CS 260D: Large-scale Machine Learning

Lecture #14: Review

Baharan Mirzasoleiman

UCLA Computer Science

Midterm Review

Midterm Exam

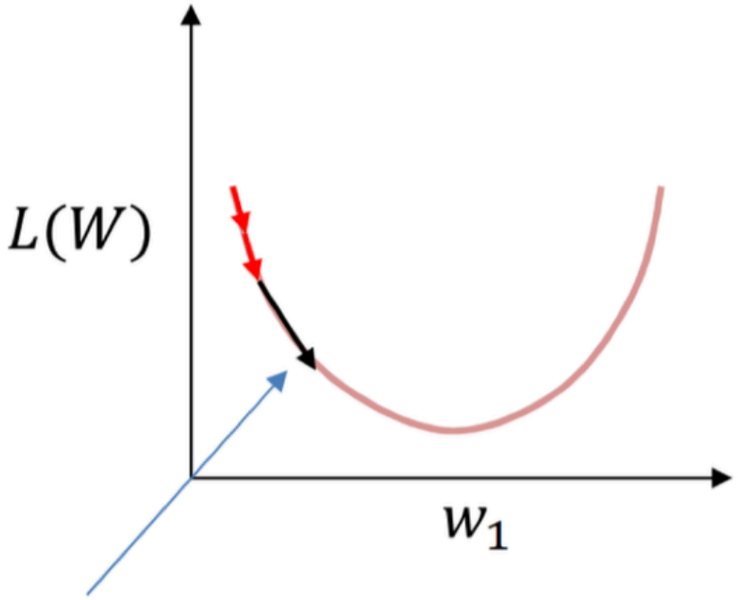
- Date: Tue Nov 18
- Exam is not open book and you won't need cheat sheet(s) or calculator
- For Lec 1/91: in the class
- For Lec 80: we will send instructions

Topics

- Momentum & adaptive learning rate, convergence of (mini-batch) SGD
- Parallel and distributed SGD & Federated learning
- Submodularity: definition, greedy, distributed & streaming methods
- Data selection for ML: efficiency & performance, robustness to noisy labels, data poisoning attacks, and spurious correlations
- Neural network pruning
- Midterm will be similar style to HWs

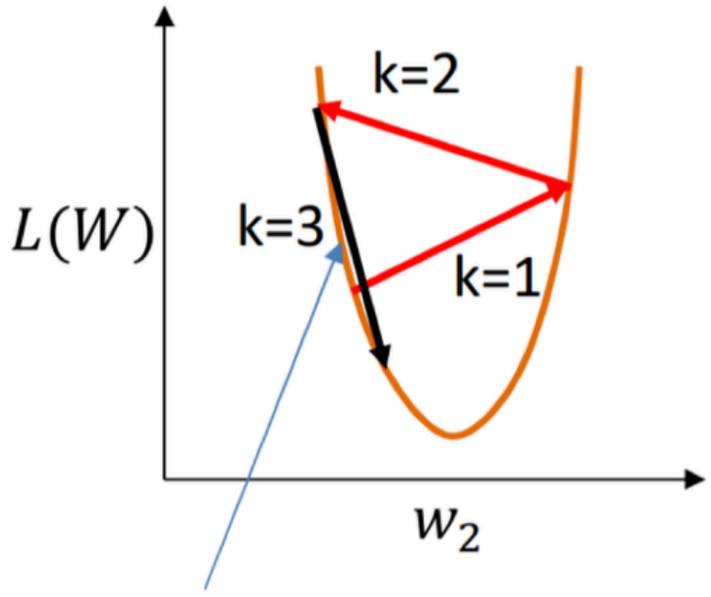
Variance & Variance Reduction

Momentum

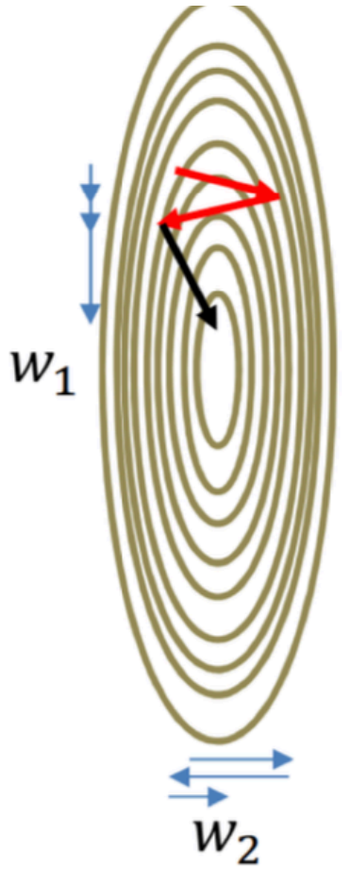


Increase stepsize because previous updates consistently moved weight right

$$W_k = W_{k-1} - \eta \nabla_w L(W)^T$$



Decrease stepsize because previous updates kept changing direction

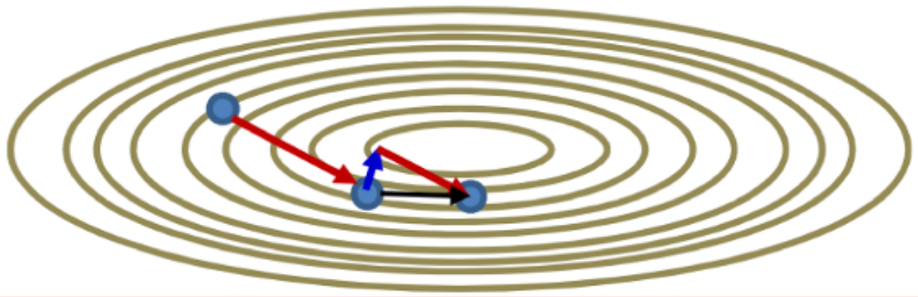


Stepsize shrinks along w2 but increases along w1

- Ideally: Have component-specific step size
  - Too many independent parameters (maintain a step size for every weight/bias)
- Adaptive solution: Start with a common step size
  - *Shrink* step size in directions where the weight oscillates
  - *Expand* step size in directions where the weight moves consistently in one direction

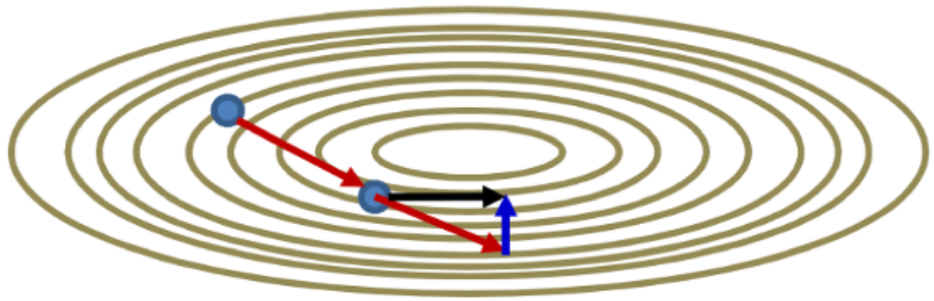
Momentum

Momentum



$$\Delta W^{(k)} = \beta \Delta W^{(k-1)} - \eta \nabla_W Err\left(W^{(k-1)}\right)$$

Nestorov



$$W^{(k)}_{extend} = W^{(k-1)} + \beta \Delta W^{(k-1)}$$

$$\Delta W^{(k)} = \beta \Delta W^{(k-1)} - \eta \nabla_W Err\left(W^{(k)}_{extend}\right)$$

$$W^{(k)} = W^{(k-1)} + \Delta W^{(k)}$$

- Momentum: Retain gradient value, but *smooth out* gradients by maintaining a running average
  - Cancels out steps in directions where the weight value oscillates
  - Adaptively increases step size in directions of consistent change

Adaptive learning rate

Idea: Increase/decrease the learning rate along dimensions with lower/higher curvature

Adam

- Estimate first moment:

$$v_i = \beta_1 v_{i-1} + (1-\beta_1) g_i$$

- Estimate second moment:

RMSProp

Adagrad

$$r_i = \beta_2 r_{i-1} + (1-\beta_2) g_i^2$$

Hessian estimate

- Update parameters:

$$W_i = W_{i-1} - \frac{\eta}{\delta + \sqrt{r_i}} v_i$$

Works well in practice,
is fairly robust to
hyper-parameters

(Mini-batch) SGD Convergence Rate

- Typically we make a (uniform) random choice $i_k \in \{1, \ldots n\}$
- Also common: mini-batch stochastic gradient descent, where we choose a random subset $I_k \subset \{1, \ldots n\}$, of size $b \ll n$, and update according to

$$x^{(k)} = x^{(k-1)} - t_k \cdot \frac{1}{b} \sum_{i \in I_k} \nabla f_i(x^{(k-1)}), \quad k = 1, 2, 3, \ldots$$

- In both cases, we are approximating the full graident by a noisy estimate, and our noisy estimate is unbiased

$$\mathbb{E}[\nabla f_{i_k}(x)] = \nabla f(x)$$

$$\mathbb{E}\left[\frac{1}{b} \sum_{i \in I_k} \nabla f_i(x)\right] = \nabla f(x)$$

The mini-batch reduces the variance by a factor $1/b$, but is also b times more expensive!

In general, smaller gradient variance ⟶ faster convergence rate for (mini-batch) SGD

(Mini-batch) SGD Convergence Rate

Recall that, under suitable step sizes, when f is convex and has a Lipschitz gradient, full gradient (FG) descent satisfies

$$f(x^{(k)}) - f^\star = O(1/k)$$

Faster convergence rate
But each iteration slower in time

What about stochastic gradient (SG) descent? Under diminishing step sizes, when f is convex (plus other conditions)

$$\mathbb{E}[f(x^{(k)})] - f^\star = O(1/\sqrt{k})$$

Slower convergence rate
Each iteration faster in time

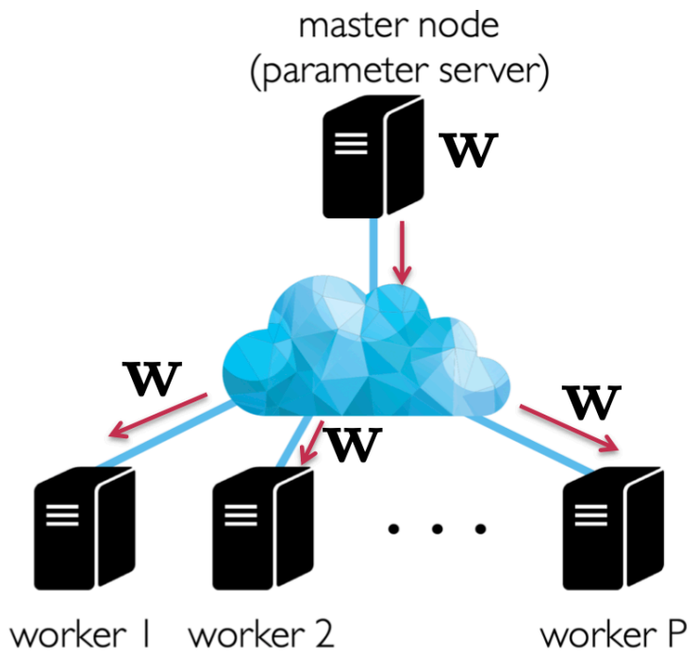
Finally, what about mini-batch stochastic gradient? Again, under diminishing step sizes, for f convex (plus other conditions)

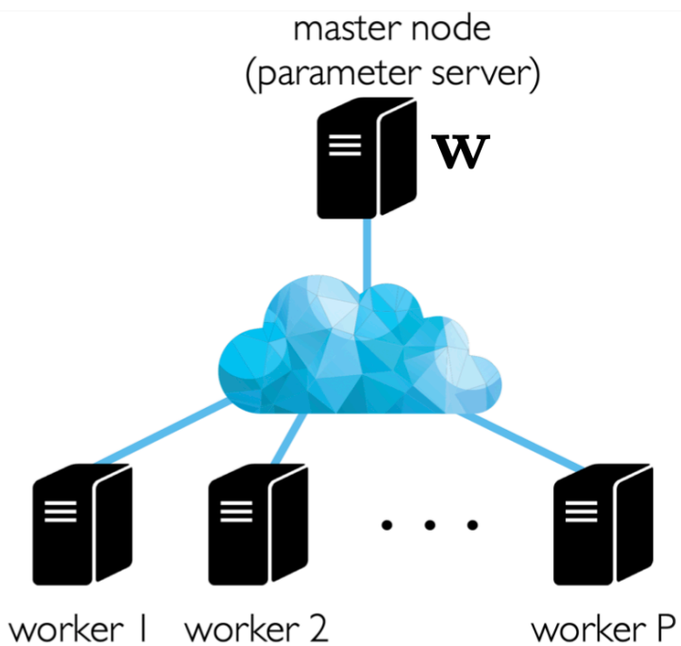
$$\mathbb{E}[f(x^{(k)})] - f^\star = O(1/\sqrt{bk} + 1/k)$$

But each iteration here b times more expensive ... and (for small b), in terms of flops, this is the same rate

Distributed Learning

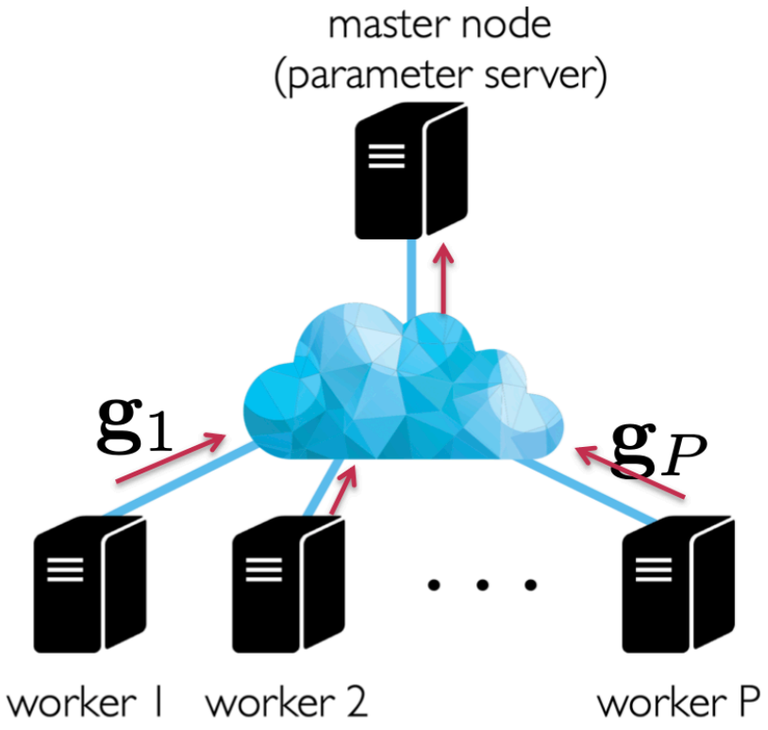
Distributed mini-batch SGD (Synchronous)

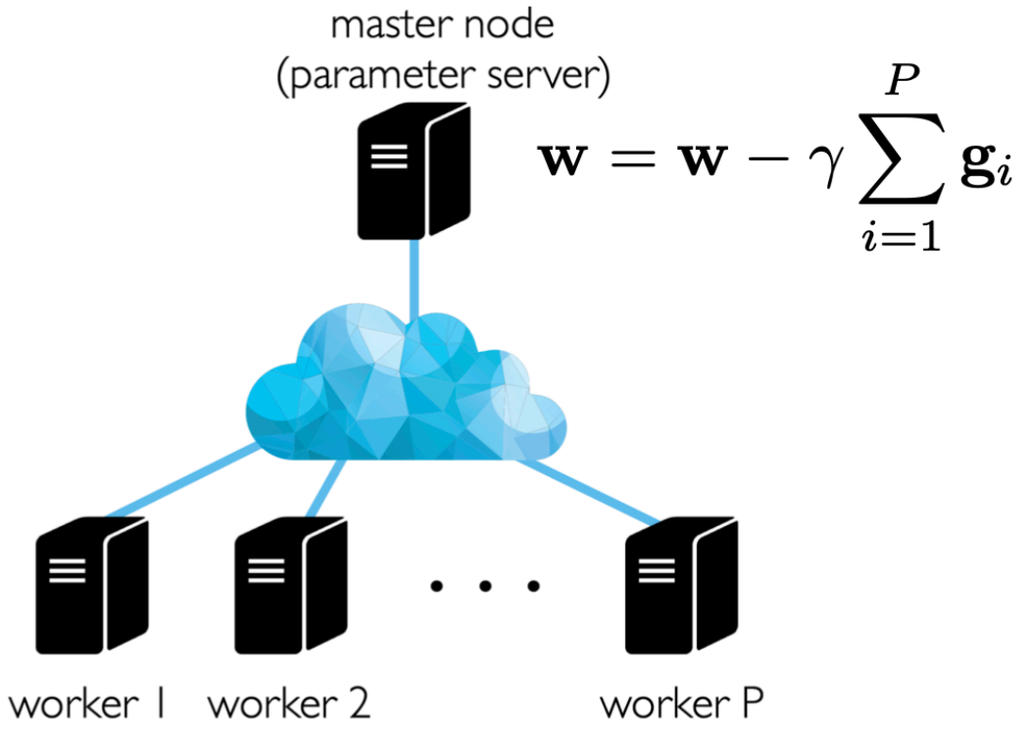




$$\mathbf{g}_1 = \sum_{i \in \mathcal{S}_1} \nabla \ell(\,(\mathbf{x}_i, y_i); \mathbf{w}\,)$$

$$\mathbf{g}_P = \sum_{i \in \mathcal{S}_P} \nabla \ell(\,(\mathbf{x}_i, y_i); \mathbf{w}\,)$$





Effect of mini-batch size on distributed SGD

Communication slows down the training

Batch	Top-1 Acc	Top-5 Acc
256	58.42%	81.51%
512	59.19%	81.84%
1024	59.00%	81.94%
2048	58.88%	81.73%
4096	57.97%	81.00%
8192	55.90%	79.40%

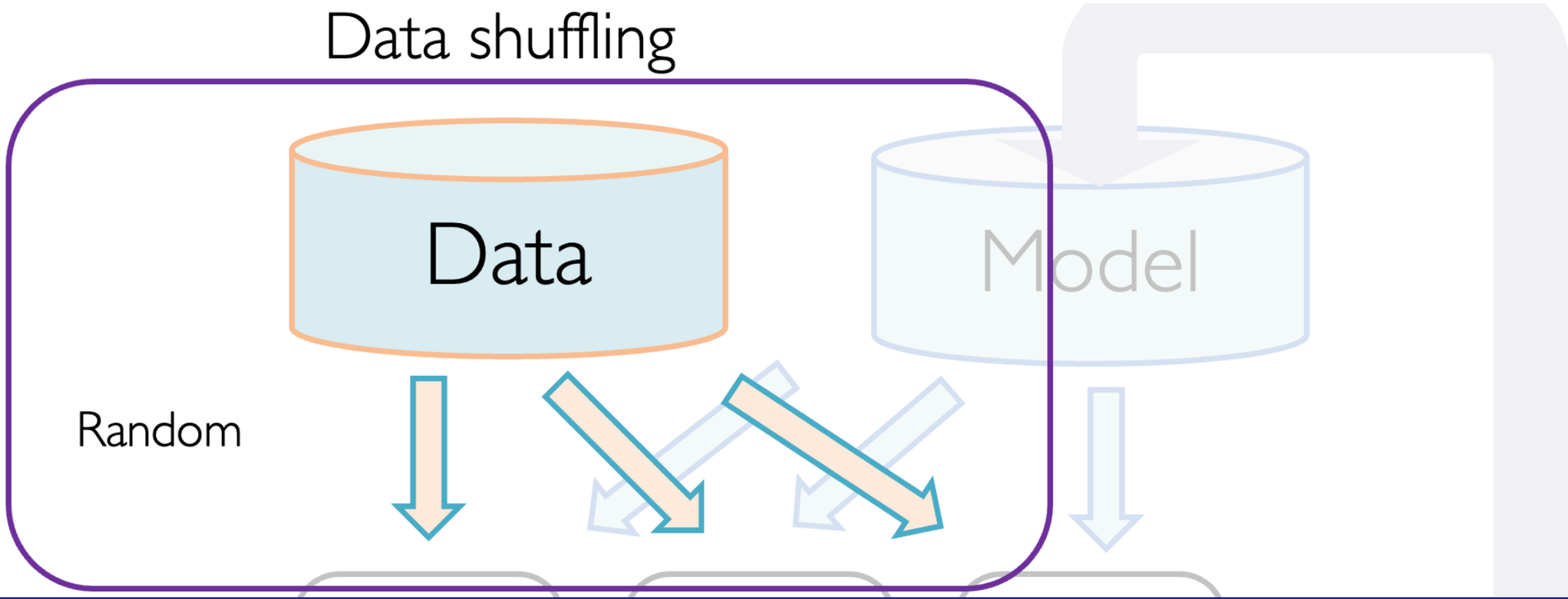
Alexnet on Imagenet

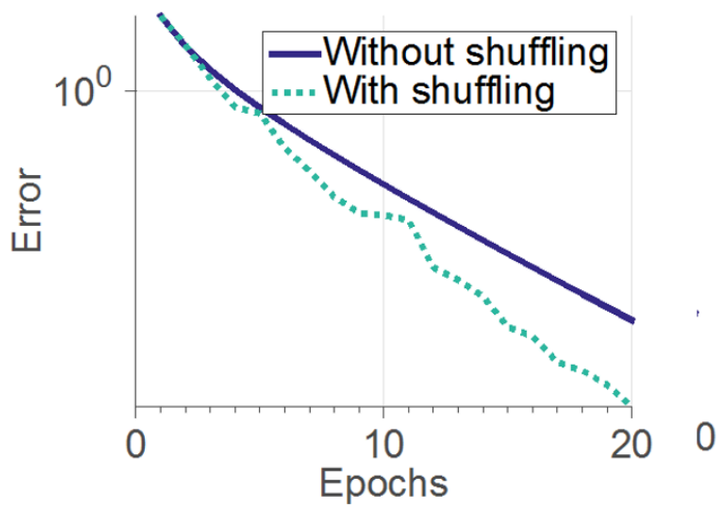
Bigger Batch
⇒ Less Communication
(smaller time per epoch)

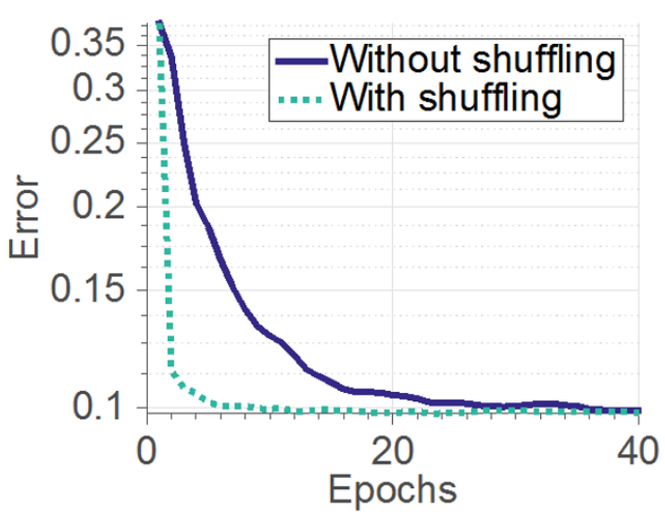
Large Batch
⇒ worse train error
(more #passes to accuracy **ε**)

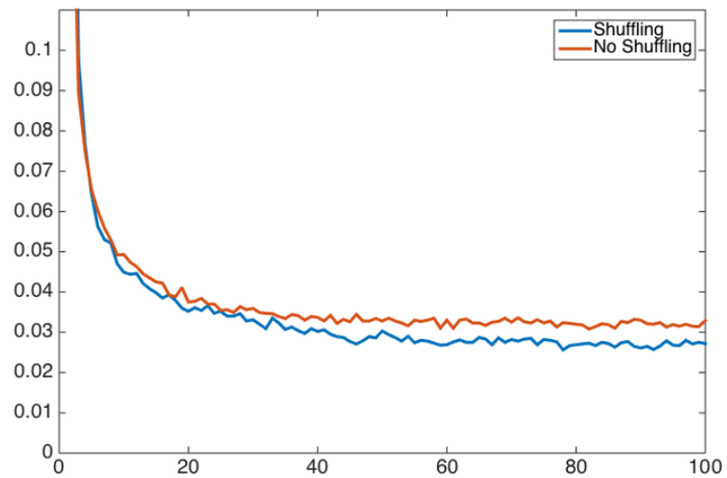
Large Batch
=> worse generalization

[Keskar, Mudigere, Nocedal, Smelyanskiy, Tang, ICLR 2017]





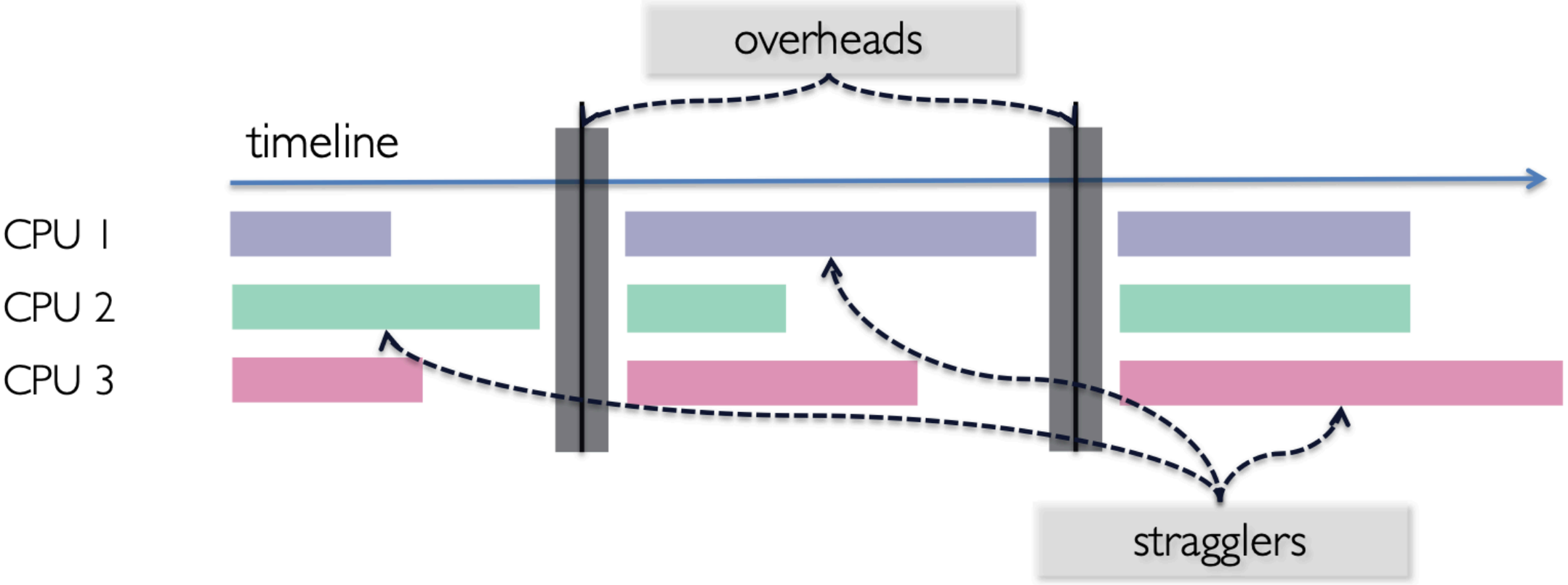




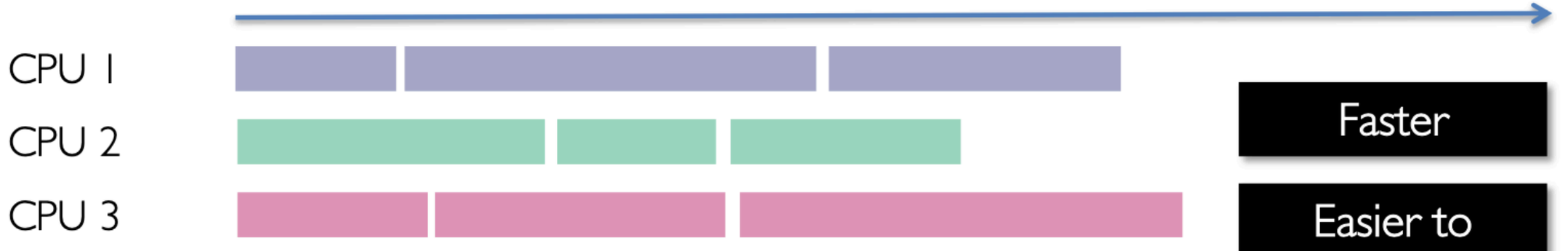
PSGD with shuffling converges faster

* [Recht and Re, 2013], [Bottou, 2012], [Zhang and Re, 2014], [Gurbuzbalaban et al., 2015], [Ioffe and Szegedy, 2015], [Zhang et al. 2015]

Async Parallel SGD



Asynchronous World



Faster

Easier to Implement

asynchrony noise affects rates, but if bounded, not by much

Hogwild! Achieves linear speedups

When/Why Hogwild Works?

Hogwild is equivalent to a noisy serial SGD

asynchrony noise affects rates, but if bounded, not by much

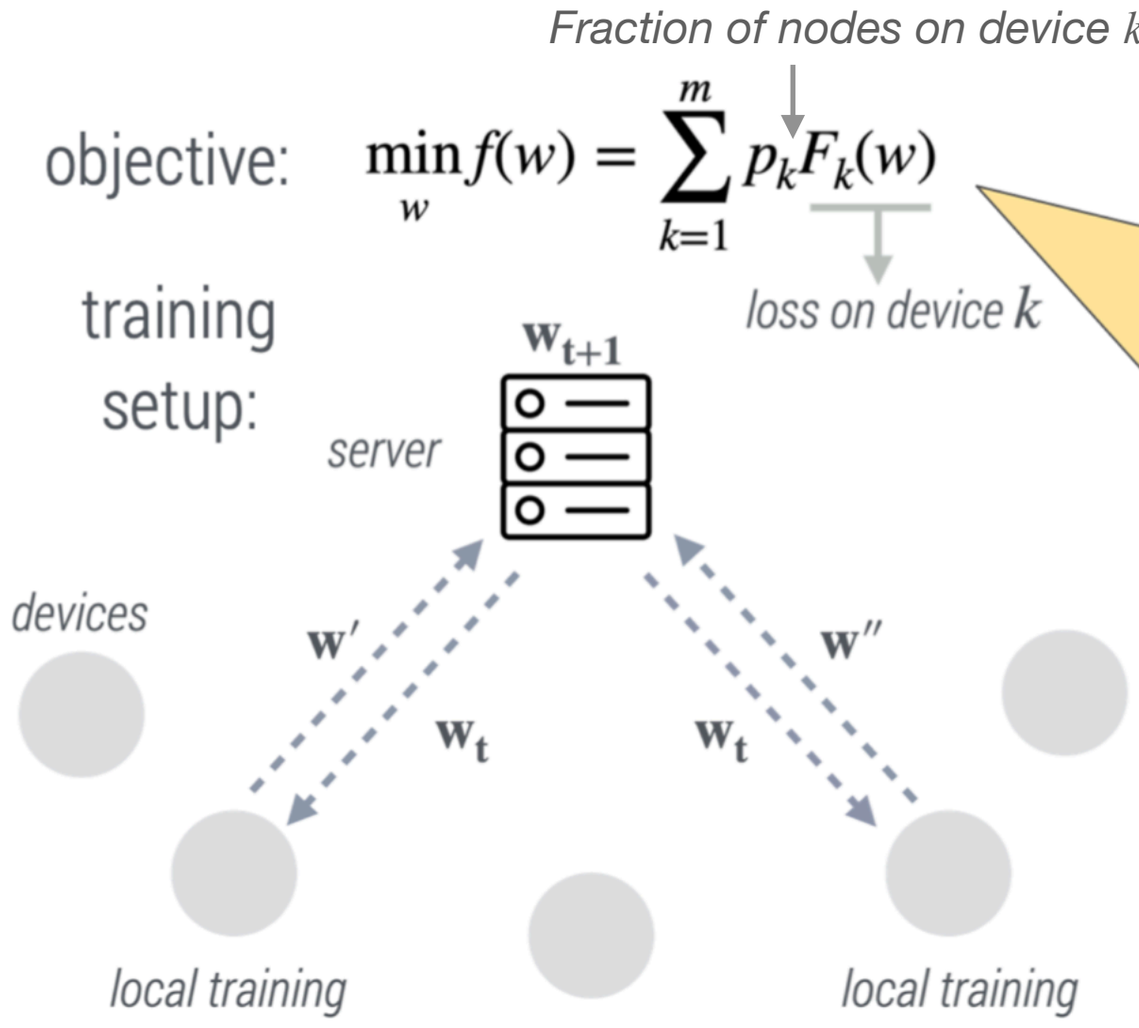
When core delay is less than $\tau \leq \frac{n}{2\Delta_{\text{av}}}$, noise does not affect convergence

Hogwild! Achieves linear speedups

*=in terms of worst case convergence

Federated Learning

Federated Learning



Typically consider solving an ERM objective, which is a (possibly) weighted average of losses across the *m* devices and their local data, i.e.,

$$\min_w \sum_{k=1}^{m} p_k \sum_{i=1}^{n_k} \ell(h(x_k^{(i)}; w), y_k^{(i)})$$

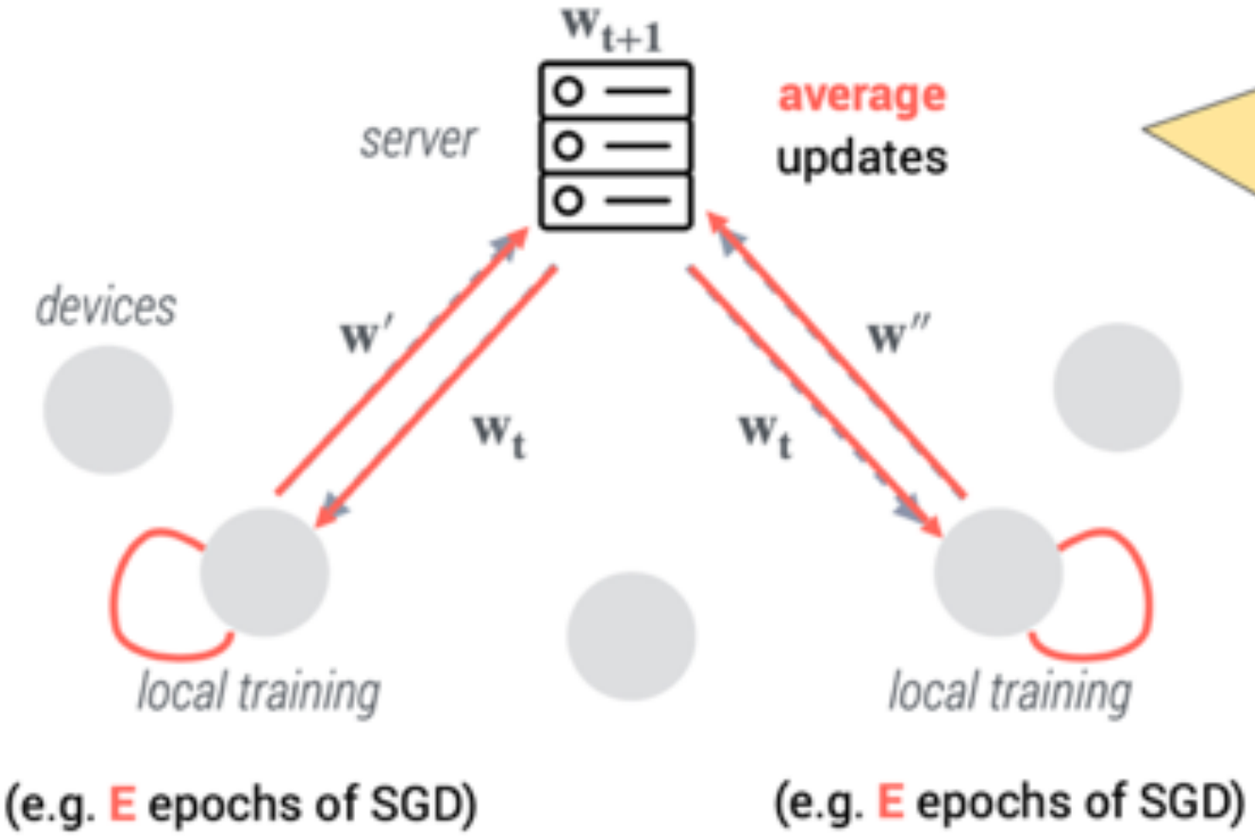
However, challenges discussed translate to other common ML objectives as well

- What if devices have different number of data points?

Federated Learning

A STANDARD BASELINE

Federated Averaging (FedAvg)



- At each communication round:
  (i) run SGD locally, then
  (ii) average the model updates
- Can add privacy mechanisms to procedure (more later …)
- Reduces communication by:
  (i) performing local updating,
  (ii) communicating with a subset of devices

Distributed/Parallel SGD vs Federated Learning

Distributed SGD: computation on device k

for $i \in$ *mini-batch* B
  $\Delta \mathbf{w} \leftarrow \Delta \mathbf{w} - \alpha \nabla f_i(\mathbf{w})$
end
$\mathbf{w} \leftarrow \mathbf{w} + \Delta \mathbf{w}$

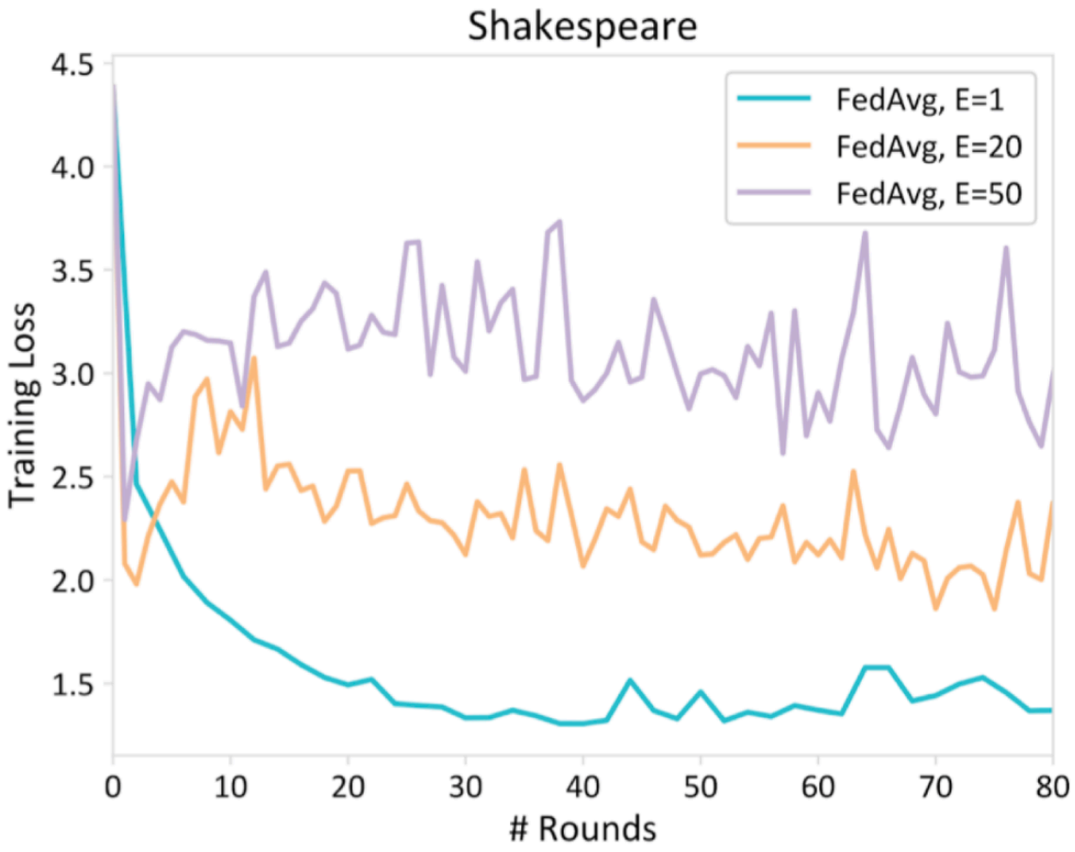
FedAvg: computation on device k

for $t = 1, 2, \ldots,$ *local iterations* T
  $\Delta \mathbf{w} \leftarrow \Delta \mathbf{w} - \alpha \nabla f_{i_t}(\mathbf{w})$
  $\mathbf{w} \leftarrow \mathbf{w} + \Delta \mathbf{w}$
end

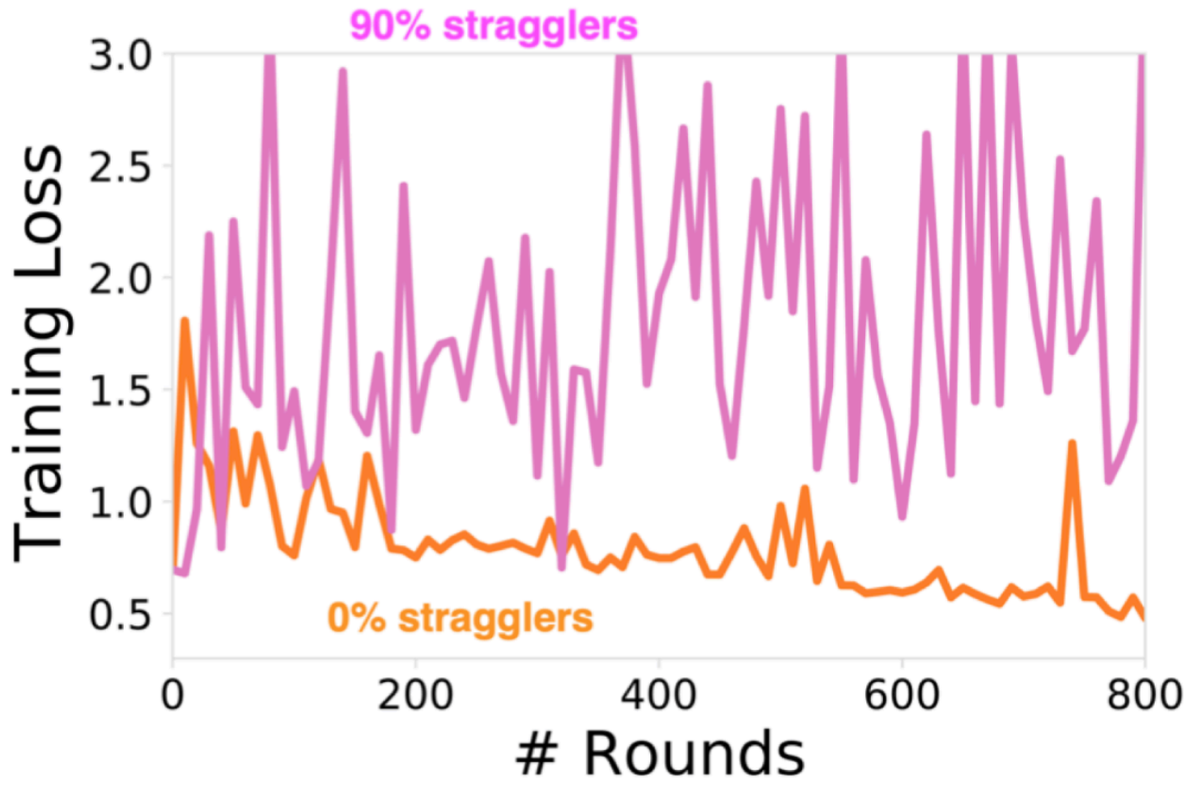
- We looked at the homogenous case in the homework
  - What if the nodes are not homogenous?
    - Stragglers
    - Different number of data points on each node
    - Slower convergence in the heterogeneous case

Federated Learning: heterogeneity

Data heterogeneity

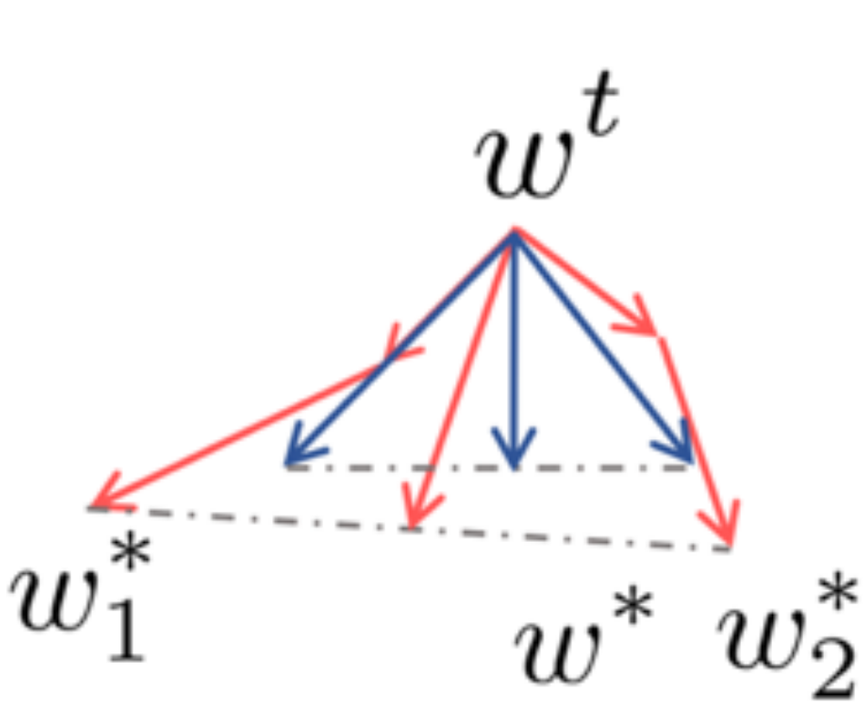


System heterogeneity



Federated Learning

Simple modification: FedProx



$$\min_{w_k} F_k(w_k) + \frac{\mu}{2} \left\| w_k - w^t \right\|^2$$

proximal term

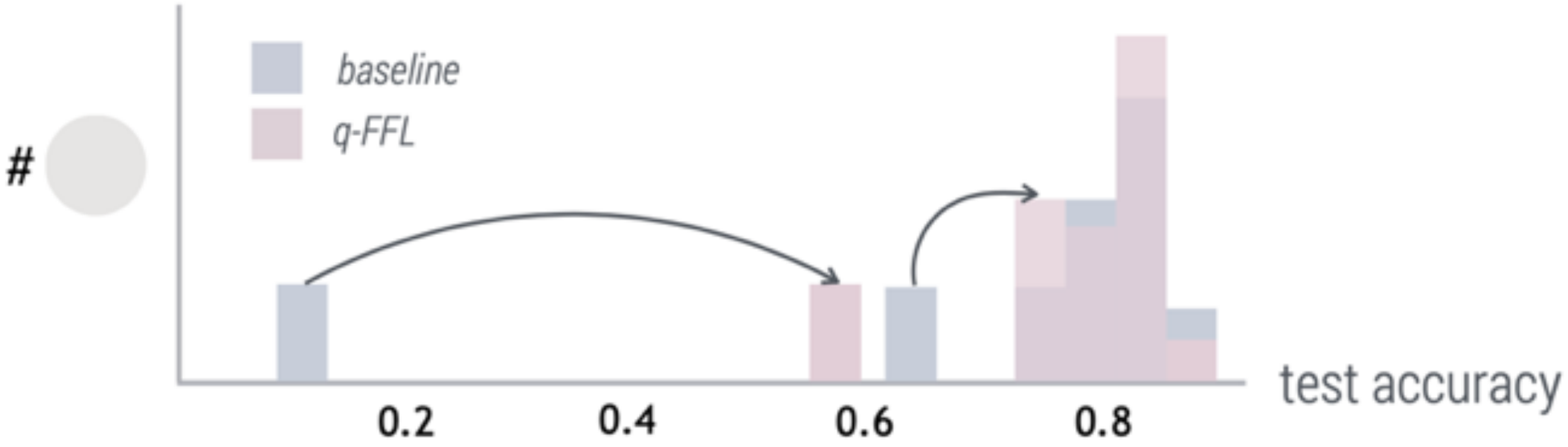
- proximal term *limits the impact of heterogeneous local updates*
- don't drop devices: instead [safely] incorporate partial work
- theoretical guarantees (with a dissimilarity assumption)

[Li et al, Federated optimization in heterogeneous networks, MLSys 2020]

Federated Learning

fair resource allocation objective

$$\textit{q-FFL:} \quad \min_{w} \frac{1}{q+1}\left(p_1 F_1^{q+1} + p_2 F_2^{q+1} + \cdots + p_N F_N^{q+1}\right)$$



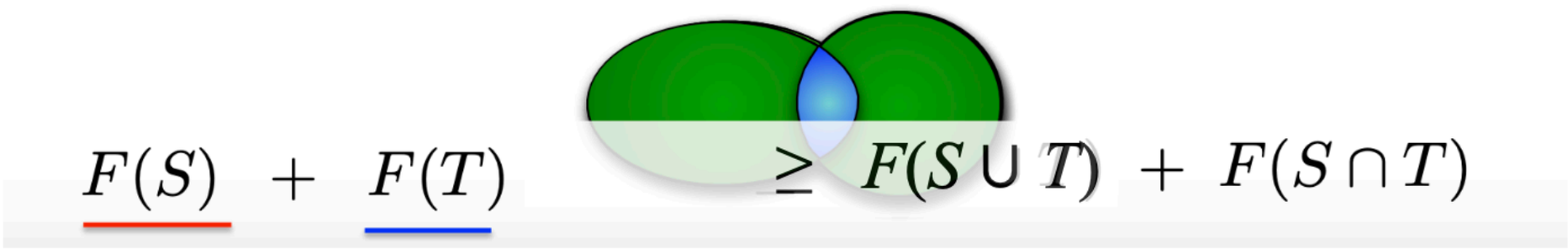
Submodular Maximization

Submodularity

- Diminishing gains: for all $A \subseteq B$

$$F(A \cup e) - F(A) \geq F(B \cup e) - F(B)$$

- Union-Intersection: for all $S, T \subseteq \mathcal{V}$



$$F(S) + F(T) \geq F(S \cup T) + F(S \cap T)$$

- Monotonicity: if $S \subseteq T$ then $F(S) \leq F(T)$

- how do you show a function is submodular? Similar to HW!

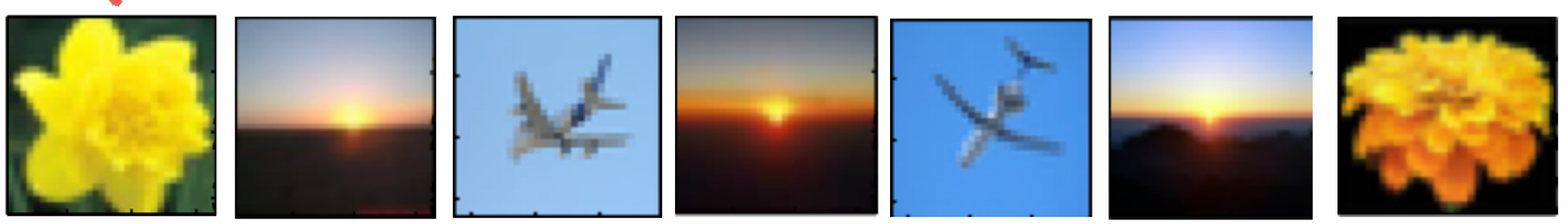
The Greedy Algorithm

Problem: Finding subset $S^* = \arg\max_{S \in V} F(S), \ |S| \leq k$

Submodular

How can we solve it?

Complexity: $\mathcal{O}(nk)$

$V = \{$  $\}$

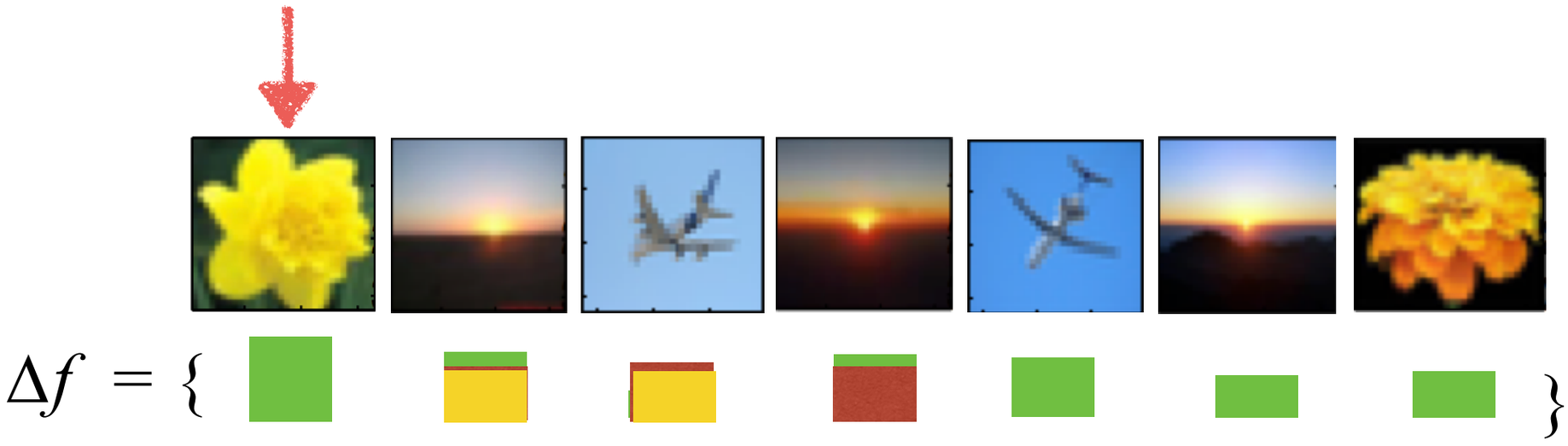
$F(e|S) = \{$  $\}$

$F\left(\{\quad\}\right) \longrightarrow \max$

Theorem [NWF '78]: For monotone submodular maximization under a cardinality constraint, greedy algorithm guarantees a $(1\text{-}1/e)$-approximation

Lazy Greedy Algorithm [Minoux'78]

- Using submodularity to improve the speed.



$$\Delta f = \{$$

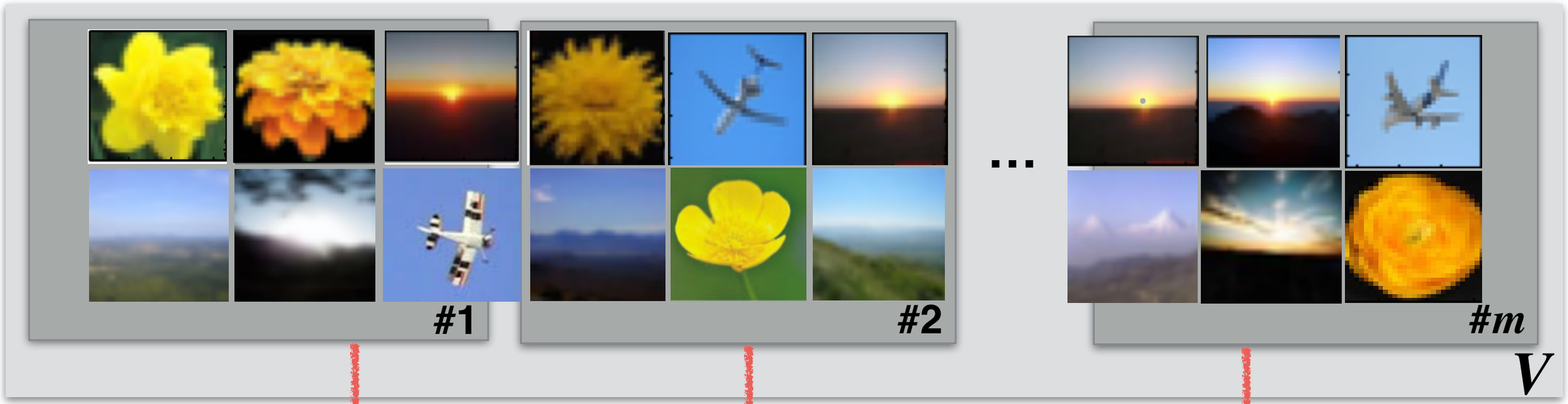
$$F(\{\quad\}) \longrightarrow \max$$

Lazy Greedy Algorithm

Submodularity can be exploited to implement an accelerated version of the classical greedy algorithm

Orders of magnitude speedups in practice, but still $\Omega(n.k)$ in the worst case.

Two-stage Greedy (GreeDi)



Theorem: $F(A) \geq \Omega\left(\frac{1}{\sqrt{\min(m,k)}}\right)\mathrm{OPT}$

Two-stage Greedy (GreeDi)

- How does the running time compare to greedy?
  - We need 1 round of communication without shuffling (2 rounds with shuffling)
  - n/m data points per machine
  - Greedy complexity: $\mathcal{O}(nk)$

Lazy updates: estimate *m* on the fly!

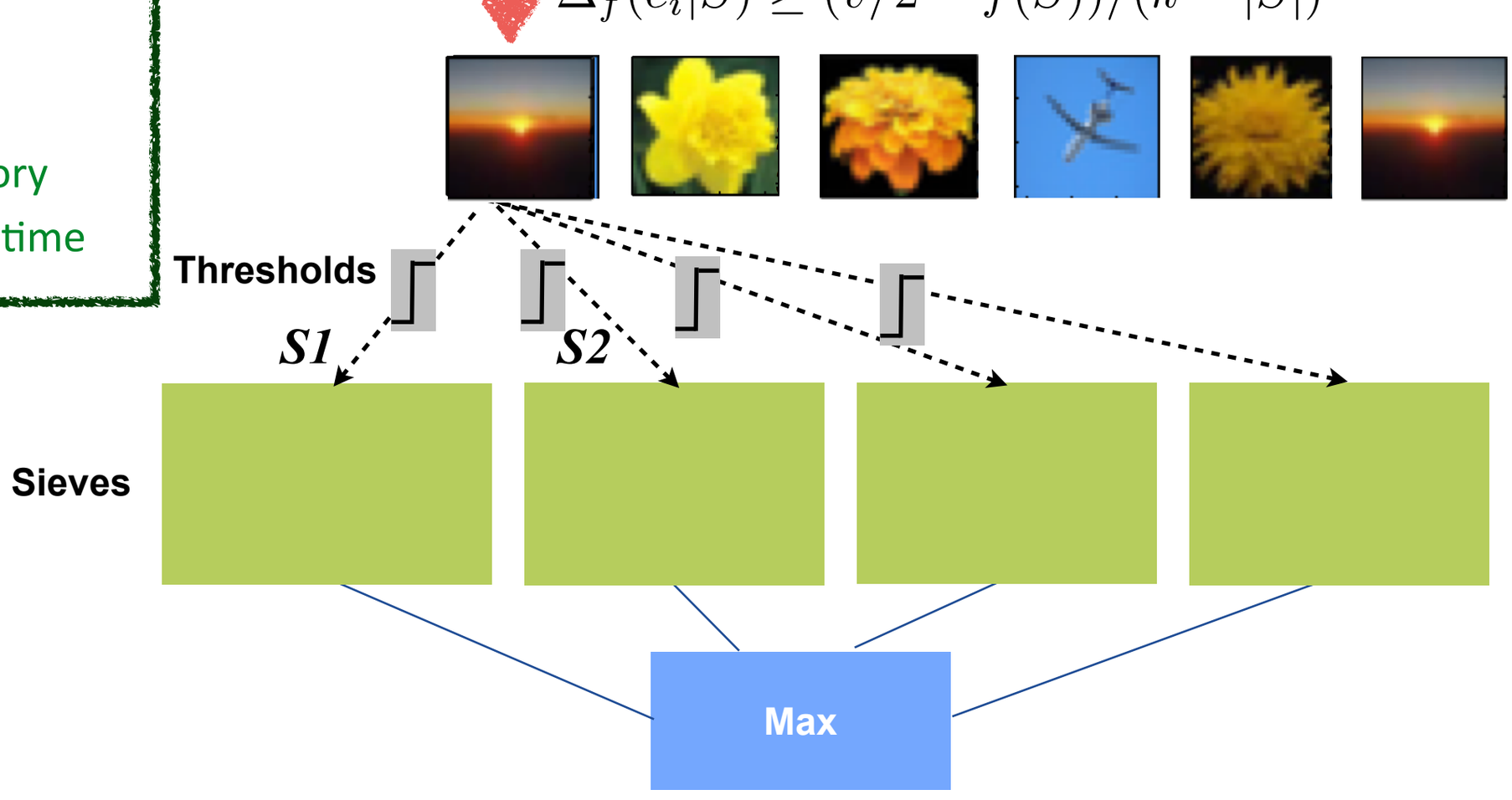
- Obtaining m requires a full pass over the data ☹
  - Hold the current maximum and lazily instantiate the threshold for an increased range

$$O = \{(1+\epsilon)^i \mid i \in Z, m \le (1+\epsilon)^i \le 2.k.m\}, \quad m=\max(m, f(\{e_i\}))$$

Properties:

- 1 pass
- $f(S) \geq (1/2\text{-}\epsilon)\mathtt{OPT}$
- $O(k \log k/\epsilon)$ memory
- $O(\log k/\epsilon)$ update time

$$\Delta_f(e_i|S) \geq (v/2 - f(S))/(k - |S|)$$



Data Selection for Machine Learning

Forgettability score

- Forgetting event: when a data point is misclassified after begin correctly classified

- Unforgettables: data points that have no forgetting event

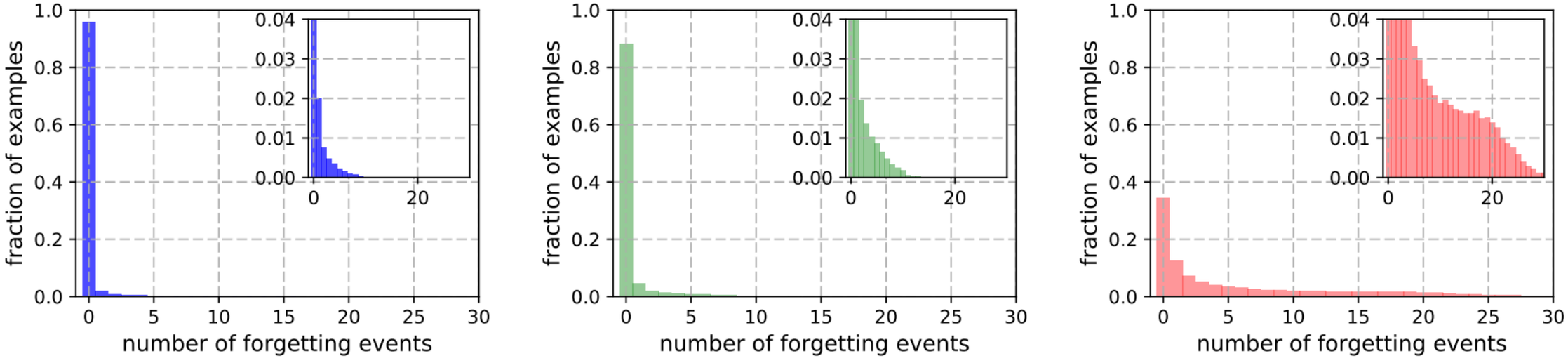
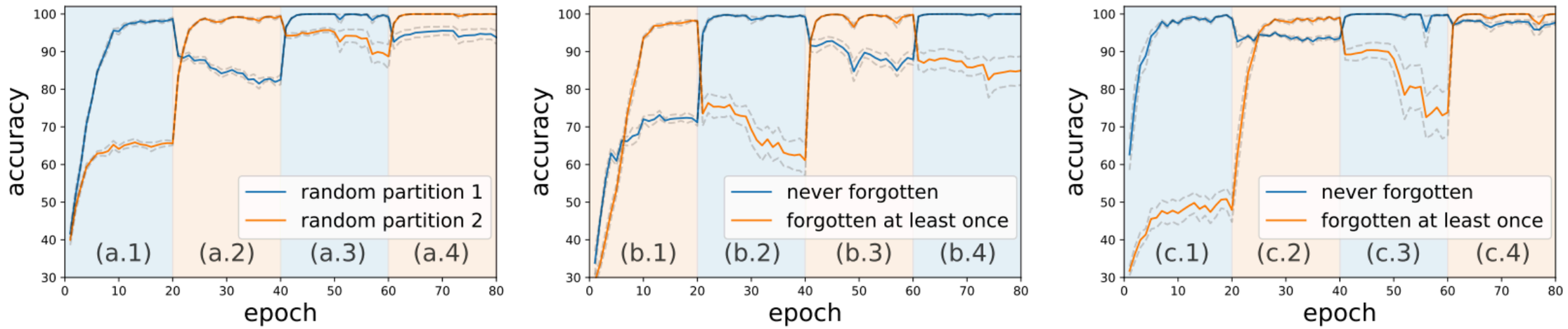


Figure 1: Histograms of forgetting events on (from left to right) *MNIST*, *permutedMNIST* and *CIFAR-10*. Insets show the zoomed-in y-axis.

Forgettable vs unforgettable examples

- Forgetting event: when a data point is misclassified after begin correctly classified
- Unforgettables: data points that have no forgetting event



(a) random partitions

(b) partitioning by forgetting events

Figure 4: Synthetic continual learning setup for *CIFAR-10*. Background color in each column indicates the training partition, curves track performance on both partitions during interleaved training. Solids lines represent the average of 5 runs and dashed lines represent the standard error. The figure highlights that examples that have been forgotten at least once can "support" those that have never been forgotten, as shown in (c.2) and (b.3).

How about noisy-labeled examples?

- Noisy-labeled examples are more forgettable

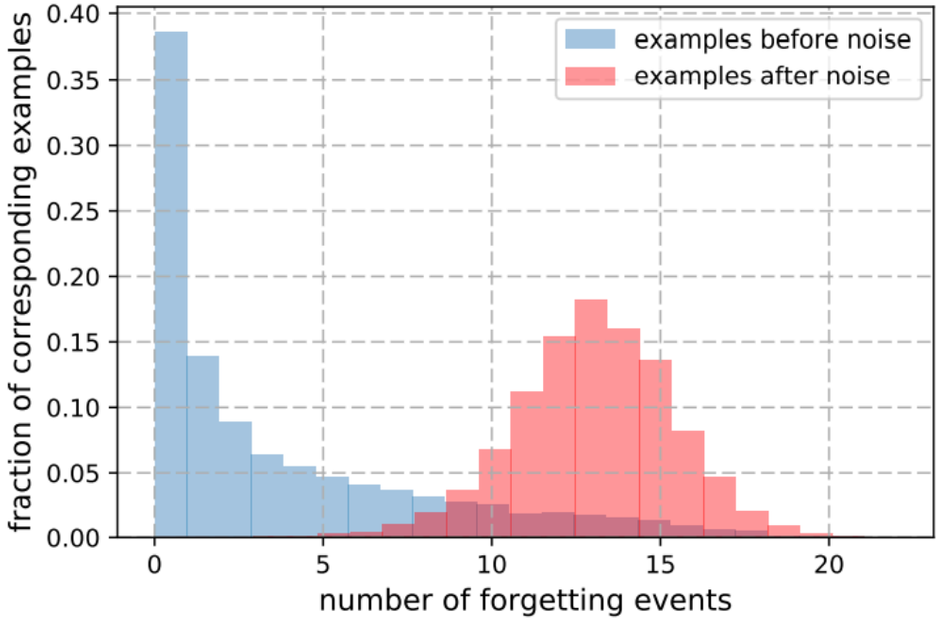


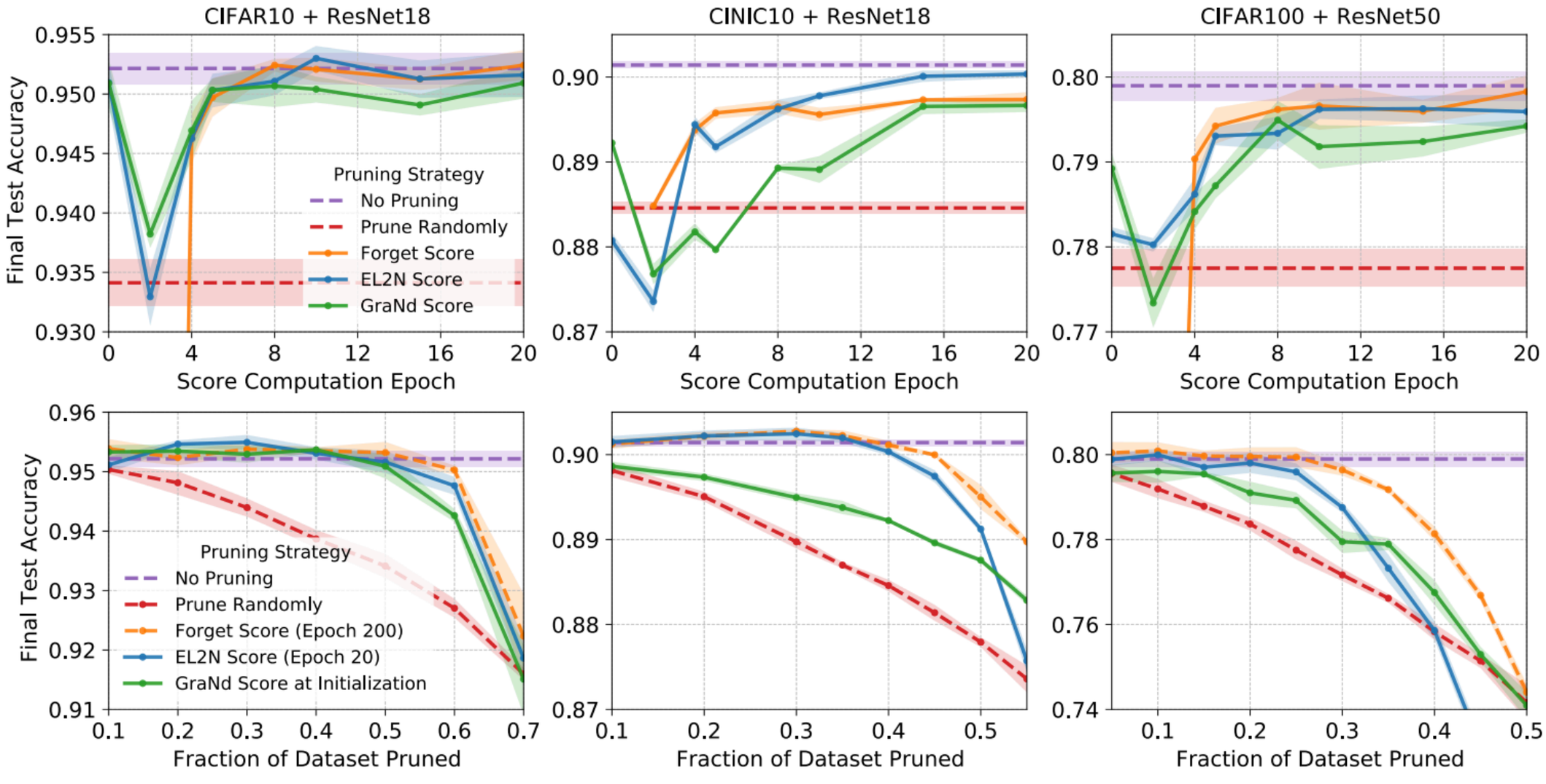
Figure 3: Distributions of forgetting events across training examples in *CIFAR-10* when 20% of labels are randomly changed. *Left.* Comparison of forgetting events between examples with noisy and original labels. The most forgotten examples are those with noisy labels. No noisy examples are unforgettable. *Right.* Comparison of forgetting events between examples with noisy labels and the same examples with original labels. Examples exhibit more forgetting when their labels are changed.

GraND and EL2N score

- GraNd score of a training example (x, y) at time t: $\mathbb{E}_{w_t}\|g_t(x,y)\|_2$

Gradient of loss w.r.t input to the last layer

- El2N score of a training example (x, y) at time t: $\mathbb{E}\|p(w_t,x)-y\|_2$



How about noisy-labeled examples?

- Noisy-labeled examples have higher EL2N score

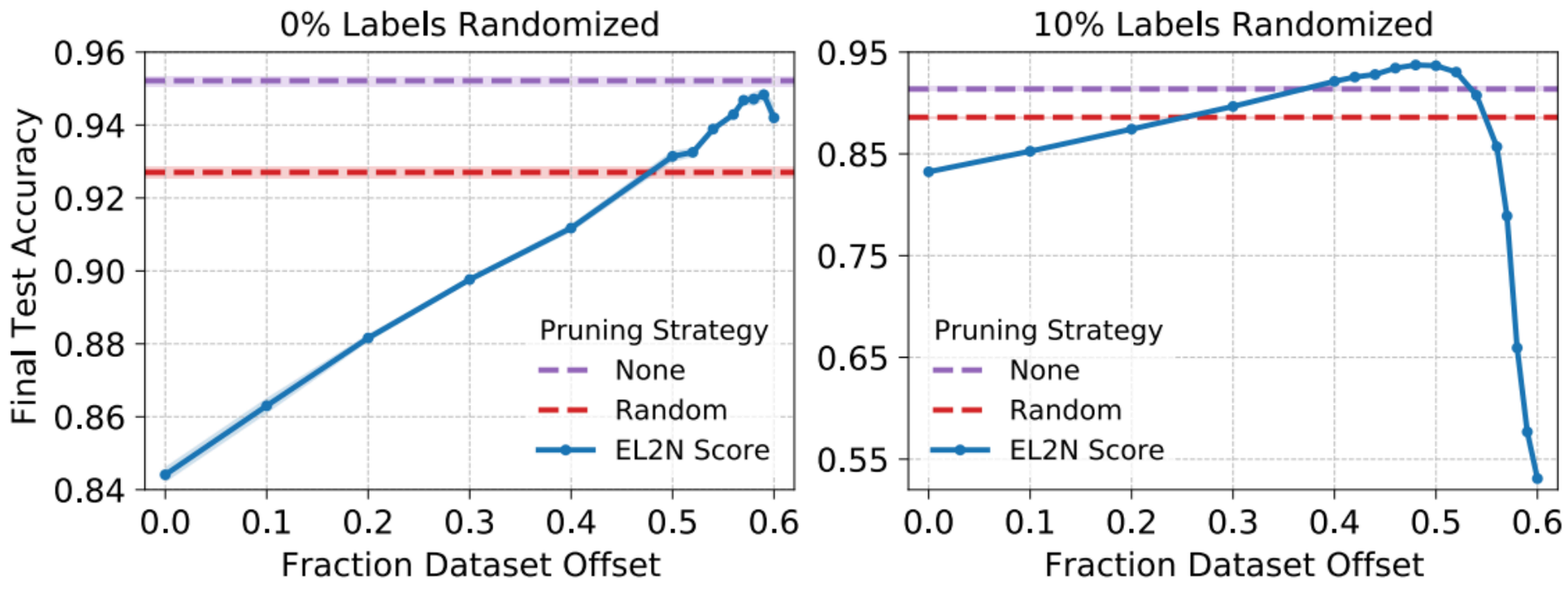
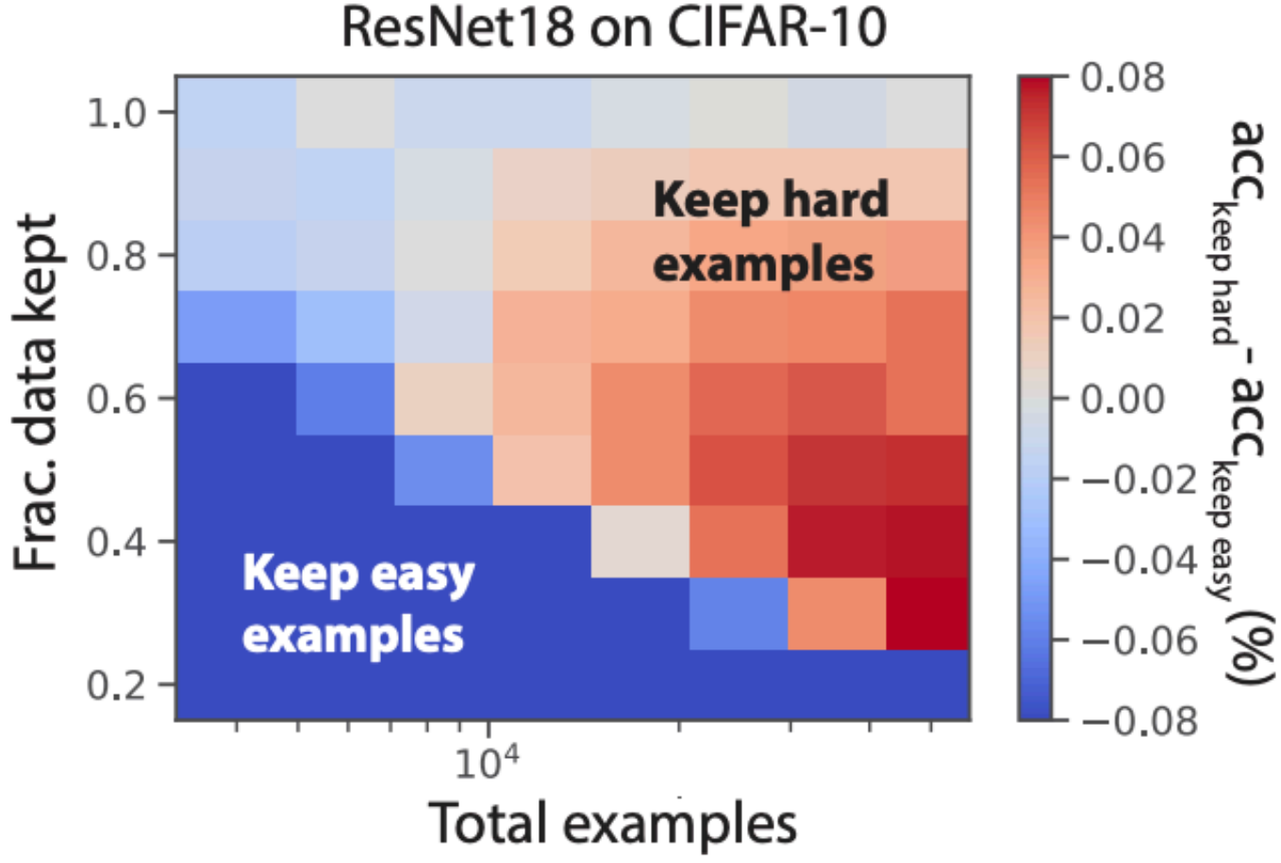


Figure 2: ResNet18 trained on a 40% subset of CIFAR-10 with clean *(left)* and 10% randomized labels *(right)*. The training subset contains the *lowest* scoring examples *after* examples with scores below the offset are discarded. Scores computed at epoch 10.

How much can we prune?

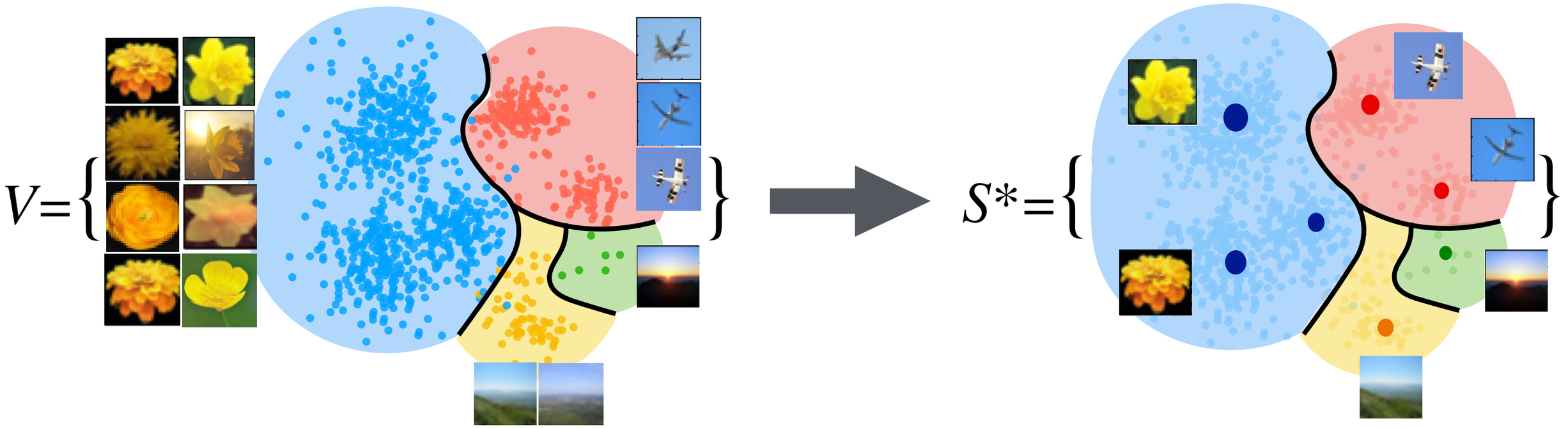
- If one does not have much data to start with, it is better to keep the easiest examples to avoid overfitting
- With abundant (scarce) initial data, one should retain only hard (easy) examples.



Data Summarization for Machine Learning

Problem: how to find the "right" data for machine learning?

- The most informative subset $S^* = \arg\max_{S \subseteq V} F(S), \quad \text{s.t.} \quad |S| \leq k$
- **What is a good choice for $F(S)$?**



- If we can find S^*, we get a $|V|/|S^*|$ speedup by only training on S^*

Data Summarization for Machine Learning

Idea: select the smallest subset S^* and weights γ that closely estimates the full gradient

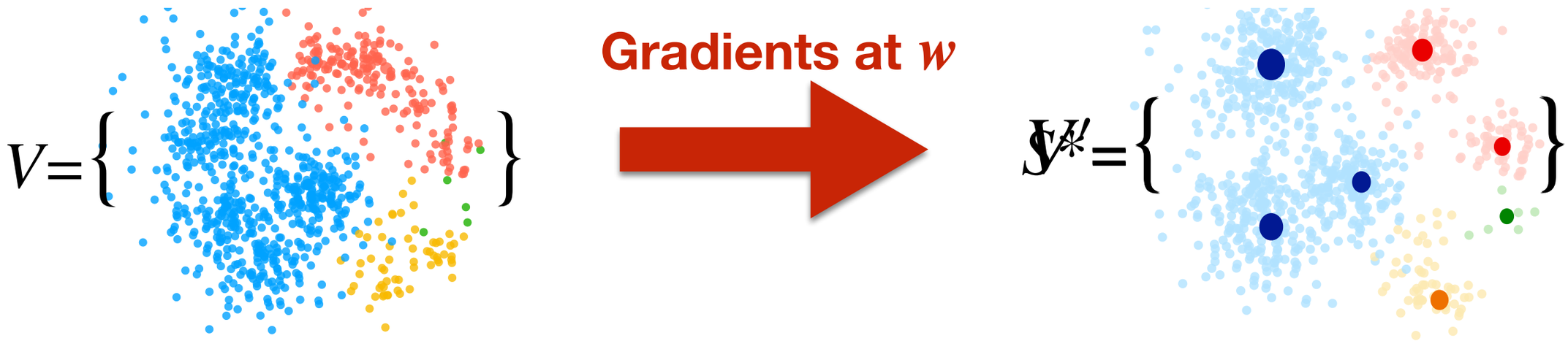
$$S^* = \arg\min_{S \subseteq V, \gamma_j \geq 0 \ \forall j} |S|, \quad \text{s.t.} \quad \max_{w \in \mathcal{W}} \| \sum_{i \in V} \nabla f_i(w) - \sum_{j \in S} \gamma_j \nabla f_j(w) \| \leq \epsilon .$$

Full gradient **Gradient of S**

Solution: for every $w \in \mathcal{W}$, S^* is the set of **exemplars** of all the data points in the **gradient space**

Training Data: $\{(x_i, y_i), i \in V\}$

$$V' = \{ \nabla f_i(w), i \in V\}$$



Learning from Summaries

Can we find a subset S^* that bounds the estimation error for all $w \in \mathscr{W}$?

$$F(S^*) = \sum_{i \in V} \min_{j \in S^*} \|\nabla f_i(w) - \nabla f_j(w)\| \leq \epsilon$$

Idea: consider worst-case approximation of the estimation error over the entire parameter space $\mathscr{W}$

$$F(S^*) = \sum_{i \in V} \min_{j \in S^*} \|\nabla f_i(w) - \nabla f_j(w)\| \leq \sum_{i \in V} \min_{j \in S^*} \underbrace{\max_{w \in \mathscr{W}} \|\nabla f_i(w) - \nabla f_j(w)\|}_{d_{ij}} \leq \epsilon$$

d_{ij}: upper-bound on the gradient difference over the entire parameter space $\mathscr{W}$

Our Approach: Learning from Coresets

How can we efficiently find upper-bounds d_{ij}?

- **Convex** $f(w)$: Linear/logistic/ridge regression, regularized SVM

$$d_{ij} \leq \text{const. } \|x_i - x_j\|$$

Feature vector

☑ S^* **can be found as a preprocessing step**

Before training:
For every class: (1) Find medoids of features, (2) train (backprop) only on the subset

- **Non-convex** $f(w)$: Neural networks

$$d_{ij} \leq \text{const. } (\|\nabla_{z_i^{(L)}} f_i(w) - \nabla_{z_j^{(L)}} f_j(w)\|) \qquad \text{[KF'19]}$$

Input to the last layer

☑ d_{ij} **is cheap to compute, but we have to update** S^*

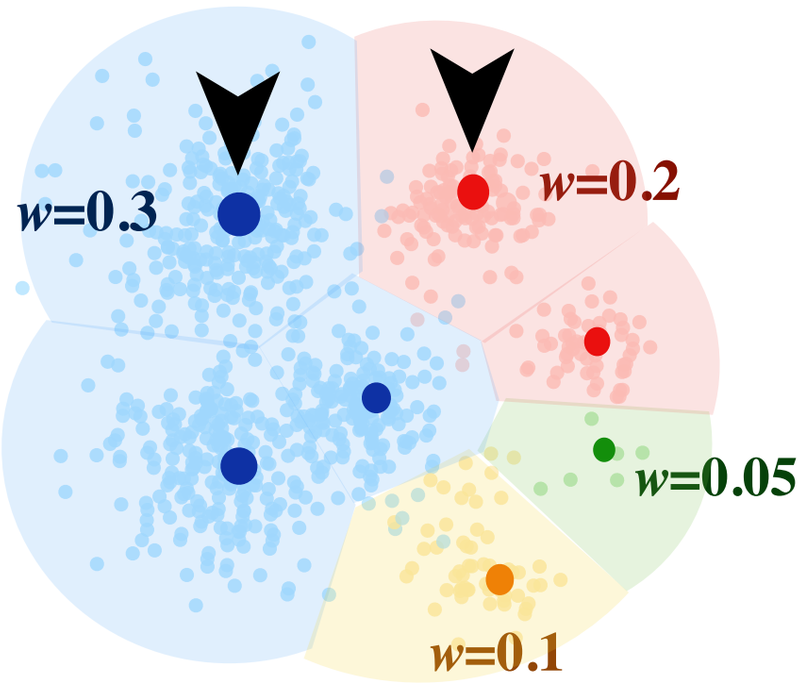
For classification tasks with softmax Cross Entropy (during the training):
For every class: (1) Find medoids of logits, (2) train (backprop) only on the subset

Our Approach: CRAIG

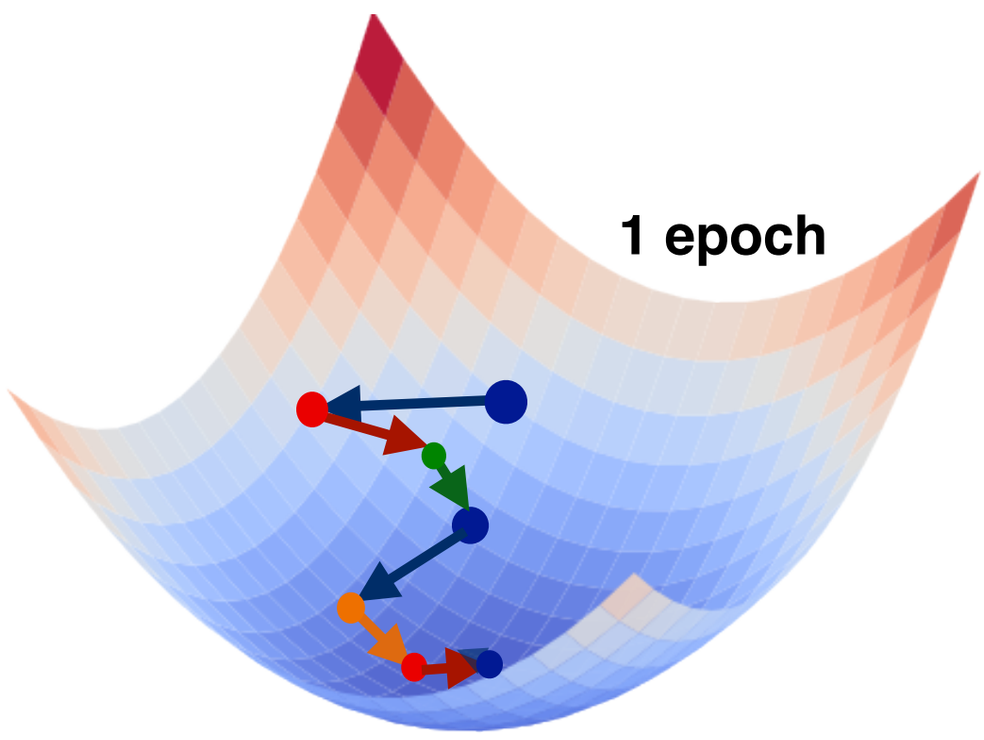
Idea: select a weighted subset that closely estimates the full gradient

Algorithm:

- **(1)** use **greedy** to find the set of exemplars S^* from dataset V
- **(2)** **weight** every elements of S^* by the size of the corresponding cluster
- **(3)** apply weighted incremental gradient descent on S^*



Gradients of data points $i \in V$



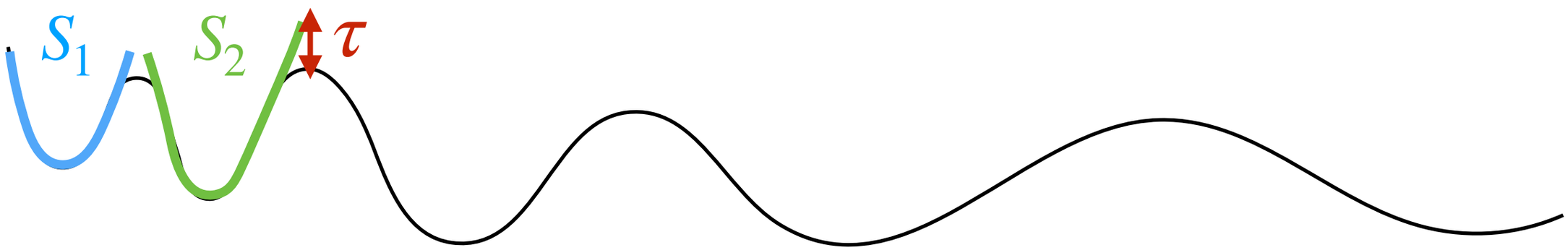
Loss function

Coresets for Data-efficient Training of Machine Learning Models, ICML 2020

Can we find subsets for training deep networks with theoretical guarantees?

Coresets for Data-efficient Deep Learning

- Can we find coresets for minimizing a non-convex loss?
  - **CREST**: **Find a corset for every convex region of the loss**
  - Train on the same coreset within every convex region



- More updates at the beginning and less updates towards the end of training

Towards Sustainable Learning: Coresets for Data-efficient Deep Learning (ICML'23)

Selecting Coresets with Submodular Maximization

- Early in training, the most effective subsets for learning deep models are easy-to-learn examples.
- As training proceeds, the model learns the most from examples with increasing levels of learning difficulty.
- Interestingly, the model never requires training on easiest-to-learn examples

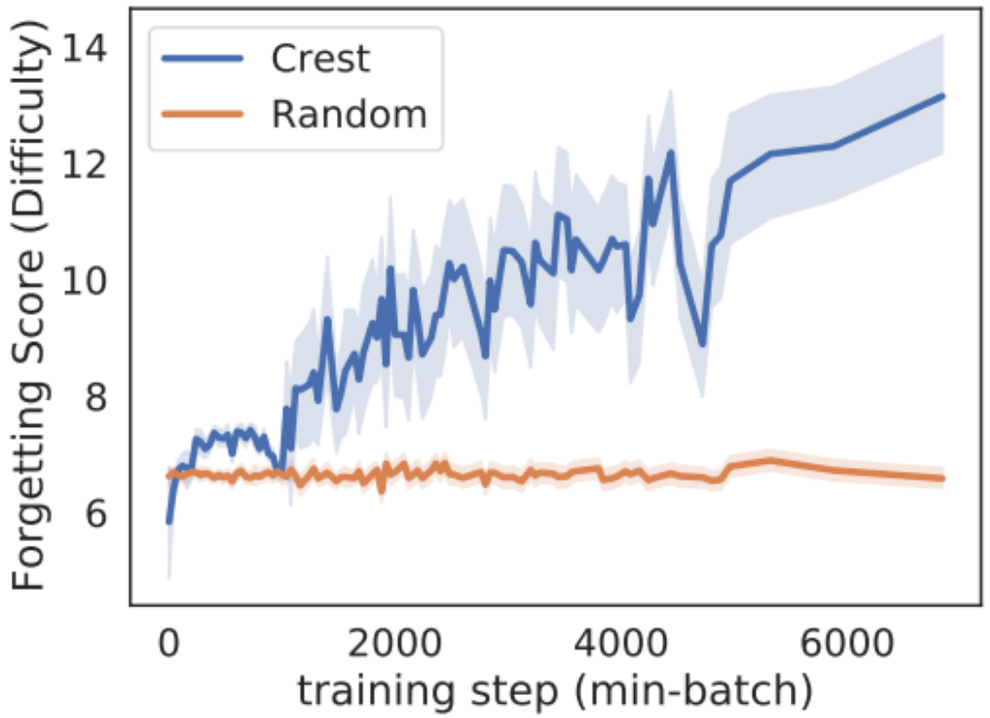


Figure 5: Average forgettability score of CREST coresets

Data Summarization for Robust Machine Learning: Noisy labels

Method: CRUST

- **Idea:** select a subset of **clean** data points that has an approximately **low-rank Jacobian** matrix

- How to find such data points?

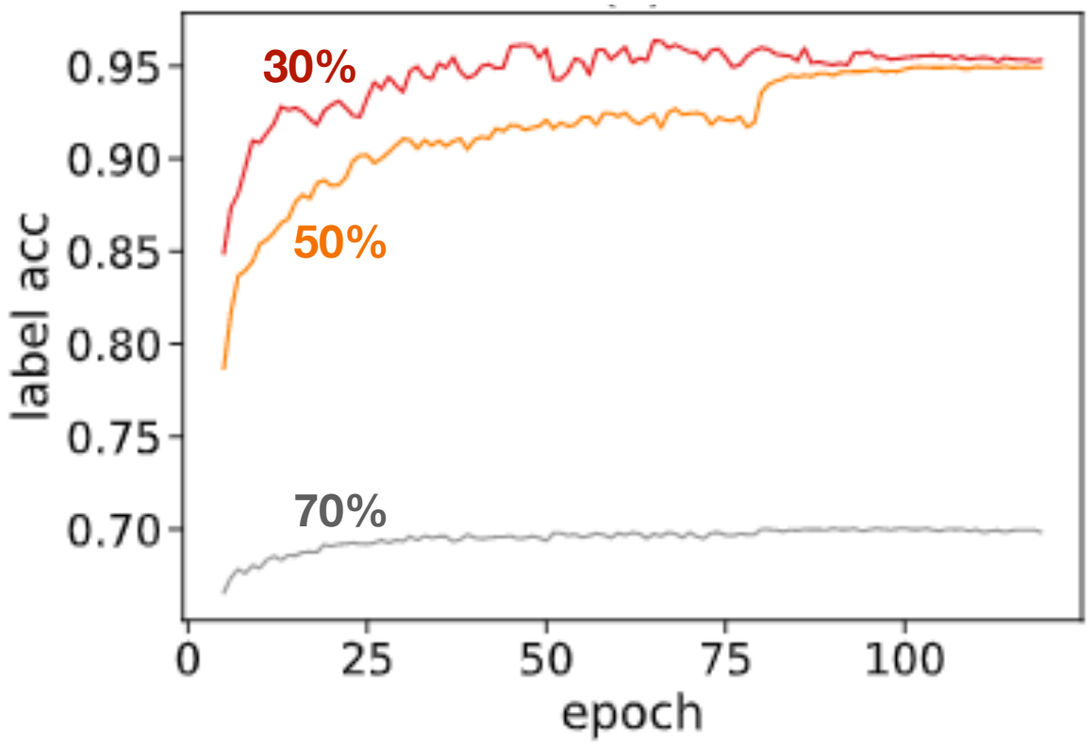
- **Clean data** with similar gradients **cluster** closely together in the gradient space

- **Noisy data spread** out (uniformly) in the gradient space

Gradient space

Experiments: CIFAR10

Training curve for CIFAR-10 with 50% symmetric noise, and corsets of size 30%, 50%, and 70%



Fraction of correct labels in the coreset with respect to epochs.

CRUST finds the clean labels

Data Summarization for Robust Machine Learning: Data Poisoning Attacks

Targeted Data poisoning attacks

- Change the prediction of x_t in the test to an adversarial label y_{adv}, by modifying a fraction (usually less than 1%) of data points within an l_∞-norm ϵ-bound:

$$\min_{\delta \in \mathscr{C}} \mathscr{L}(x_t, y_{\text{adv}}, \theta(\delta)) \quad s.t. \quad \theta(\delta) = \arg\min_\theta \sum_{i \in V} \mathscr{L}(x_i + \delta_i, y_i, \theta)$$

Adversarial loss **Training loss**

where $\mathscr{C} = \{\delta \in \mathbb{R}^{n \times m} : \|\delta\|_\infty \leq \epsilon, \delta_i = 0 \ \forall i \notin V_p\}$

Small bounded perturbation

Robustness against data poisoning attacks

- **Motivation**: the poisons need to pull the representation of the target towards the poison class.

$$\min_{\delta \in \mathscr{C}} \mathscr{L}(x_t, y_{\text{adv}}, \theta(\delta)) \quad s.t. \quad \theta(\delta) = \arg\min_{\theta} \sum_{i \in V} \mathscr{L}(x_i + \delta_i, y_i, \theta)$$

Adversarial loss **Training loss**

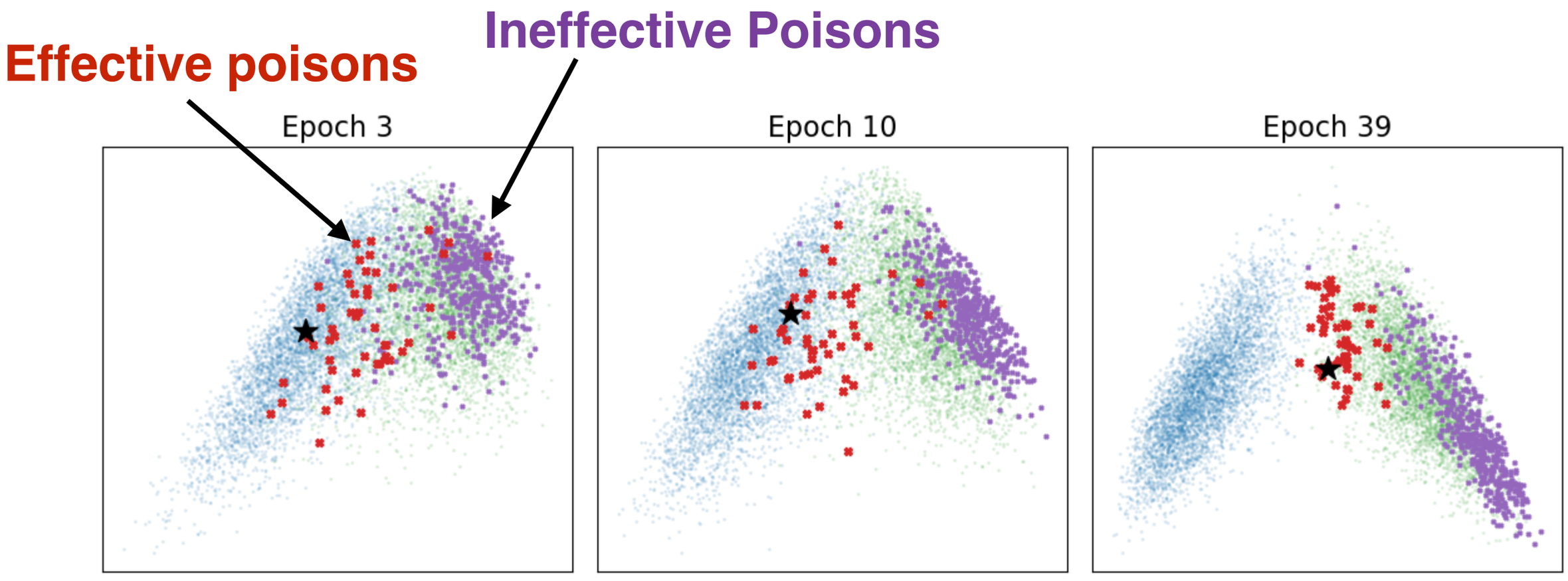
- To do so, they need to mimic the gradient of the adversarially labeled target. Formally,

$$\nabla \mathscr{L}(x_t, y_{\text{adv}}, \theta) \approx \frac{1}{|V_p|} \sum_{i \in V_p} \nabla \mathscr{L}(x_i + \delta_i, y_i, \theta)$$

Adversarial gradient **Training gradient**

Effective Poisons

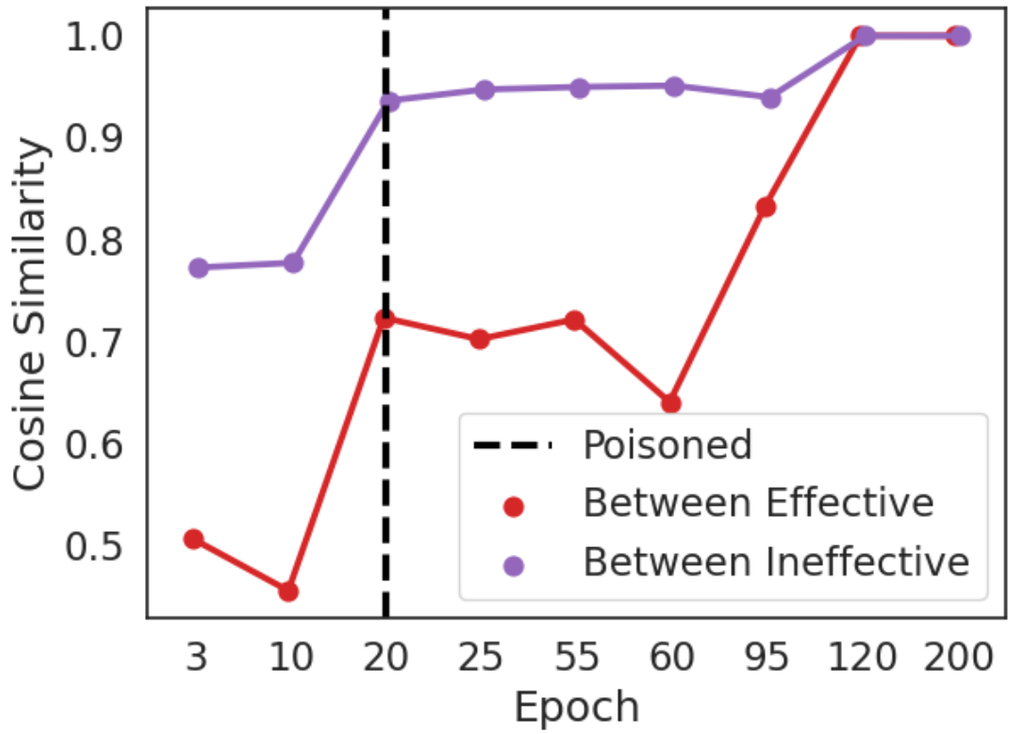
- **Observation**: only a fraction of the poisons can be optimized by bounded perturbations to have a gradient that is close enough to the target gradient on a pre-trained clean model.



- Effective poisons are closer to target in gradient space

Effective poisons get isolated

- Ineffective poisons gradients are relatively similar to each other
- Effective poisons gradients are not similar to each other or clean data
- Effective poisons' gradients get isolated



Method (Epic):

- Cluster the data using k-medoids in the gradient space
- Drop (isolated) clusters of size 1

Data Summarization for Robust Machine Learning: Spurious Correlations

Spurious Biases

- **Spurious correlation**: undesirable correlation between a class-irrelevant feature and label
  - Results in poor worst-group accuracy (test acc on minority groups that do not contain the spurious feature)

Majority: high test accuracy

Minority: low test accuracy

How Spurious Features are Learned?

- Simple features (with **smaller variance or larger magnitude**) are learned faster

- If a simple spurious feature exists in a class, it is learned **early** in training (first few epochs)

- Spurious features with **smaller variance or larger magnitude** are learned faster

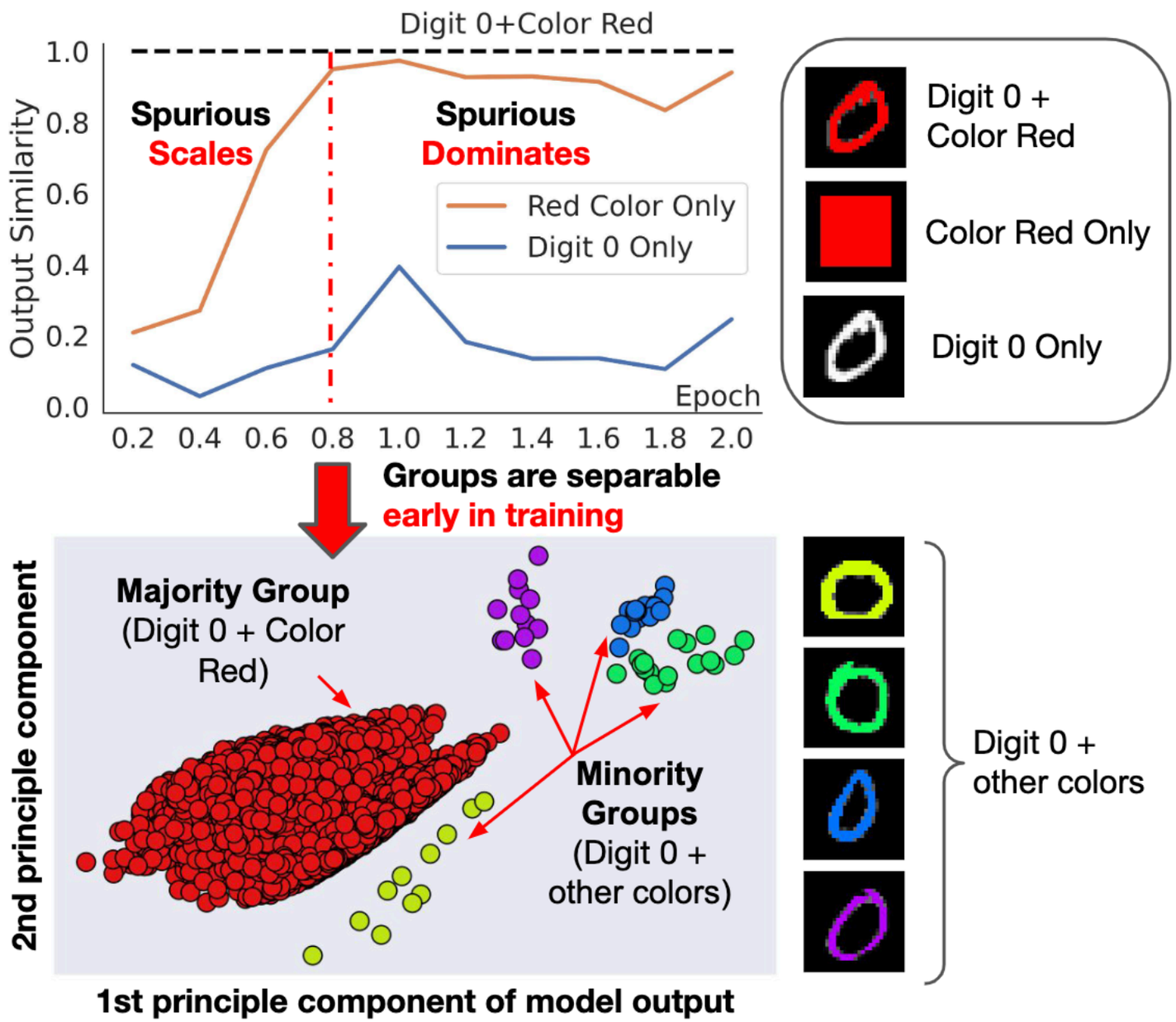
- The model **does not learn the core features** of the majority groups

- Majority examples have a low loss because the model can use the spurious feature to correctly predict their label

- **Larger models are more prone** to learning spurious correlations

Eliminating Spurious Biases Early in Training

- Cluster the data based on model's output early in training
- Larger cluster contains the majority group



Eliminating Spurious Biases Early in Training

- To eliminate learning the spurious biases:
  - 1- Upsample minority: In every mini-batch, sample from smaller (minority) clusters with a higher probability
  - 2- Upweight minority: Group Distributionally Robust Optimization (GDRO) upweights the loss of the group with largest loss during training

Can we find corsets for training Large Language Models (LLMs) with theoretical guarantees?

Smaller Subsets Yield a better Performance

- Because:
  - There is a big shift between training and test data distributions

- Besides:
  - Larger data harms the post-training (SFT) performance
  - The model forgets the information learned during pretraining

Data-efficient Fine-tuning of LLMs

Can we find a subset S^* that closely estimates the full gradient?

$$S^* = \arg\min_{S \subseteq V} |S|, \quad \text{s.t.} \quad \max_{w \in \mathcal{W}} \left\| \frac{1}{|V|} \sum_{i \in V} \nabla f_i(w) - \frac{1}{|S|} \sum_{j \in S} \nabla f_j(w) \right\| \leq \epsilon .$$

Full gradient **Gradient of S**

- For vision/classification, last layer gradients (logits) can be used

Challenge for LLMs:

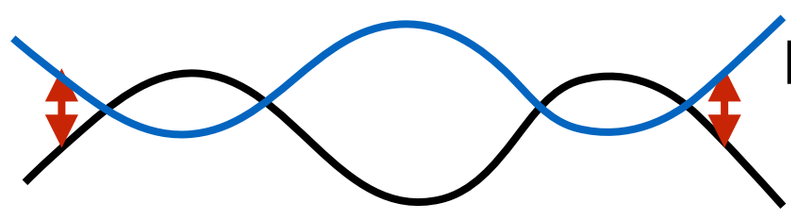
- (Even last layer) Gradients are too high-dimensional!
- For example, dimensionality of the last V projection of Phi-2 has
  - 6.5M dimensions when training the full parameters
  - 327K dimensions when training with LoRA

Small2Large (S2L): Data-efficient Fine-tuning of LLMs

Observation:

- Fine-tuning has a relatively smooth loss
- Curvature is small during fine-tuning

Lemma (informal): Assuming a small curvature, examples with similar loss trajectory (i.e. similar loss values during fine-tuning) have similar gradients.



- Gradient clusters can be found by clustering loss trajectories!

S2L: Data-efficient Fine-tuning of LLMs

Finding a subset that captures the gradient of full data

- Fine-tune a small proxy model on (a subset of) data and save the loss trajectories (e.g. at the beginning of every epoch)
- Cluster the loss trajectories

- Sample equal number of examples from every cluster
- Yields better performance on smaller groups of the data, and a better overall downstream performance

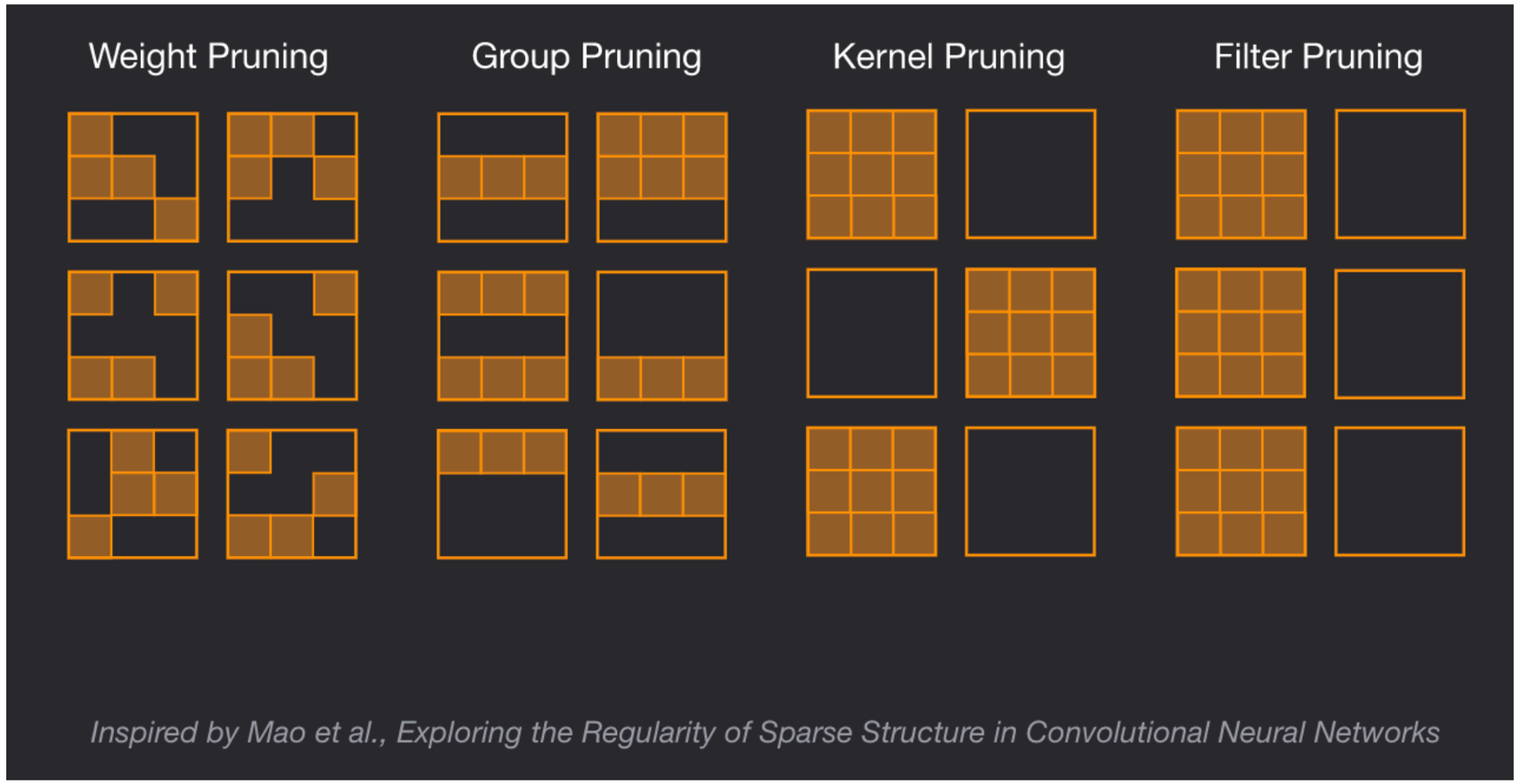
SmallToLarge (S2L): Scalable Data Selection for Fine-tuning Large Language Models by Summarizing Training Trajectories of Small Models, [NeurIPS'24]

Neural Network Pruning:

what's the motivation?

Types of Pruning

- Trade-off between precision and acceleration
  - More fine-grained (or unstructured) pruning methods are more precise but make acceleration more difficult
  - Removing bigger chunks at a time (structured pruning) is less accurate, but allows for sparse matrix computation



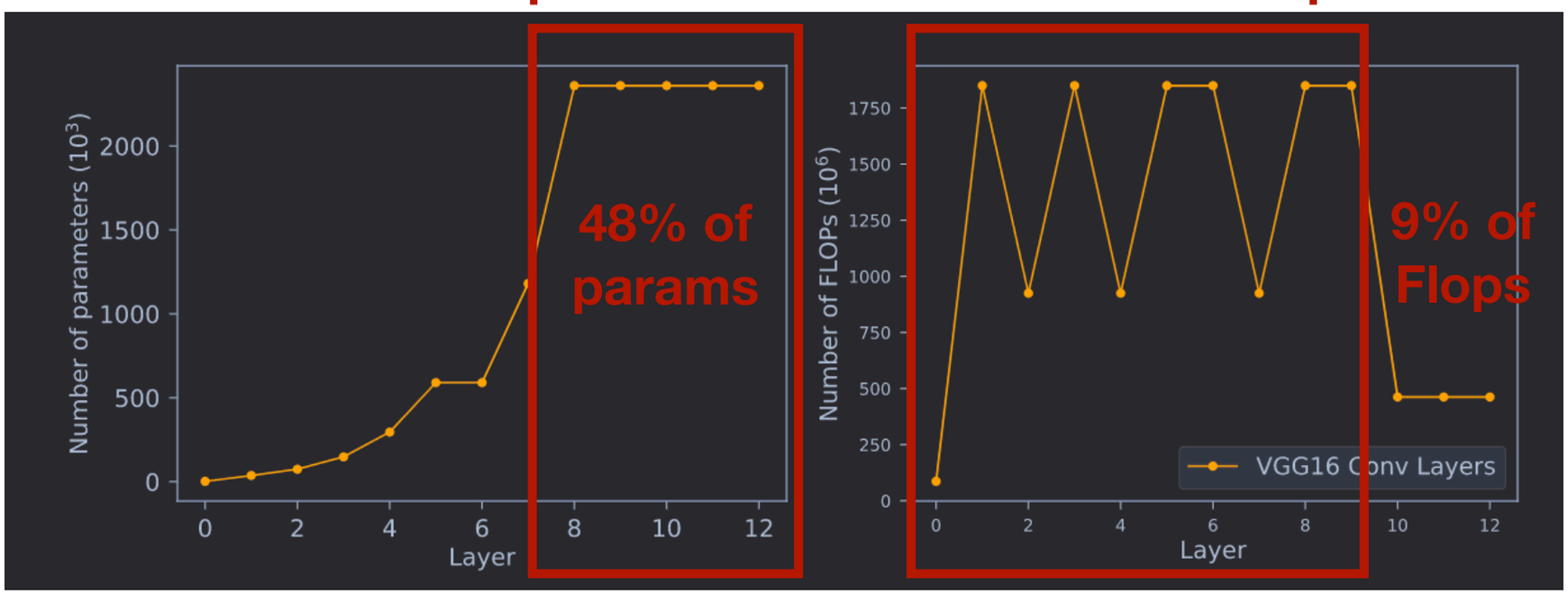
Evaluation

- **Compression ratio**: total_params/nonzero_params
- **Theoretical Speedup**: total_flops/nonzero_flops
- Both metrics should be reported:

Flops: Floating point operations

  - The speedup greatly depends on where in the network the pruning is performed.
  - For the same compression ratio, two similar architectures can have widely different speedup values.

Most of the parameters **Most of the computation**



For same speedup we need to prune more from the end

VGG.The last 3 layers hold 48% of the parameters while being only responsible of 9% of the total FLOPs

Neural Network Pruning

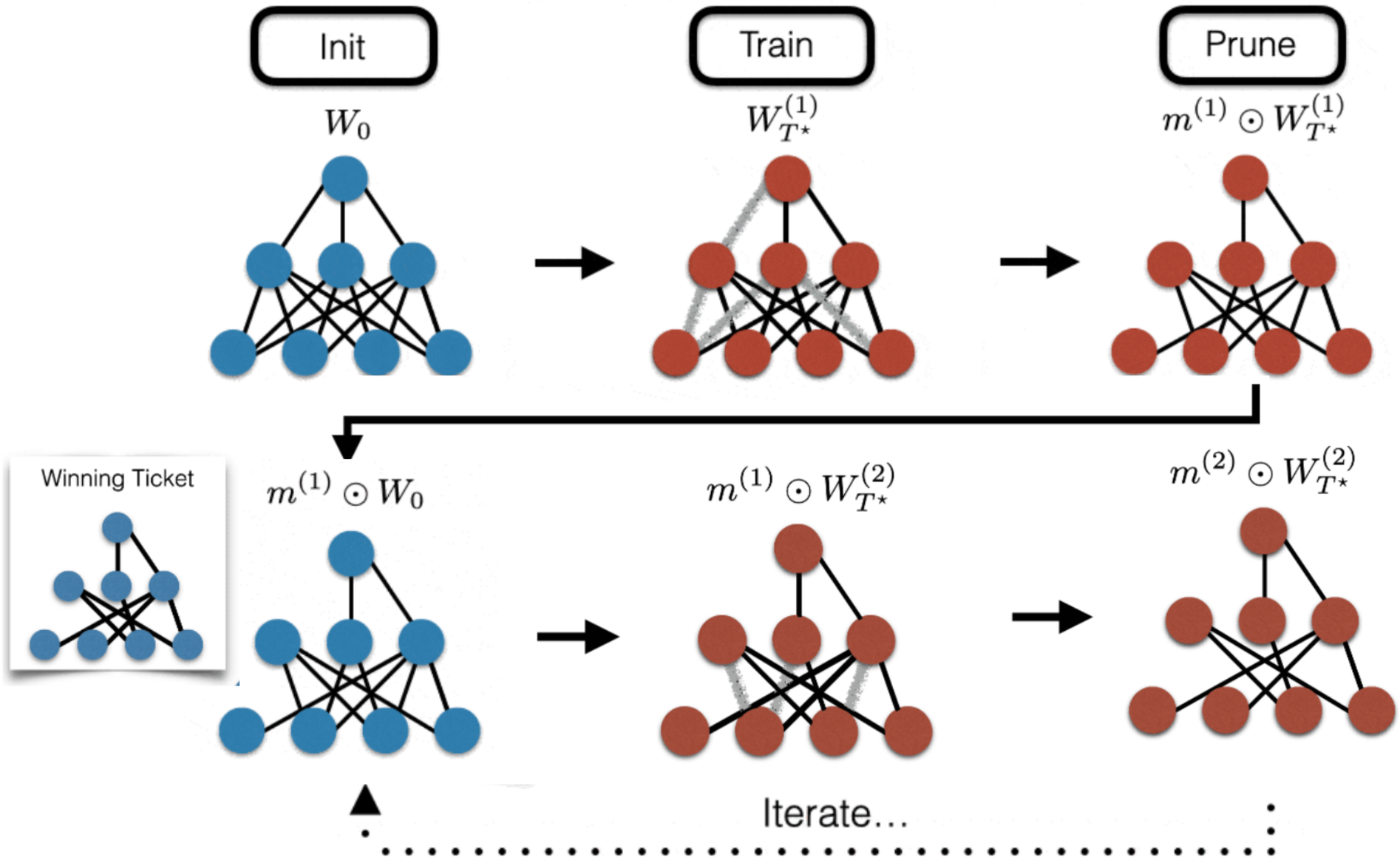
- The most common ways:
  - **Weight Magnitude pruning**: compute L1-norm, i.e $\sum_i |x_i|$, of the weights (or group/kernel/filter depending on the granularity), and remove the ones with the lowest values.
  - **Gradient magnitude pruning**: multiply the weights by their corresponding gradients before computing the L1-norm.
- These criteria can be evaluated:
  - **locally**: each channel is pruned until the desired sparsity is reached, resulting in equally sparse layers.
  - **globally**: we evaluate the weights over the whole network, resulting in a sparse network, but with layers having different sparsity values.

Can we train the pruned networks?

- Pruning methods can reduce parameter-counts by more than 90% without harming accuracy.
  - Benefits: decreased size and energy consumption, efficient inference
- However, pruned networks are harder to train from scratch, and reach lower accuracy.
  - If randomly initialized!!
  - Initializing the weights back to their original values allows to recover the performance.

Finding the Winning Ticket

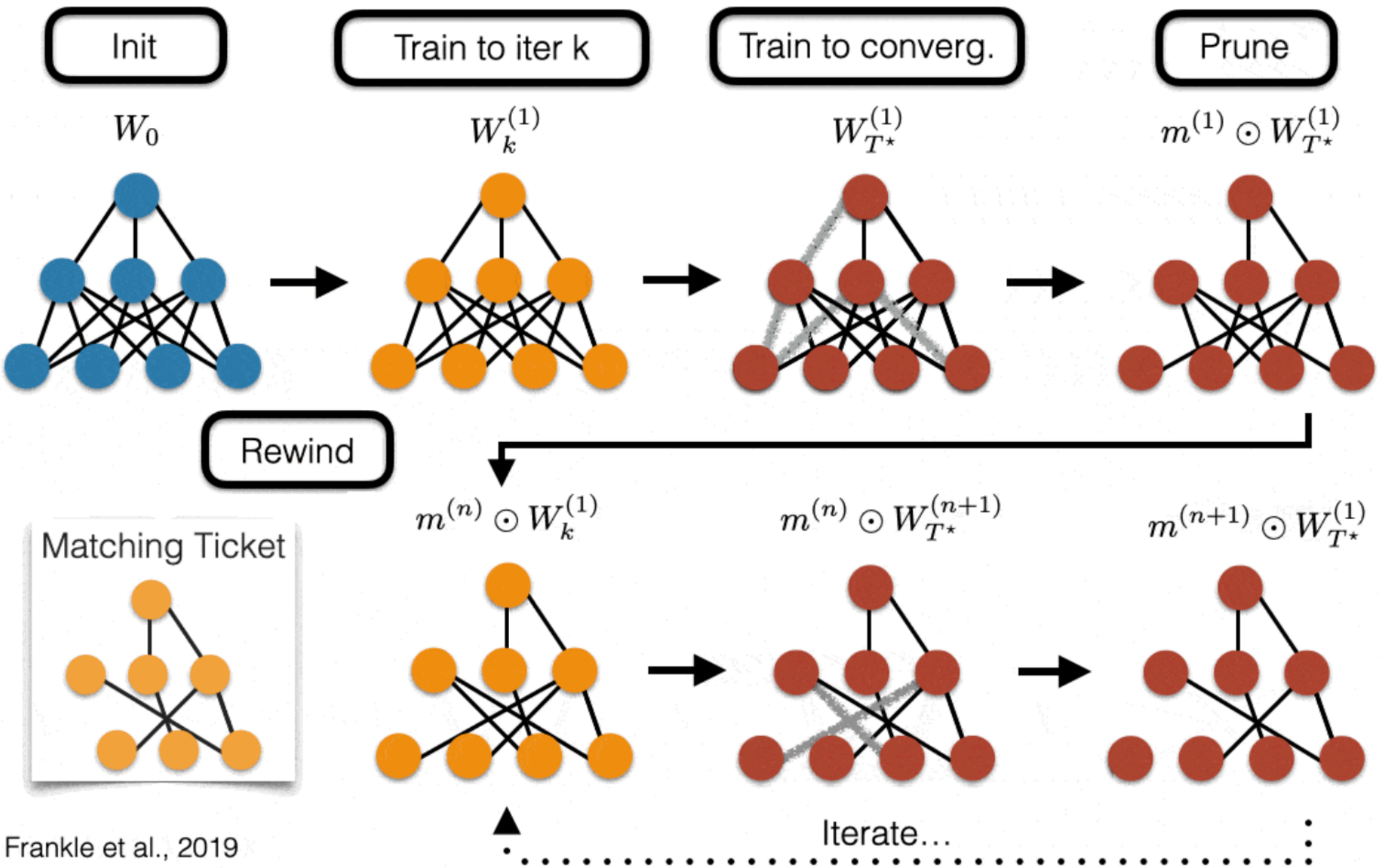
- Iterative magnitude pruning (IMP)



Lottery Ticket Hypothesis - IMP procedure (Frankle & Carbin, 2019)

Iterative Magnitude Pruning with Rewinding

- Instead of rewinding to iteration 0 after pruning, we rewind to iteration k



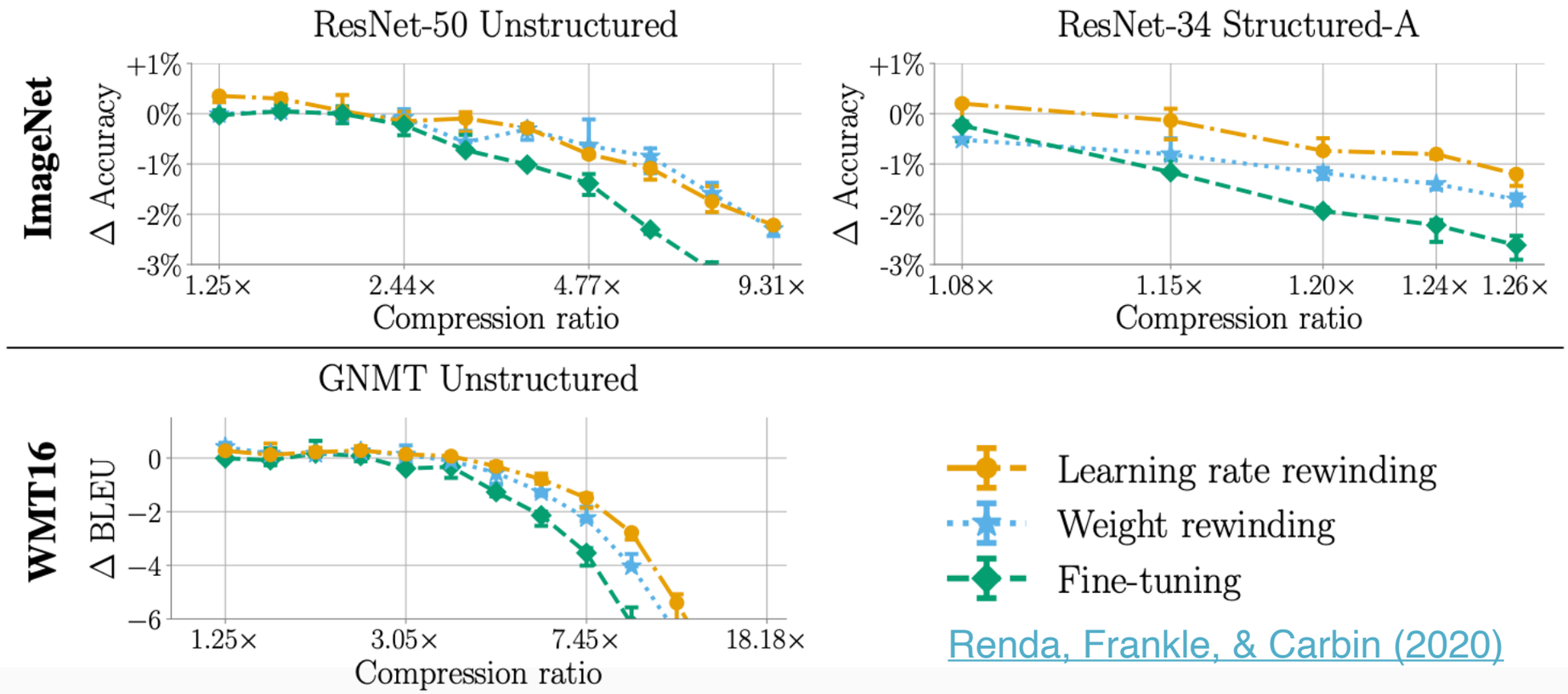
Frankle et al., 2019
Viz: @RobertTLange

Comparing Rewinding and Fine-tuning

- **Fine-Tuning**: After pruning, the remaining weights are trained from their final trained values using a small learning rate (e.g. the final learning rate).

- **Weight Rewinding**: Both weights and learning rate schedule are reset to iteration k.

- **Learning Rate Rewinding**: uses the final unpruned weight and *only* resets the learning rate schedule to iteration k



Renda, Frankle, & Carbin (2020)

Can we prune at initialization?

- SNIP prunes weights that preserve loss at initialization

1st order Taylor Approximation of loss

$$S(\theta_q) = \lim_{\epsilon \to 0} \left| \frac{\mathcal{L}(\boldsymbol{\theta}_0) - \mathcal{L}(\boldsymbol{\theta}_0 + \epsilon \boldsymbol{\delta}_q)}{\epsilon} \right| = \left| \theta_q \frac{\partial \mathcal{L}}{\partial \theta_q} \right|$$

- GraSP prunes weights that preserve gradient norm at initialization

$$\mathbf{S}(-\boldsymbol{\theta}) = -\boldsymbol{\theta} \odot \mathbf{Hg}$$

Loss reduction for one gradient step

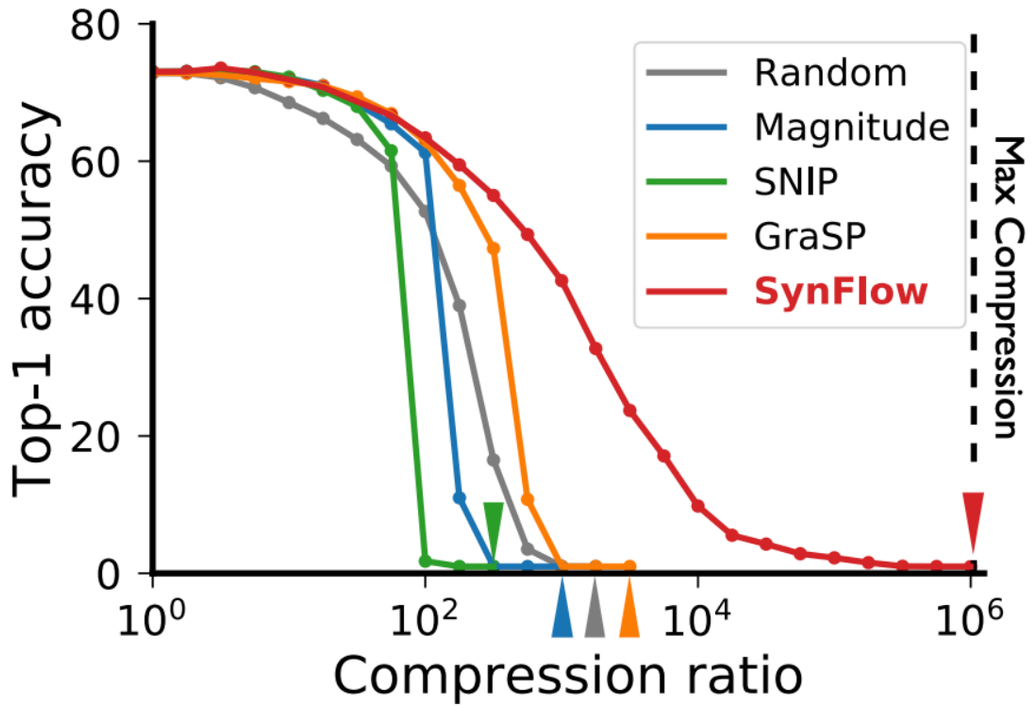
$$\Delta \mathcal{L}(\boldsymbol{\theta}) = \lim_{\epsilon \to 0} \frac{\mathcal{L}\left(\boldsymbol{\theta} + \epsilon \nabla \mathcal{L}(\boldsymbol{\theta})\right) - \mathcal{L}(\boldsymbol{\theta})}{\epsilon} = \nabla \mathcal{L}(\boldsymbol{\theta})^\top \nabla \mathcal{L}(\boldsymbol{\theta})$$

1st order Taylor Approximation of loss reduction

$$\begin{aligned} \mathbf{S}\left(\boldsymbol{\delta}\right) &= \Delta \mathcal{L}(\boldsymbol{\theta}_0 + \boldsymbol{\delta}) - \underbrace{\Delta \mathcal{L}(\boldsymbol{\theta}_0)}_{\text{Const}} = 2\boldsymbol{\delta}^\top \nabla^2 \mathcal{L}(\boldsymbol{\theta}_0) \nabla \mathcal{L}(\boldsymbol{\theta}_0) + \mathcal{O}(\|\boldsymbol{\delta}\|_2^2) \\ &= 2\boldsymbol{\delta}^\top \mathbf{Hg} + \mathcal{O}(\|\boldsymbol{\delta}\|_2^2), \end{aligned}$$

Can we prune at initialization without data?

- Can we obtain winning tickets without any training and in the absence of any data?
- Calculating the mask at once leads to layers collapse
  - Magnitude pruning fully prunes the widest layers
  - Random pruning fully prunes the smallest layers
- SynFlow: Iteratively prune based on $\mathcal{R}_{\text{SF}} = \mathbb{1}^T \left(\prod_{l=1}^{L} |\theta^{[l]}| \right) \mathbb{1}$ at initialization



Layer collapse leads to a sudden drop in accuracy [Tanaka et al. (2020)]

Tickets Generalize across Datasets & Optimizers

- What does this imply?
  - Tickets can be used as general inductive bias
  - A robust matching ticket on a very large dataset (a *universal ticket*) can flexibly act as an initializer for (potentially all/most) loosely domain-associated tasks.
- Similarly to the concept of meta-learning a weight initialization, tickets can *perform a form of amortized search in weight initialization space*.