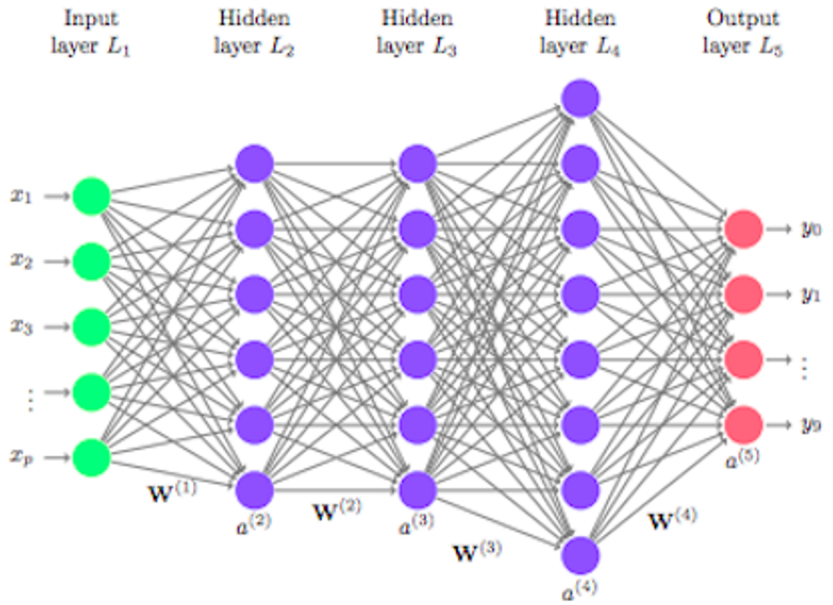
Why is DNN training expensive ?

1. It is large.
    a. during training (GPU memory)
    b. during inference (need pruning)
2. It takes forever.
3. It sometime needs to be repeatedly done.

Capuchin: Tensor-based GPU Memory Management for Deep Learning

Xuan Peng, Xuanhua Shi, Hulin Dai, Hai Jin, Weiliang Ma, Qian Xiong, Fan Yang, Xuehai Qian

Problem:



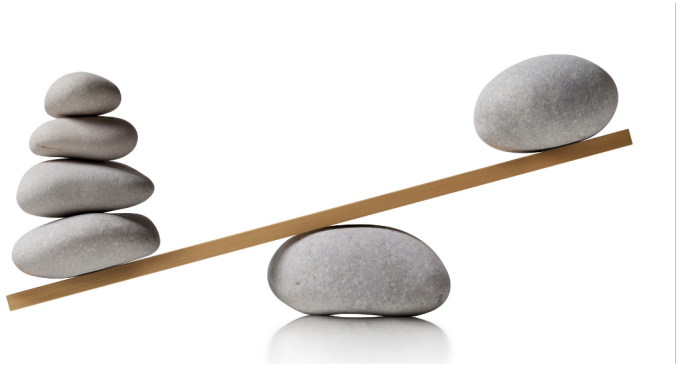
Memory of GPU is limited.

More specifically **feature maps.**

- Values in the forward prop must be saved for backward prop

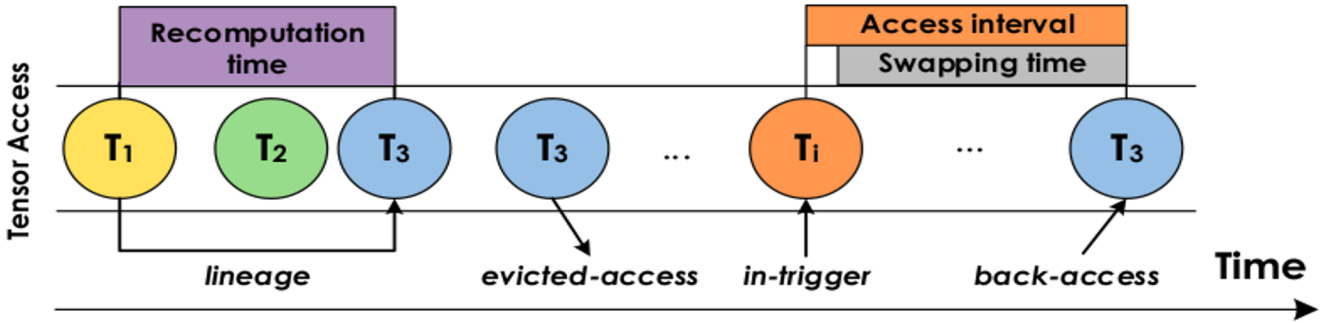
First Solution(Design Space):

1. Store everything into DRAM (What if communication costs are too high)
2. Recompute everything (Cost of recomputation is too high)



Schedule we have everything when we need.

1. What/When to evict, what/when to prefetch ? (Swapping)
2. What/when to recompute ? (Recomputation)



Assumptions of the current state of art

1. Computation graph is available (static analysis)
    a. Not suitable for RNN or dynamic computation graphs.
2. Decisions are made at granularity of layer.
    a. Fine grained tensor properties are ignored.
3. System properties such as Network bandwidth are ignored.
4. Static analysis does not take into account non determinism of concurrency.

How

1. Rely on historic pattern .
2. Use system properties (time to transfer) and tune based on reality.
3. Maintain Lineage to know you can recompute.
4. Having an memory changes the cost of computing its children

Approximate algorithm

1. Use swap as much as possible.

   Why is it always better to prefetch if we can ?

a. If swapped ahead of time, time to transfer is overlapped. There might be a time with sufficient communication bandwidth.
b. Cost of recomputation cannot be offset. We always have something to compute

FFT = **(historical)** When you next need it - Last time you used it

Sort by FFT and set **in-trigger**

(calculated using network bandwidth-not accurate so tune)

Approximate algorithm

1. Still having memory pressure.
2. Evict with high MSPS . **Why** ?

$$MSPS = \frac{Memory\ Saving}{Recomputation\ Time}$$

Recomputation (**historical**)

1. best swap benefit (high FFT) option cost O1.
2. best recomputation(high MSPS) cost option O2

If swap cost < recomputation cost:

Do swap

Else Recompute.

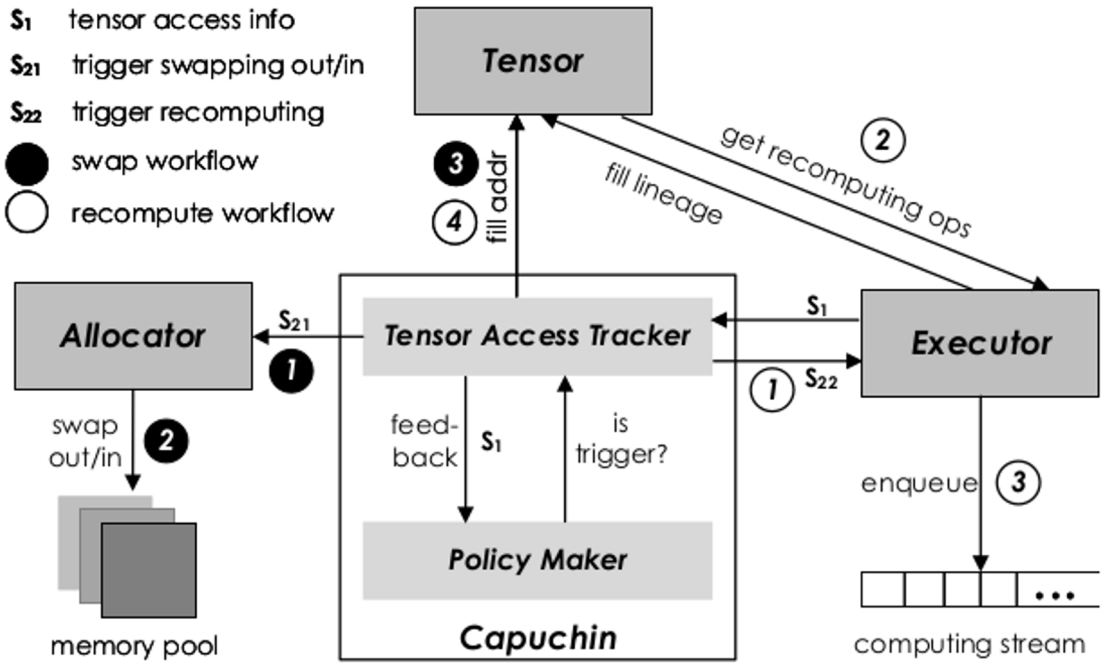
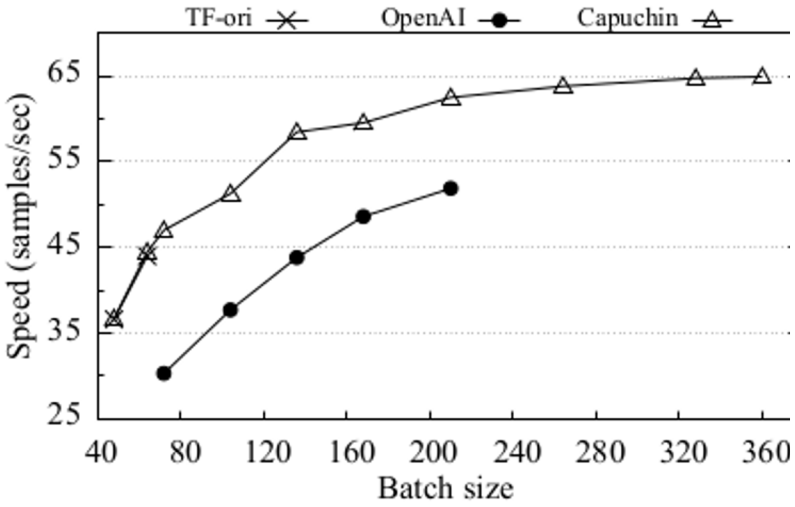


Figure 5. *Capuchin* System Architecture

Improves memory consumption and throughput



(f) BERT

Table 2. Maximum Batch Size in Graph Mode

Models	TF-ori	vDNN	OpenAI	*Capuchin*
Vgg16	228	272	260	**350**
ResNet-50	190	520	540	**1014**
ResNet-152	86	330	440	**798**
InceptionV3	160	400	400	**716**
InceptionV4	88	220	220	**468**
BERT	64	-	210	**450**

Let’s think of a few scenarios disprove this work

1. If GPU had huge amount of memory making recomputation unnecessary.
2. If GPU had infinite compute power, making memory pointless.

Drawing Early-Bird Tickets: Toward More Efficient Training of Deep Learning Networks

Haoran You, Chaojian Li, Pengfei Xu, Yonggan Fu, Yue Wang, Richard G. Baraniuk, Yingyan Lin, Xiaohan Chen, Zhangyang Wang

Problem and Motivation

Neural networks are large and expensive to train.

ResNet50: 14 days on a single M40 GPU

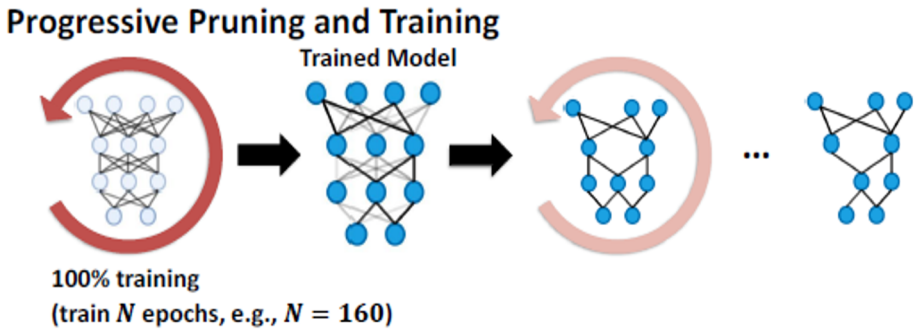
A DNN: $10K, as much carbon as five cars in their lifetimes

Problem and Motivation

Pruning: smaller networks & good performance at inference

Frankle & Carbin, 2019: Dense, randomly initialized networks have winning tickets (small but critical subnetworks) that can be trained alone to achieve similar accuracy in similar numbers of iterations

But common pruning methods are even more expensive to train.

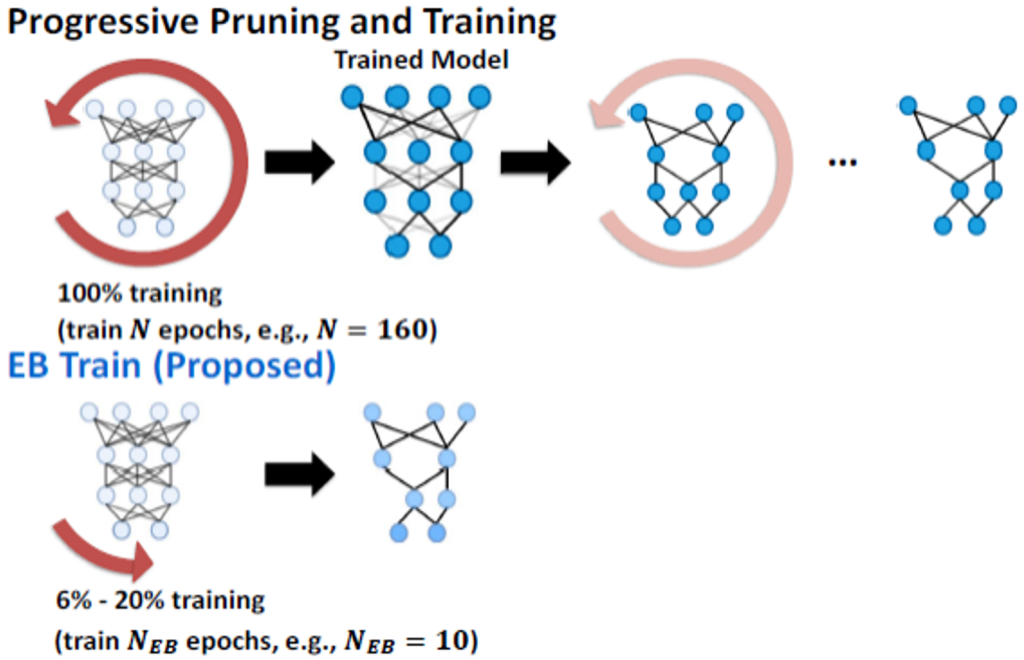


Can we identify winning tickets in an early stage?

1. Yes, winning tickets can be identified at a early training stage
2. A mask distance metric to identify EB tickets
    a. low computational overhead
    b. without needing to know the true winning tickets that emerge after the full training

Can we identify winning tickets in an early stage?

3. Efficient Early-Bird Training



Related Work

Rahaman et al., 2019; Xu et al., 2019: deep networks will first learn low-complexity (lower-frequency) functional components, before absorbing high-frequency features: the former being more robust to perturbations.

Achille et al., 2019: the early stage of training seems to first discover the important connections and the connectivity patterns between layers, which becomes relatively fixed in the later training stage.

Li et al., 2019: training a deep network with a large initial learning rate helps the model focus on memorizing easier-to-fit, more generalizable pattern faster and better

Do EB tickets always exist?

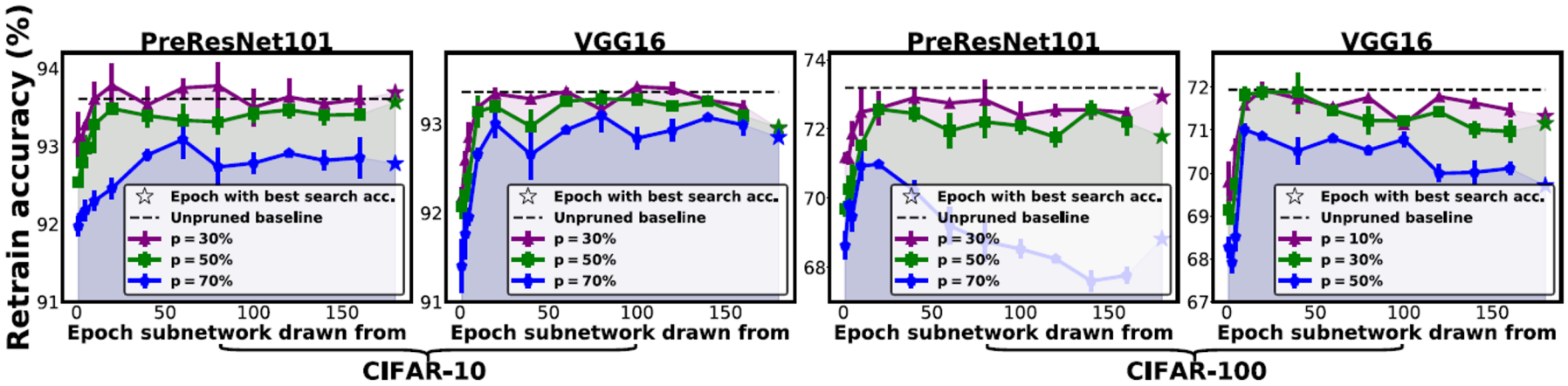


Figure 1: Retraining accuracy vs. epoch numbers at which the subnetworks are drawn, for both PreResNet101 and VGG16 on the CIFAR-10/100 datasets, where p indicates the channel pruning ratio and the dashed line shows the accuracy of the corresponding dense model on the same dataset, ☆ denotes the retraining accuracies of subnetworks drawn from the epochs with the best search accuracies, and error bars show the minimum and maximum of three runs.

Do EB tickets always exist?

1. There exist EB tickets that outperform later tickets.
2. There exist EB tickets that outperform baselines.

Large learning rates are important

Table 1: The retraining accuracy of subnetworks drawn at different training epochs using different learning rate schedules, with a pruning ratio of 0.5. Here [0, 100] represents $[0_{LR \to 0.01}, 100_{LR \to 0.001}]$ while [80, 120] denotes $[80_{LR \to 0.01}, 120_{LR \to 0.001}]$, for compactness.

	LR Schedule	Retrain acc. (%) (CIFAR-100)				Retrain acc. (%) (CIFAR-10)			
		10	20	40	final	10	20	40	final
VGG16	[0, 100]	66.70	67.15	66.96	69.72	92.88	93.03	92.80	92.64
	[80, 120]	**71.11**	**71.07**	**69.14**	**69.74**	**93.26**	**93.34**	**93.20**	**92.96**
PreResNet101	[0, 100]	69.68	69.69	69.79	70.96	92.41	92.72	92.42	93.05
	[80, 120]	**71.58**	**72.67**	**72.67**	**71.52**	**93.60**	**93.46**	**93.56**	**93.42**

Low-precision training does not destroy EB tickets

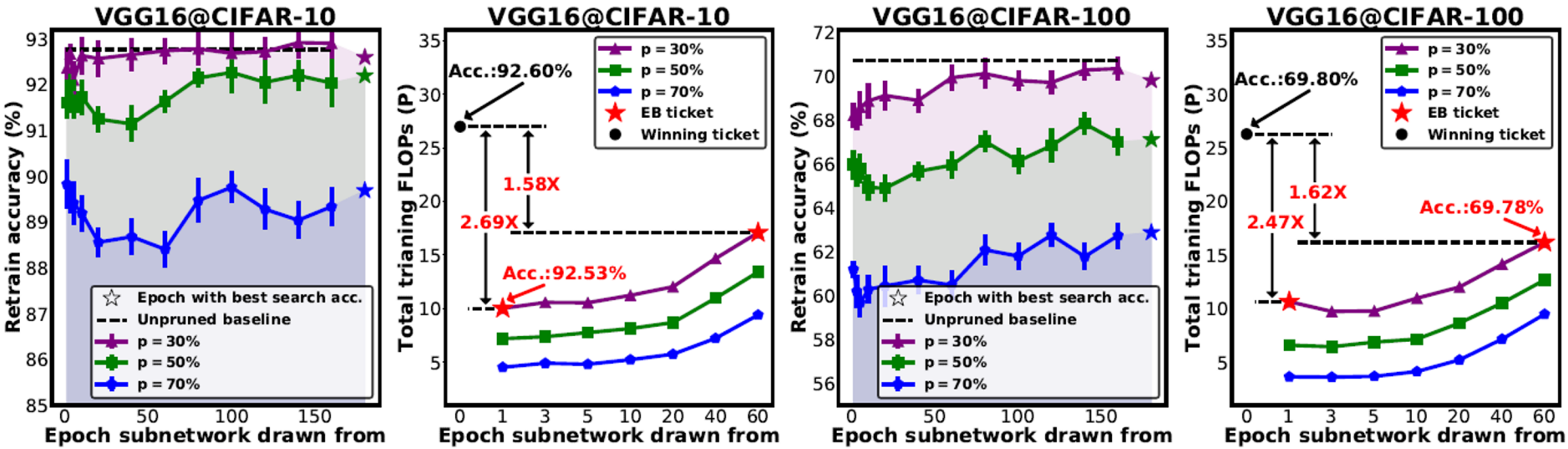


Figure 2: Retraining accuracy and total training FLOPs comparison vs. epoch number at which the subnetwork is drawn, when using 8 bits precision during the stage of identifying EB tickets based on the VGG16 model and CIFAR-10/100 datasets, where p indicates the channel-wise pruning ratio and the dashed line shows the accuracy of the corresponding dense model on the same dataset.

How to identify EB tickets practically?

Given a target pruning ratio p, prune the channels.

Denote the pruned channels as 0 while the kept ones as 1, the original network can be mapped into a binary “ticket mask”.

Mask distance: the Hamming distance between their two ticket masks.

Stop to draw EB tickets when the last five recorded mask distances are all smaller than epsilon.

How to identify EB tickets practically?

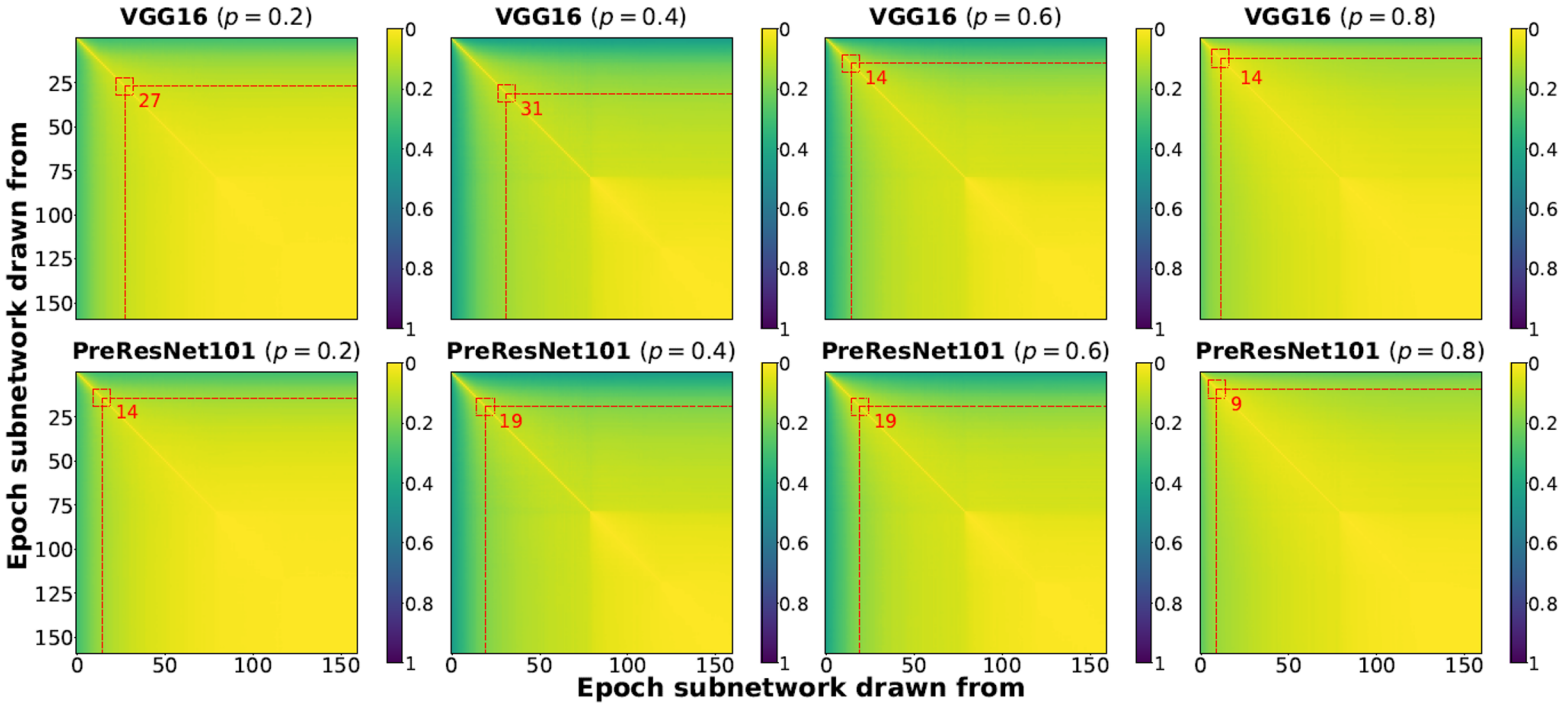
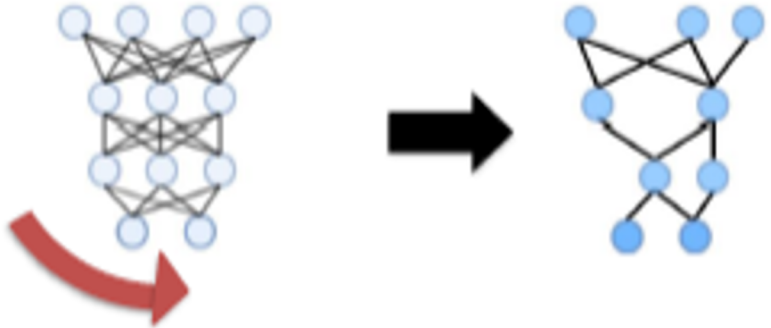


Figure 3: Visualization of the pairwise mask distance matrix for VGG16 and PreResNet101 on CIFAR-100.

EB Train (Proposed)



6% - 20% training

(train N_{EB} epochs, e.g., $N_{EB} = 10$)

Conclusion

1. Winning tickets can be identified at the very early training stage via low-cost training schemes (eg. early stopping, low-precision training) at large learning rates.
2. The proposed mask distance can effectively and efficiently identify Early-Bird tickets.
3. The proposed efficient training via EB tickets can achieve up to 4.7 energy savings while maintaining comparable or even better accuracy on various networks and datasets as compared to the most competitive state-of-the-art training methods.

DeltaGrad: Rapid retraining of machine learning models

Yinjun Wu, Edgar Dobriban, Susan B. Davidson

Motivation

Machine learning models may need to be retrained on slightly changed datasets.

- Data removal for privacy or robustness
- Bias correction

Retraining from scratch is expensive.

Technical Contributions

Proposed DeltaGrad for rapid retraining of gradient descent based machine learning models using information cached during training, when slight changes happen in the training dataset.

Showed experimentally that it is accurate and fast on several medium-scale problems on standard datasets, including two-layer neural networks.

The speed-ups can be up to 6.5x with negligible accuracy loss.

Problem Setup

The training set $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n$ has n samples. The loss or objective function for a general machine learning model is defined as:

$$F(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} F_i(\mathbf{w})$$

where $\mathbf{w}$ represents a vector of the model parameters and $F_i(\mathbf{w})$ is the loss for the i-th sample. The gradient and Hessian matrix of $F(\mathbf{w})$ are

$$\nabla F(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} \nabla F_i(\mathbf{w}), \quad \mathbf{H}(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} \mathbf{H}_i(\mathbf{w})$$

Suppose the model parameter is updated through mini-batch stochastic gradient descent (SGD) for $t = 1, \ldots, T$:

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \frac{\eta_t}{B}\sum_{i\in\mathscr{B}_t} \nabla F_i(\mathbf{w}_t)$$

where $\mathscr{B}_t$ is a randomly sampled mini-batch of size B and η_t is the learning rate at the t_{th} iteration. As a special case of SGD, the update rule of gradient descent (GD) is $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta_t/n \sum_{i=1}^n \nabla F_i(\mathbf{w}_t)$. After training on the full dataset, the training samples with indices $R = \{i_1, i_2, \ldots, i_r\}$ are removed, where $r \ll n$. Our goal is to efficiently update the model parameter to the minimizer of the new empirical loss. Our algorithm also applies when r new datapoints are *added*.

The naive solution is to apply GD directly over the remaining training samples (we use $\mathbf{w}^U$ to denote the corresponding model parameter), i.e. run:

$$\mathbf{w}^U{}_{t+1} \leftarrow \mathbf{w}^U{}_t - \frac{\eta_t}{n-r}\sum_{i\notin R} \nabla F_i\left(\mathbf{w}^U{}_t\right) \qquad (1)$$

which aims to minimize $F^U(\mathbf{w}) = 1/(n-r)\sum_{i\notin R} F_i(\mathbf{w})$.

DeltaGrad

Naive solution: $\mathbf{w}^U{}_{t+1} \leftarrow \mathbf{w}^U{}_t - \frac{\eta_t}{n-r} \sum_{i \notin R} \nabla F_i \left(\mathbf{w}^U{}_t \right)$

DeltaGrad:

$$\mathbf{w}^I{}_{t+1} = \mathbf{w}^I{}_t - \frac{\eta_t}{n-r} \left[n \nabla F \left(\mathbf{w}^I{}_t \right) - \sum_{i \in R} \nabla F_i \left(\mathbf{w}^I{}_t \right) \right]$$

Approximate

$n \nabla F \left(\mathbf{w}^I{}_t \right) = \sum_{i=1}^{n} \nabla F_i \left(\mathbf{w}^I{}_t \right)$ by leveraging the historical gradient $\nabla F \left(\mathbf{w}_t \right)$ (recall that $\mathbf{w}_t$ is the model parameter before deletions), for each of the T iterations.

DeltaGrad

Assuming strong convexity, DeltaGrad approximates the correct iteration values well when the dataset change ratio is low and the iteration number is large.

DeltaGrad can be extended to stochastic gradient descent.

Table 1. Prediction accuracy of BaseL and DeltaGrad with batch addition/deletion. MNIST^n refers to MNIST with a neural net.

Dataset		BaseL(%)	DeltaGrad(%)
Add (0.005%)	MNIST	87.530 ± 0.0025	87.530 ± 0.0025
	MNIST^n	92.340 ± 0.002	92.340 ± 0.002
	covtype	62.991 ± 0.0027	62.991 ± 0.0027
	HIGGS	55.372 ± 0.0002	55.372 ± 0.0002
	RCV1	92.222 ± 0.00004	92.222 ± 0.00004
Add (1%)	MNIST	87.540 ± 0.0011	87.542 ± 0.0011
	MNIST^n	92.397 ± 0.001	92.397 ± 0.001
	covtype	63.022 ± 0.0008	63.022 ± 0.0008
	HIGGS	55.381 ± 0.0007	55.380 ± 0.0007
	RCV1	92.233 ± 0.00010	92.233 ± 0.00010
Delete (0.005%)	MNIST	86.272 ± 0.0035	86.272 ± 0.0035
	MNIST^n	92.203 ± 0.004	92.203 ± 0.004
	covtype	62.966 ± 0.0017	62.966 ± 0.0017
	HIGGS	52.950 ± 0.0001	52.950 ± 0.0001
	RCV1	92.241 ± 0.00004	92.241 ± 0.00004
Delete (1%)	MNIST	86.082 ± 0.0046	86.074 ± 0.0048
	MNIST^n	92.373 ± 0.003	92.370 ± 0.003
	covtype	62.943 ± 0.0007	62.943 ± 0.0007
	HIGGS	52.975 ± 0.0002	52.975 ± 0.0002
	RCV1	92.203 ± 0.00007	92.203 ± 0.00007

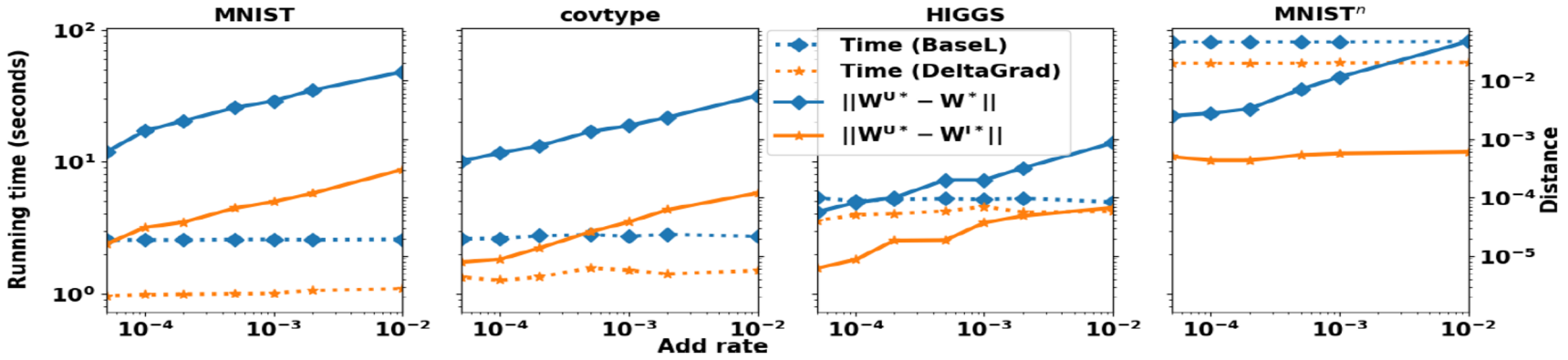


Figure 2. Running time and distance with varied add rate

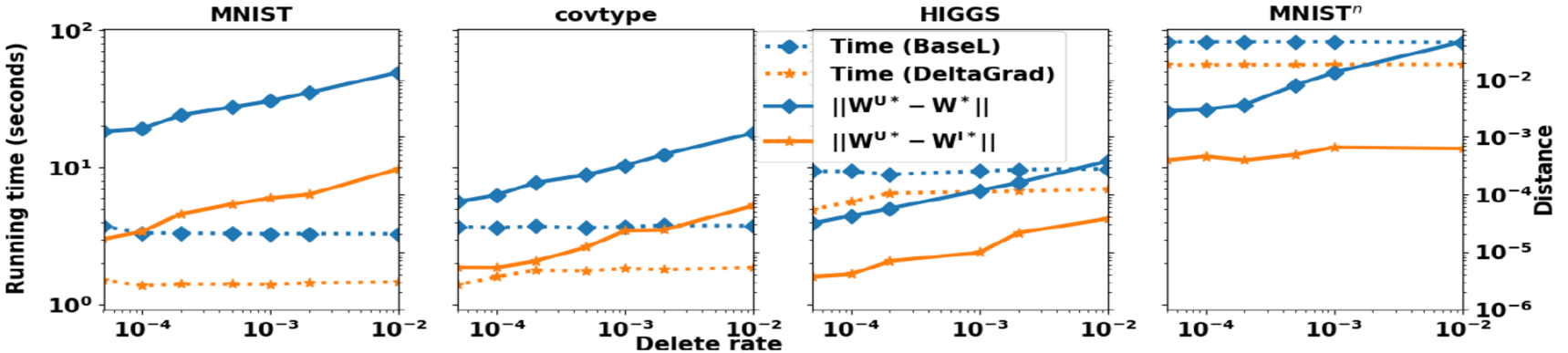


Figure 3. Running time and distance with varied delete rate

Table 2. Distance and prediction performance of BaseL and DeltaGrad in online deletion/addition

Dataset	Distance		Prediction accuracy (%)	
	$\|\mathbf{w}^{U*} - \mathbf{w}^*\|$	$\|\mathbf{w}^{I*} - \mathbf{w}^{U*}\|$	BaseL	DeltaGrad
MNIST (Addition)	5.7×10^{-3}	2×10^{-4}	87.548 ± 0.0002	87.548 ± 0.0002
MNIST (Deletion)	5.0×10^{-3}	1.4×10^{-4}	87.465 ± 0.002	87.465 ± 0.002
covtype (Addition)	8.0×10^{-3}	2.0×10^{-5}	63.054 ± 0.0007	63.054 ± 0.0007
covtype (Deletion)	7.0×10^{-3}	2.0×10^{-5}	62.836 ± 0.0002	62.836 ± 0.0002
HIGGS (Addition)	2.1×10^{-5}	1.4×10^{-6}	55.303 ± 0.0003	55.303 ± 0.0003
HIGGS (Deletion)	2.5×10^{-5}	1.7×10^{-6}	55.333 ± 0.0008	55.333 ± 0.0008
RCV1 (Addition)	0.0122	3.6×10^{-6}	92.255 ± 0.0003	92.255 ± 0.0003
RCV1 (Deletion)	0.0119	3.5×10^{-6}	92.229 ± 0.0006	92.229 ± 0.0006

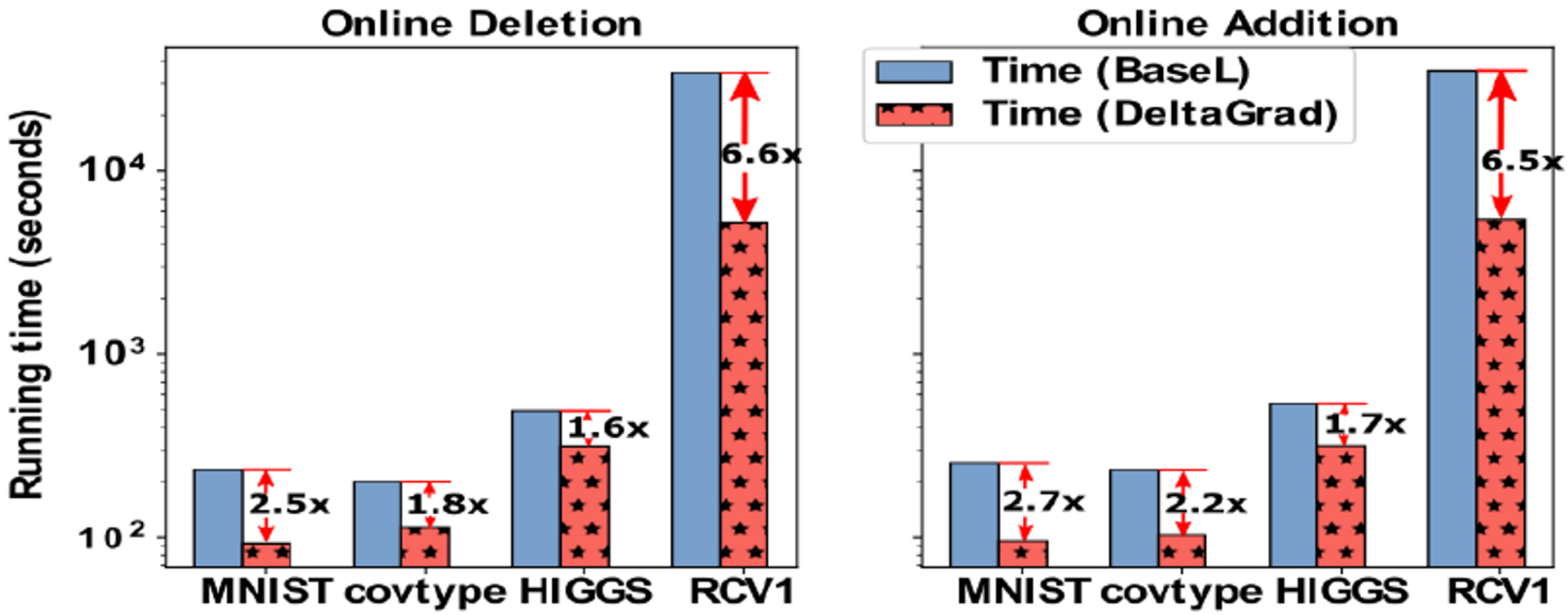


Figure 4. Running time comparison of BaseL and DeltaGrad with 100 continuous deletions/addition

Applications

Privacy related data deletion

Continuous model updating

Robustness

Data valuation

Bias reduction

Uncertainty quantification / predictive inference

Future Work

Smaller batch sizes in stochastic gradient descent

Models without strong convexity and smoothness

Comparison and Discussion

Comparison of solutions.

1. Algorithmic:
    a. Approximates the model and has a bounded error.
    b. Relies on model properties or math.

1. Systemic:
    a. Take an declarative model and change how it is run (i.e schedule)
    b. Relies on system properties or atleast system+model properties.
    c. Will give exactly the same solution as an unoptimized version.


ANY
QUESTIONS?