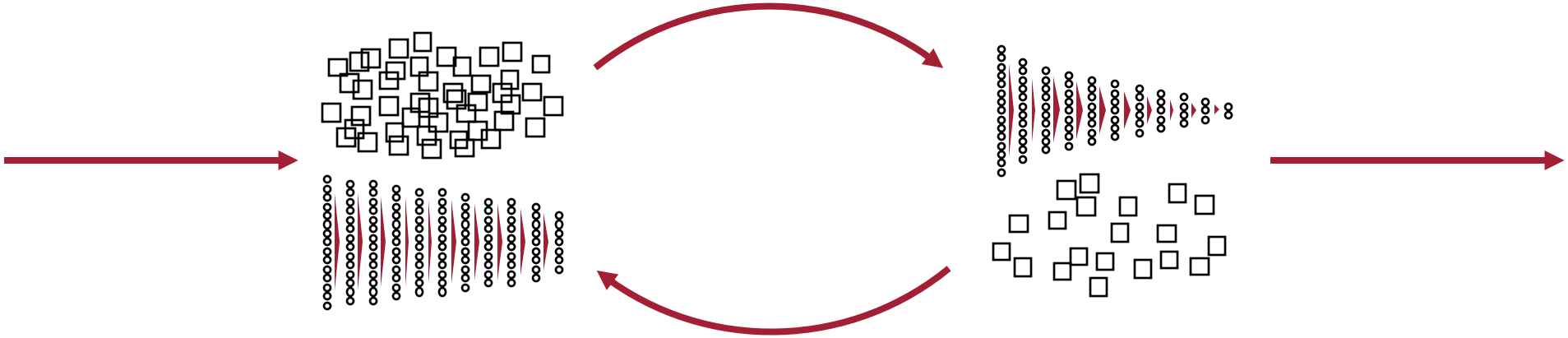


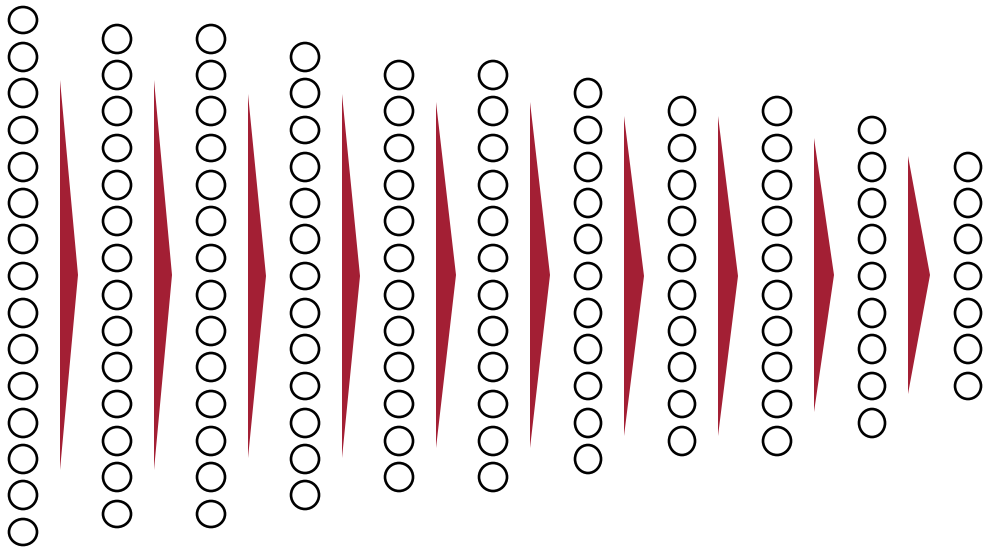
Efficient Deep Learning: From Theory to Practice

Lucas Liebenwein, lucasl@mit.edu, http://www.mit.edu/~lucasl/

CSAIL, MIT

Thesis Supervisor: Daniela Rus, Thesis Committee: Michael Carbin, Song Han, Daniela Rus

Neural networks are SOTA



Natural Language Processing

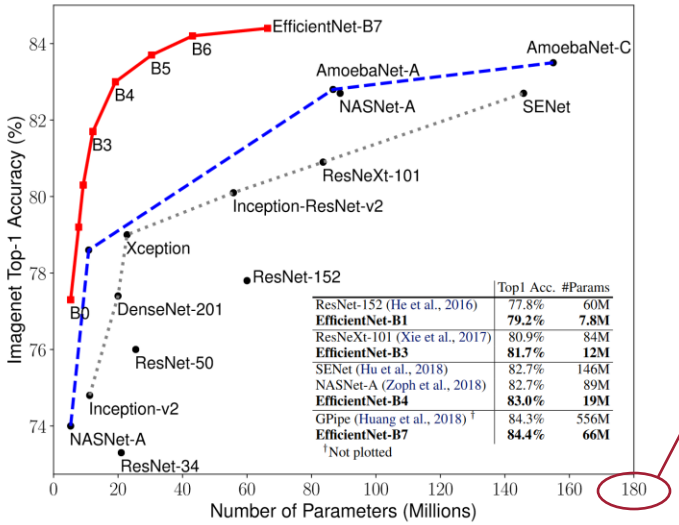
Computer Vision

Robotics

lucasl@mit.edu

The bigger, the better

More Parameters



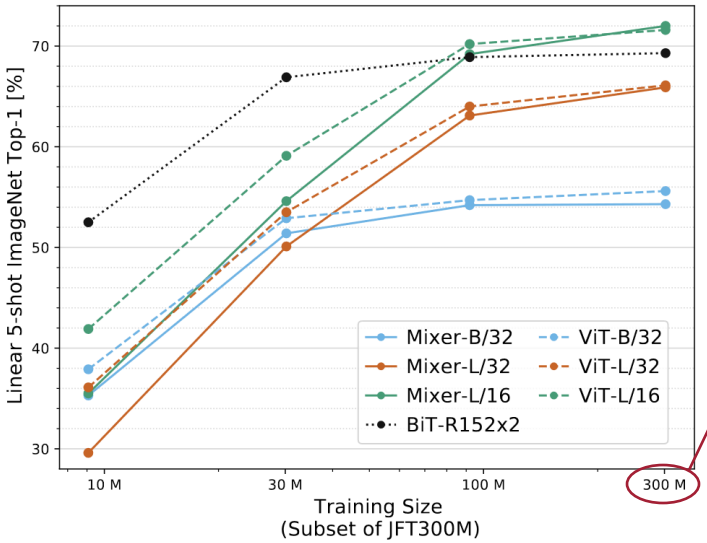
	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**79.2%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.7%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**83.0%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.4%**	**66M**

†Not plotted

180m

Tan, Mingxing, and Quoc Le. "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks." ICML. 2019.

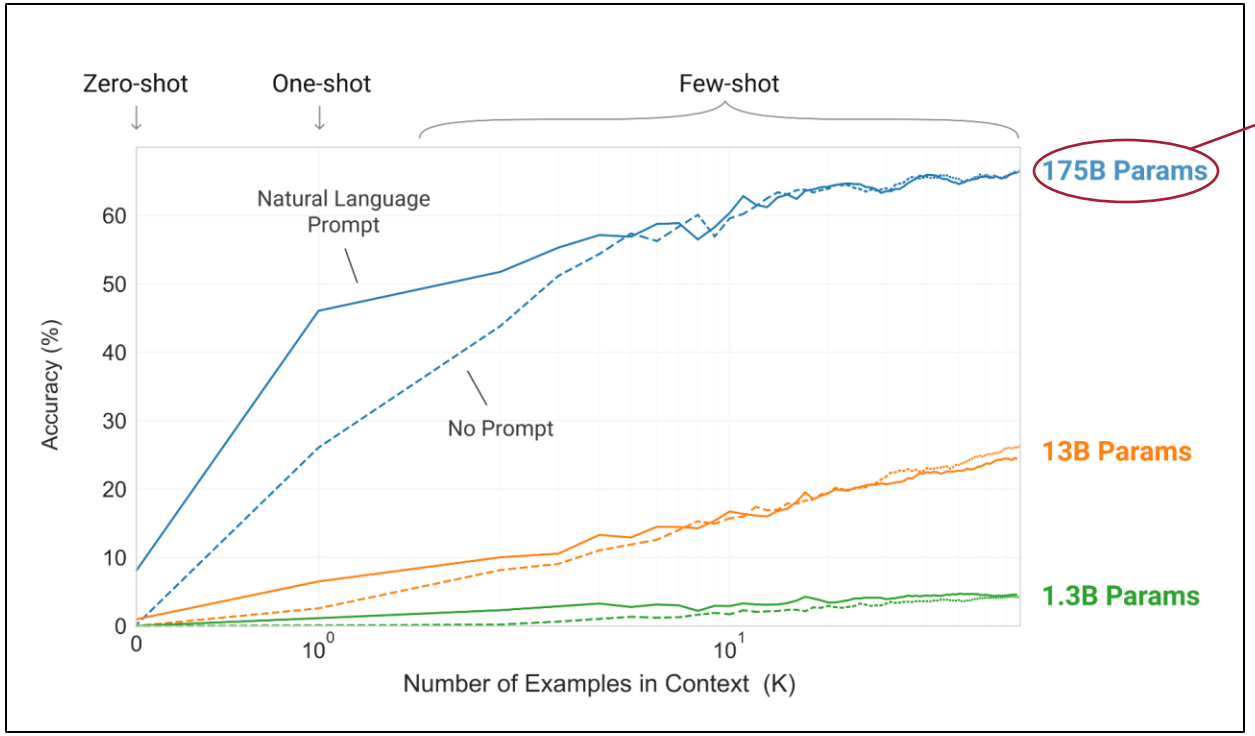
More Data



300m

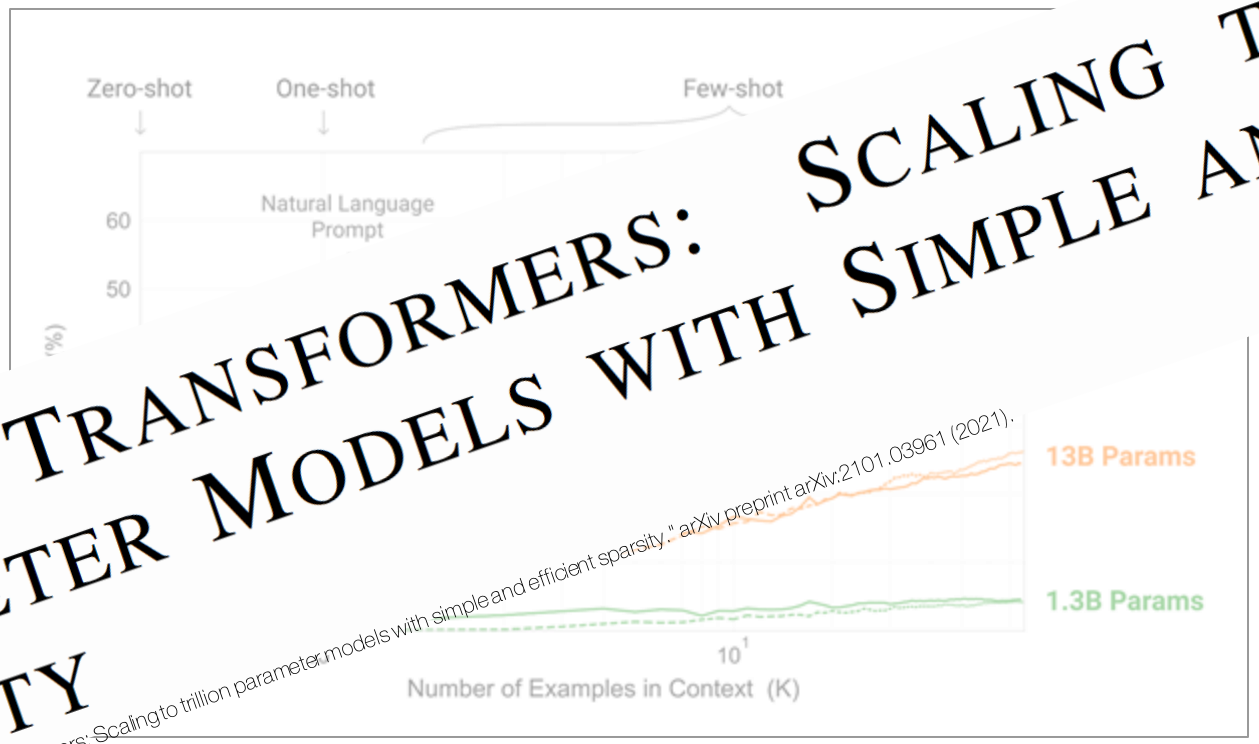
Tolstikhin, Ilya, et al. "Mlp-mixer: An all-mlp architecture for vision." arXiv preprint arXiv:2105.01601. 2021.

The bigger, the better



Brown, Tom, et al. "Language models are few-shot learners." arXiv preprint arXiv:2005.14165 (2020).

The bigger, the better



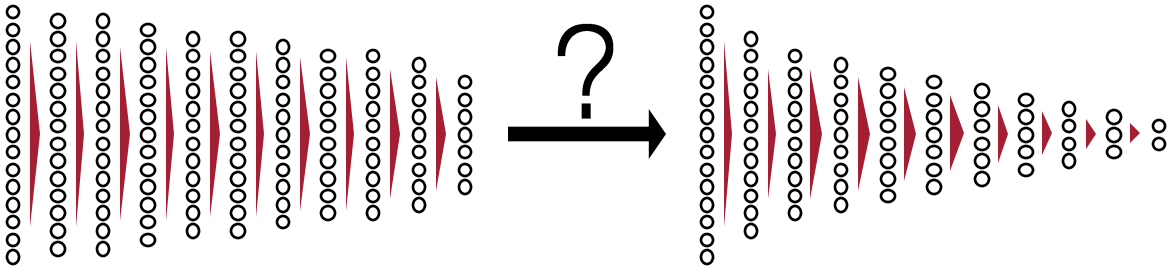
SWITCH TRANSFORMERS: SCALING TO TRILLION PARAMETER MODELS WITH SIMPLE AND EFFICIENT SPARSITY

Fedus, William, et al. "Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity." arXiv preprint arXiv:2101.03961 (2021).

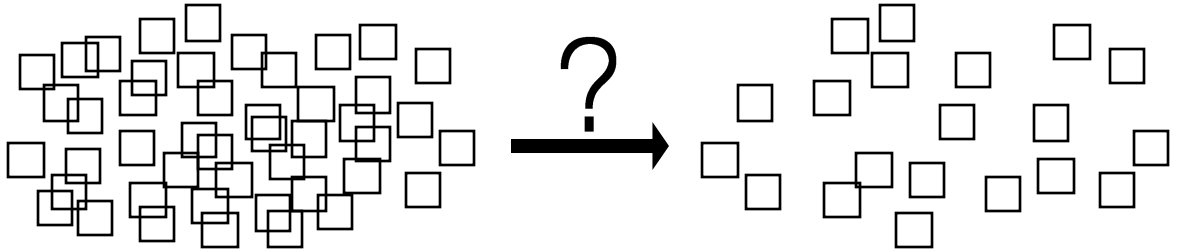
Brown, Tom, et al. "Language models are few-shot learners." arXiv preprint arXiv:2005.14165 (2020).

Less resources, same performance?

Less Parameters



Effects on Generalization

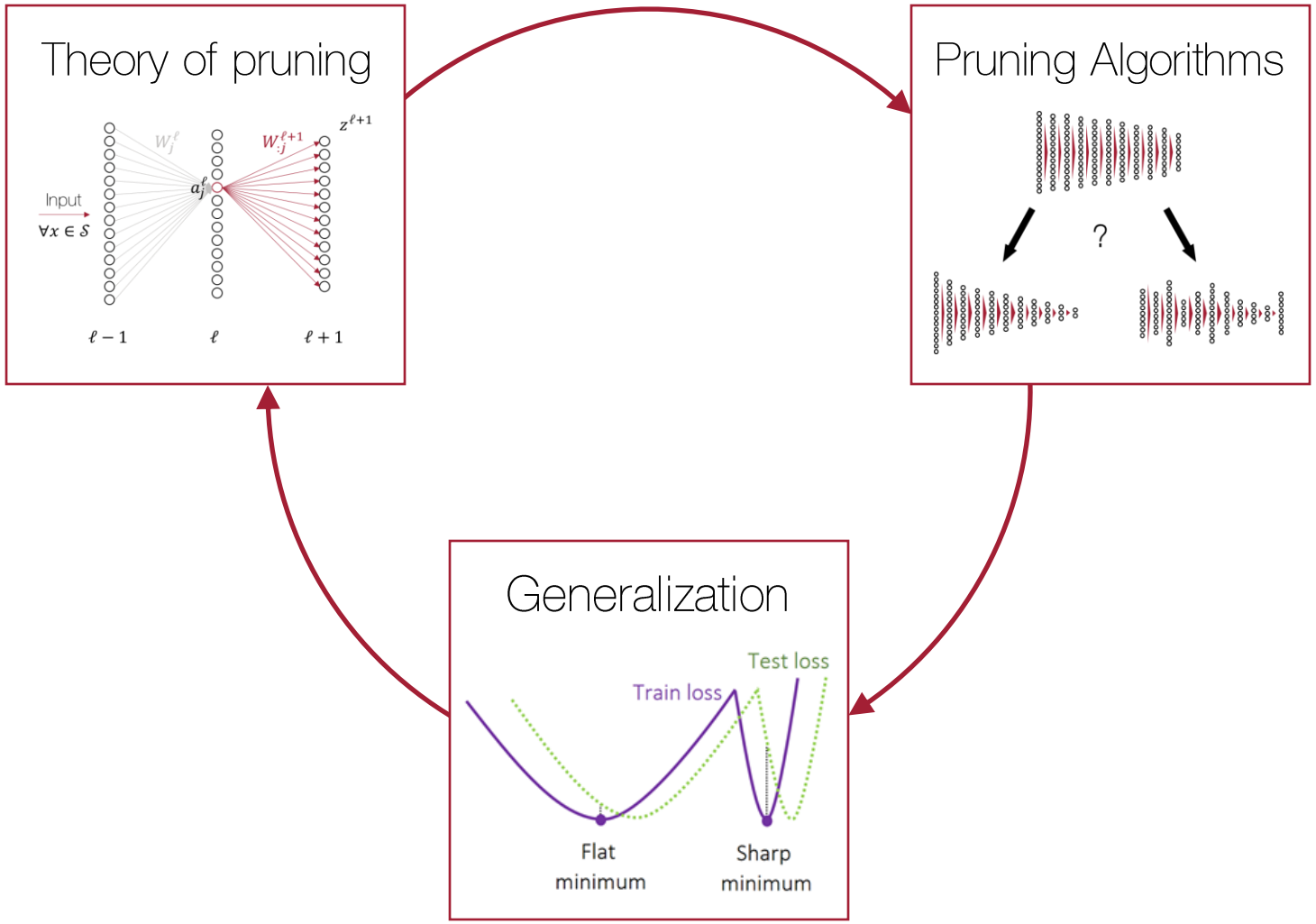


lucasl@mit.edu

Goals of the talk

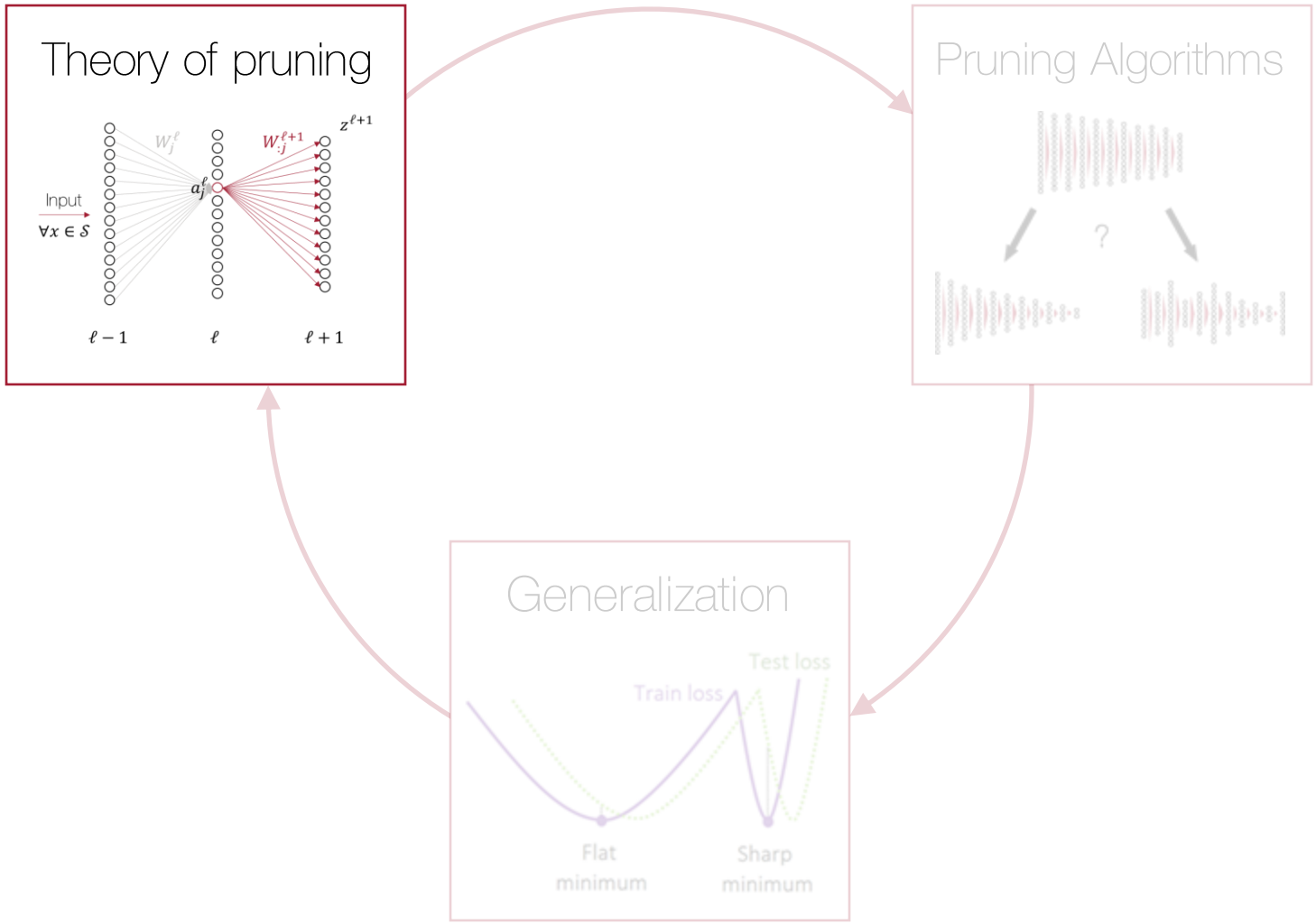
1. Derive a theoretical understanding of *parameter efficiency* in deep learning.
2. Understand the limitations and the potential of theoretical results.
3. Translate our findings into efficient algorithms in practice.
4. Examine the effects of pruning on the generalization performance.

Outline



lucasl@mit.edu

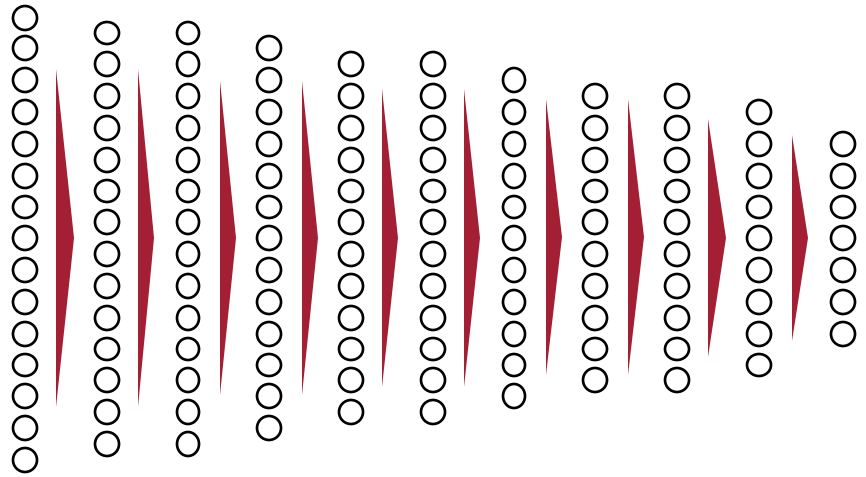
Outline


Theory of pruning
Input
$\forall x \in \mathcal{S}$
W_j^ℓ
a_j^ℓ
$W_{:j}^{\ell+1}$
$z^{\ell+1}$
$\ell - 1$
ℓ
$\ell + 1$
Pruning Algorithms
?
Generalization
Test loss
Train loss
Flat minimum
Sharp minimum

Objective: neuron pruning

For a given network $f_\theta(x)$ with parameters θ and $\epsilon, \delta \in (0,1)$, generate a compressed, *dense* reparameterization $\hat{\theta}$ (i.e., with less neurons) such that

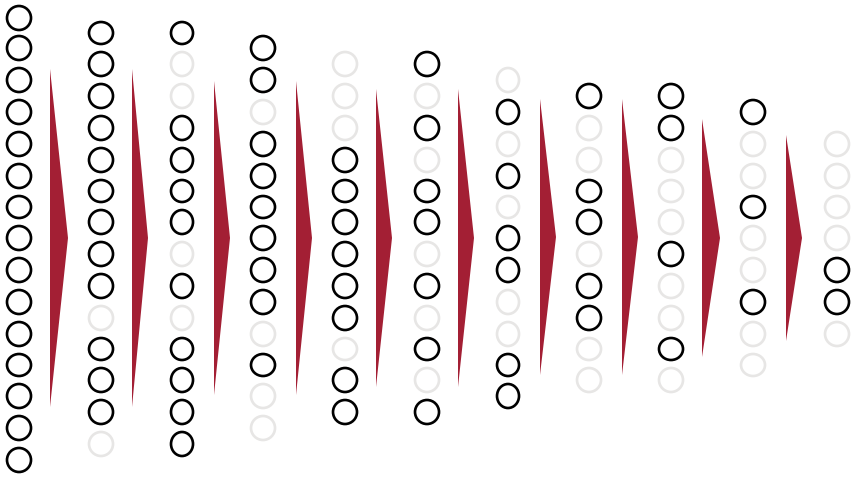
$$\mathbb{P}_{x\sim\mathcal{D}}\left(f_{\hat{\theta}}(x) \in (1 \pm \epsilon) f_\theta(x)\right) \geq (1-\delta) \text{ and } size\left(\hat{\theta}\right) \ll size(\theta).$$



Objective: neuron pruning

For a given network $f_\theta(x)$ with parameters θ and $\epsilon, \delta \in (0,1)$, generate a compressed, *dense* reparameterization $\hat{\theta}$ (i.e., with less neurons) such that

$$\mathbb{P}_{x\sim\mathcal{D}}\left(f_{\hat{\theta}}(x) \in (1 \pm \epsilon) f_\theta(x)\right) \geq (1-\delta) \text{ and } size\left(\hat{\theta}\right) \ll size(\theta).$$

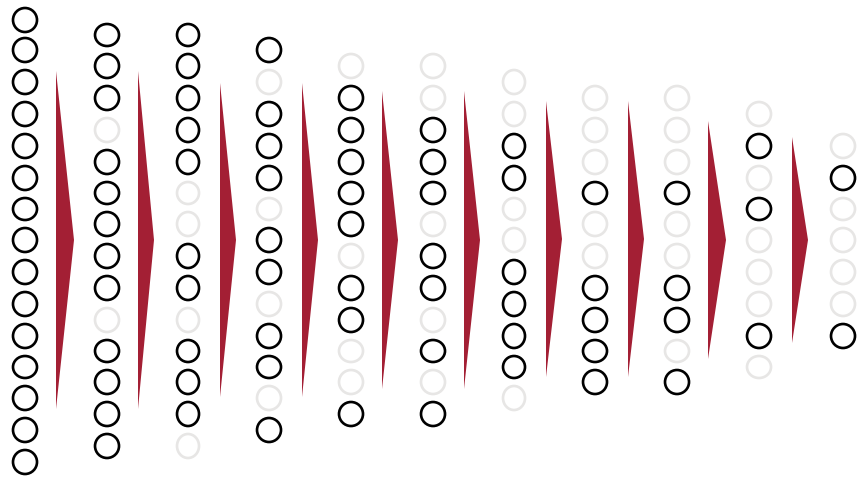


Relative importance s_j^ℓ of neuron j in layer ℓ

Objective: neuron pruning

For a given network $f_\theta(x)$ with parameters θ and $\epsilon, \delta \in (0,1)$, generate a compressed, *dense* reparameterization $\hat{\theta}$ (i.e., with less neurons) such that

$$\mathbb{P}_{x\sim\mathcal{D}}\left(f_{\hat{\theta}}(x) \in (1 \pm \epsilon) f_\theta(x)\right) \geq (1-\delta) \text{ and } size\left(\hat{\theta}\right) \ll size(\theta).$$

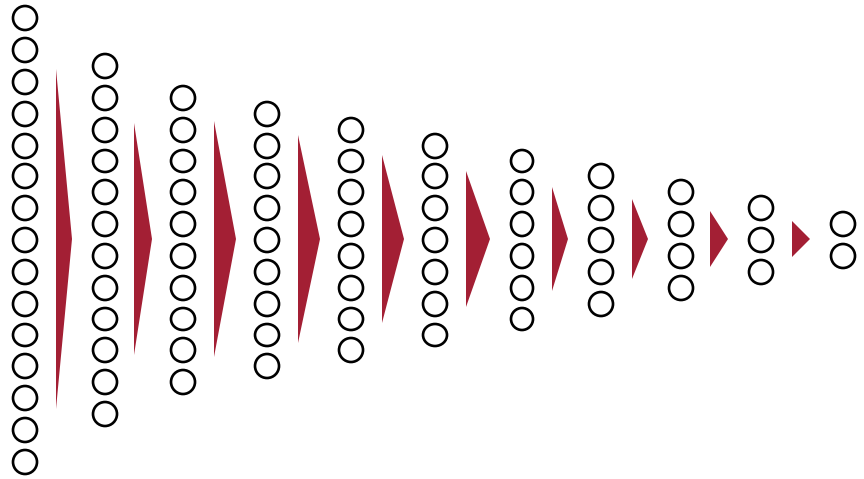


Relative importance s_j^ℓ of neuron j in layer ℓ

Objective: neuron pruning

For a given network $f_\theta(x)$ with parameters θ and $\epsilon, \delta \in (0,1)$, generate a compressed, *dense* reparameterization $\hat{\theta}$ (i.e., with less neurons) such that

$$\mathbb{P}_{x\sim\mathcal{D}}\left(f_{\hat{\theta}}(x) \in (1 \pm \epsilon) f_\theta(x)\right) \geq (1-\delta) \text{ and } size\left(\hat{\theta}\right) \ll size(\theta).$$



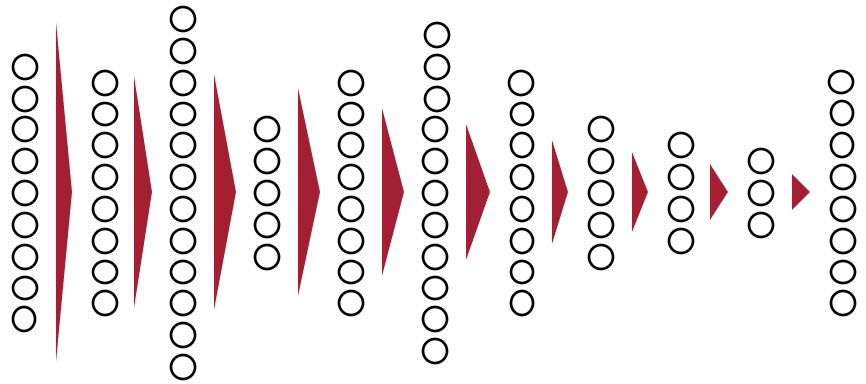
Relative importance s_j^ℓ of neuron j in layer ℓ

Budget allocation to layers $1, \ldots, \ell, \ldots, L$

Objective: neuron pruning

For a given network $f_\theta(x)$ with parameters θ and $\epsilon, \delta \in (0,1)$, generate a compressed, *dense* reparameterization $\hat{\theta}$ (i.e., with less neurons) such that

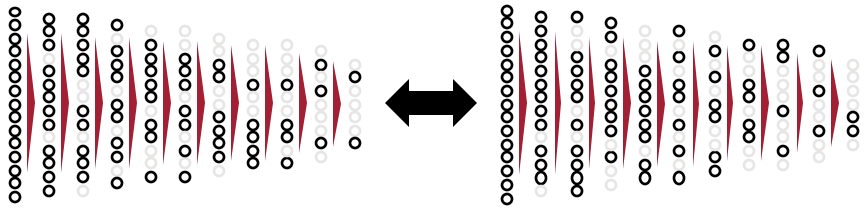
$$\mathbb{P}_{x\sim\mathcal{D}}\left(f_{\hat{\theta}}(x) \in (1 \pm \epsilon) f_\theta(x)\right) \geq (1-\delta) \text{ and } size\left(\hat{\theta}\right) \ll size(\theta).$$



Relative importance s_j^ℓ of neuron j in layer ℓ

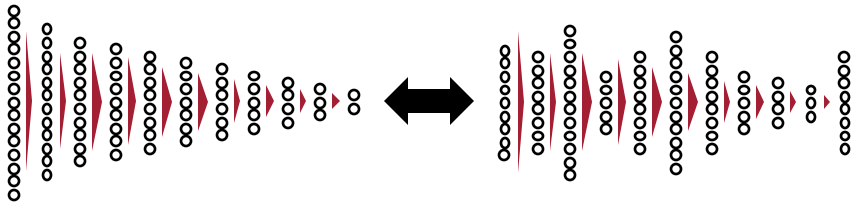
Budget allocation to layers $1, \ldots, \ell, \ldots, L$

Related work



Relative importance s_j^ℓ of neuron j in layer ℓ

- $s_j^\ell \sim \left\|W_j^\ell\right\|_2$, Li et al. 2016
- $s_j^\ell \sim \left\|W_j^\ell\right\|_1$, He et al., 2018
- $s_j^\ell \sim 1 \,/ \max_x \left| z^{\ell+1}(x) - z_{[j]}^{\ell+1}(x) \right|$, Luo et al., 2017

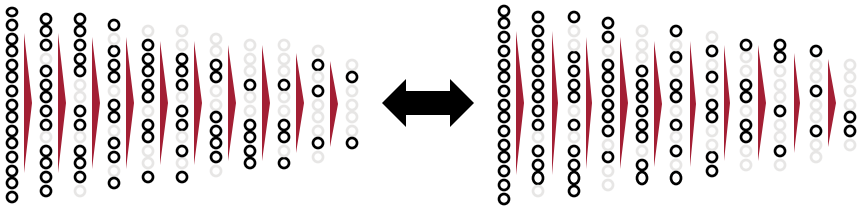


Budget allocation to layers $1, \ldots, \ell, \ldots, L$

- Uniform budget allocation, He et al., 2018
- Manual ablation study, Li et al. 2016
- Sequential pruning process, Luo et al. 2017

j…neuron, ℓ…layer, s_j^ℓ…importance, W_j^ℓ…neuron weights, x…input, $z^\ell(x)$…pre-activation, $z_{[j]}^\ell$…pre-activation with feature j only

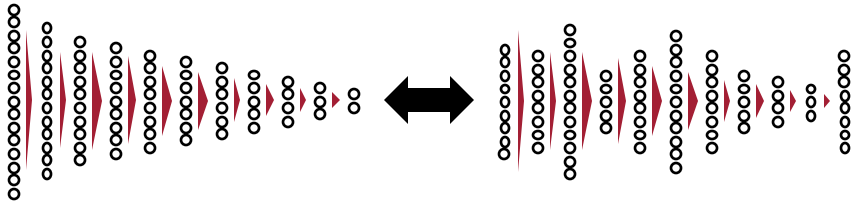
Related work



Relative importance s_j^ℓ of neuron j in layer ℓ

✘ no performance guarantees

✘ data-oblivious heuristics



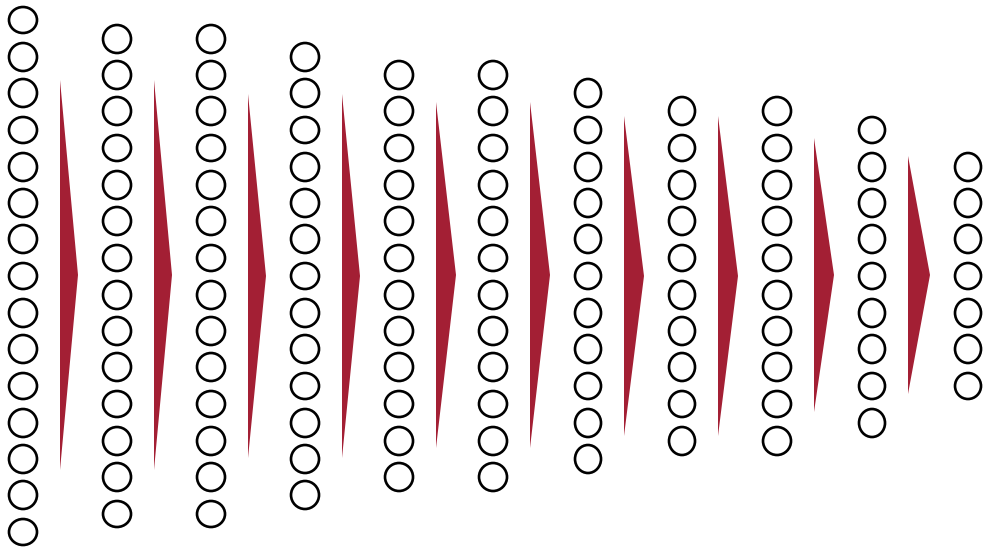
Budget allocation to layers $1, \ldots, \ell, \ldots, L$

✘ manual procedure

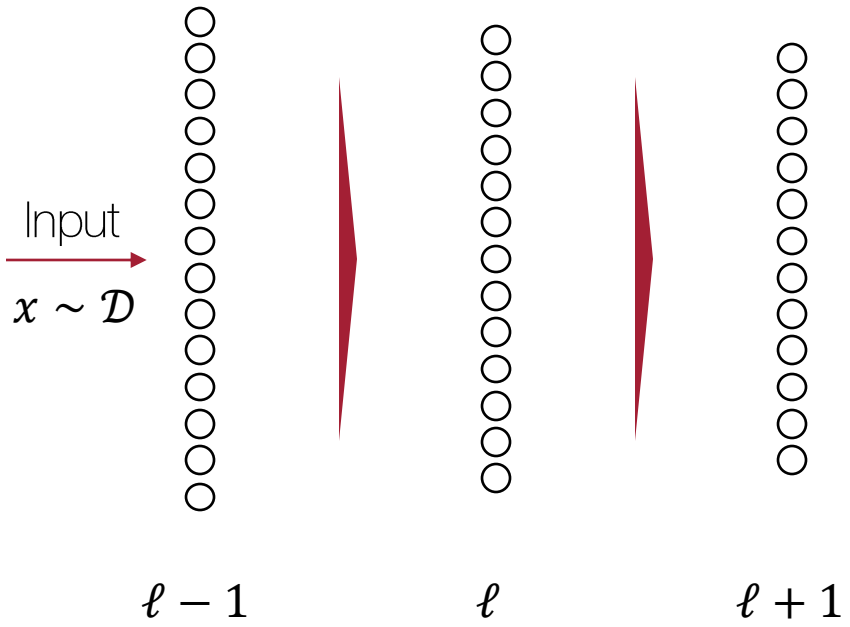
✘ architecture-specific

lucasl@mit.edu

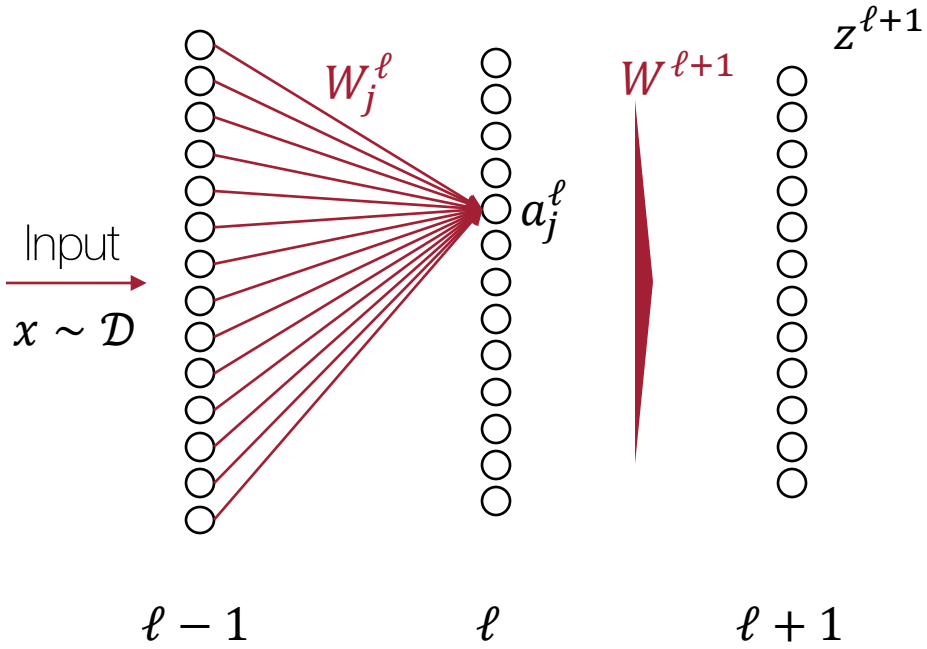
Our method: neuron importance



Our method: neuron importance

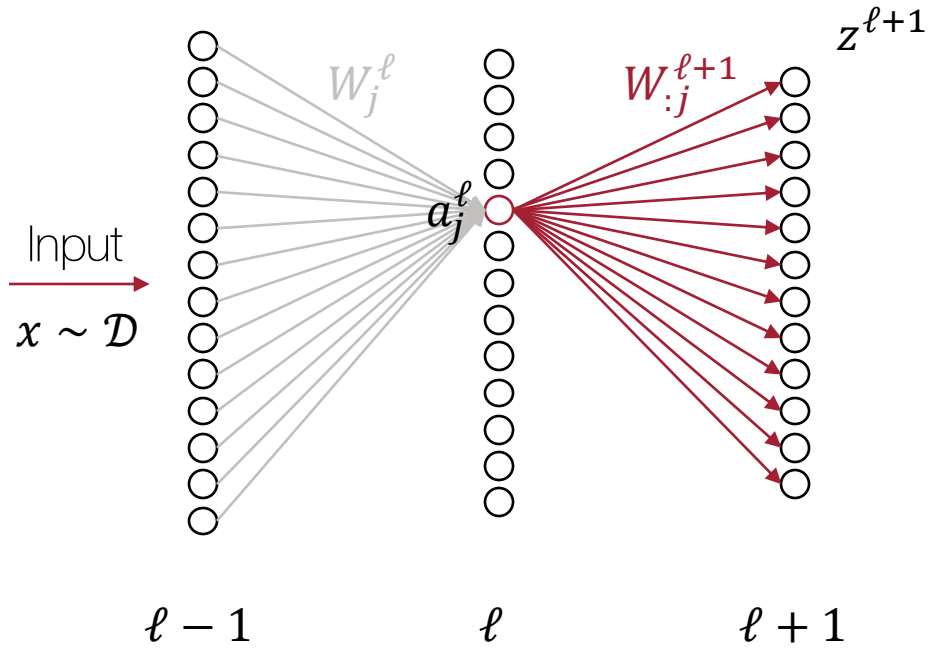


Our method: neuron importance



Quantify importance s_j^ℓ of neuron j in layer ℓ

Our method: neuron importance

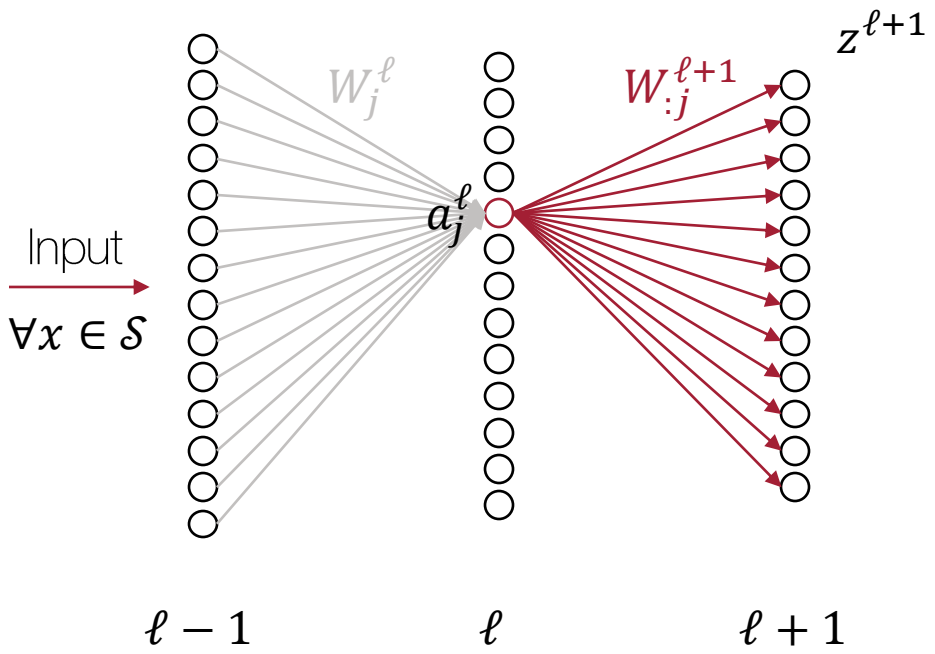


Quantify importance s_j^{ℓ} of neuron j in layer ℓ

Consider maximum contribution of activation $a_j^{\ell}(x)$ in pre-activation $z^{\ell+1}(x)$

$s_j^{\ell} \sim \max_i \frac{w_{ij}^{\ell} a_j^{\ell}(x)}{z_i^{\ell+1}(x)}$ … neuron importance for fixed input $x \sim \mathcal{D}$

Our method: neuron importance

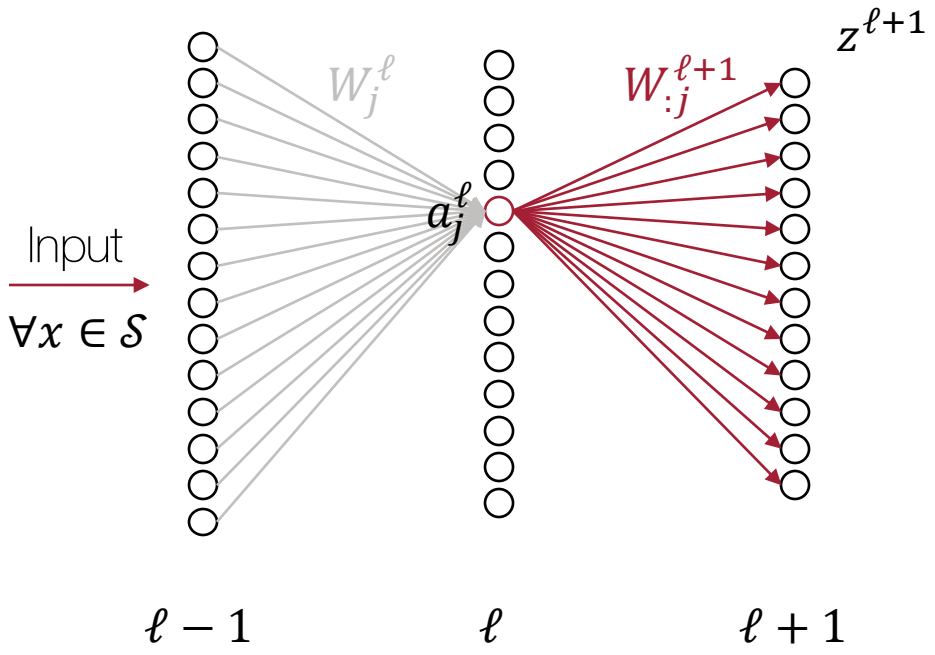


Quantify importance s_j^ℓ of neuron j in layer ℓ

Consider maximum contribution of activation $a_j^\ell(x)$ in pre-activation $z^{\ell+1}(x)$

$s_j^\ell \sim \max_{x \in S} \max_i \frac{w_{ij}^\ell a_j^\ell(x)}{z_i^{\ell+1}(x)}$ … neuron importance for any input $x \sim \mathcal{D}$ with high probability

Our method: neuron importance



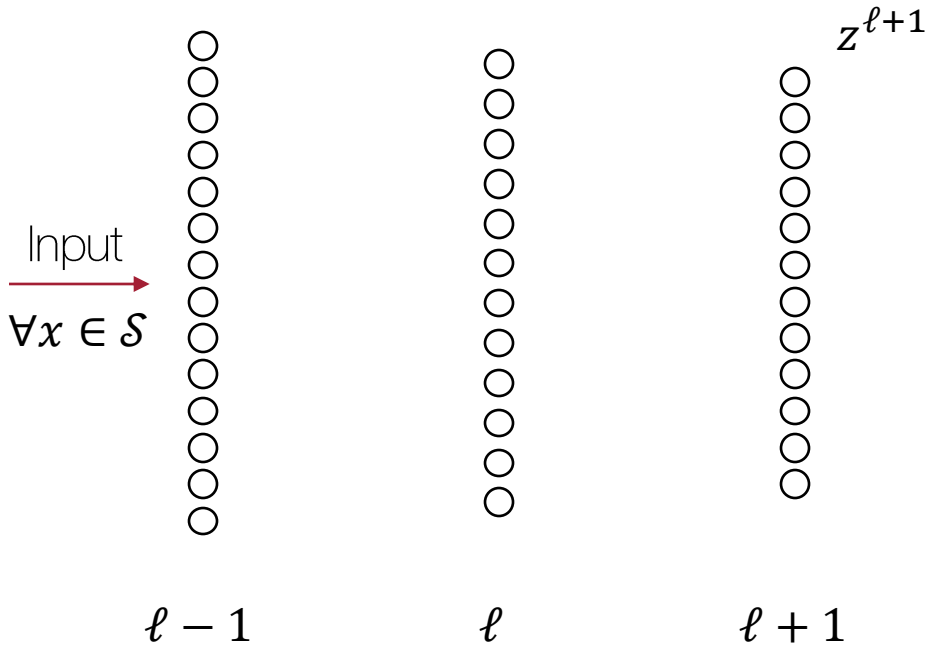
Quantify importance s_j^ℓ of neuron j in layer ℓ

Consider maximum contribution of activation $a_j^\ell(x)$ in pre-activation $z^{\ell+1}(x)$

$s_j^\ell \sim \max_{x \in S} \max_i \frac{w_{ij}^\ell a_j^\ell(x)}{z_i^{\ell+1}(x)}$ … neuron importance for any input $x \sim \mathcal{D}$ with high probability

Provable Prune step:
keep neuron j in layer ℓ with probability $p_j^\ell \sim s_j^\ell$

Our method: neuron importance



Quantify importance s_j^ℓ of neuron j in layer ℓ

Consider maximum contribution of activation $a_j^\ell(x)$ in pre-activation $z^{\ell+1}(x)$

$s_j^\ell \sim \max_{x \in S} \max_i \frac{w_{ij}^\ell a_j^\ell(x)}{z_i^{\ell+1}(x)}$ … neuron importance for any input $x \sim \mathcal{D}$ with high probability

Provable Prune step:
keep neuron j in layer ℓ with probability $p_j^\ell \sim s_j^\ell$

Neuron compression bounds

For a given network $f_\theta(x)$ with parameters θ and $\epsilon, \delta \in (0,1)$, PFP generates a compressed, *dense* reparameterization $\hat{\theta}$ (i.e., with less neurons) such that $\mathbb{P}_{x\sim\mathcal{D}}\left(f_{\hat{\theta}}(x) \in (1 \pm \epsilon) f_\theta(x)\right) \geq (1-\delta)$ and the number of neurons is bounded by $\mathcal{O}\left(\sum_{\ell=1}^{L} \frac{L^2 \left(\Delta^\ell\right)^2 S^\ell \log^\eta/_\delta}{\epsilon^2}\right)$

$S^\ell :=$ "sum of sensitivities"

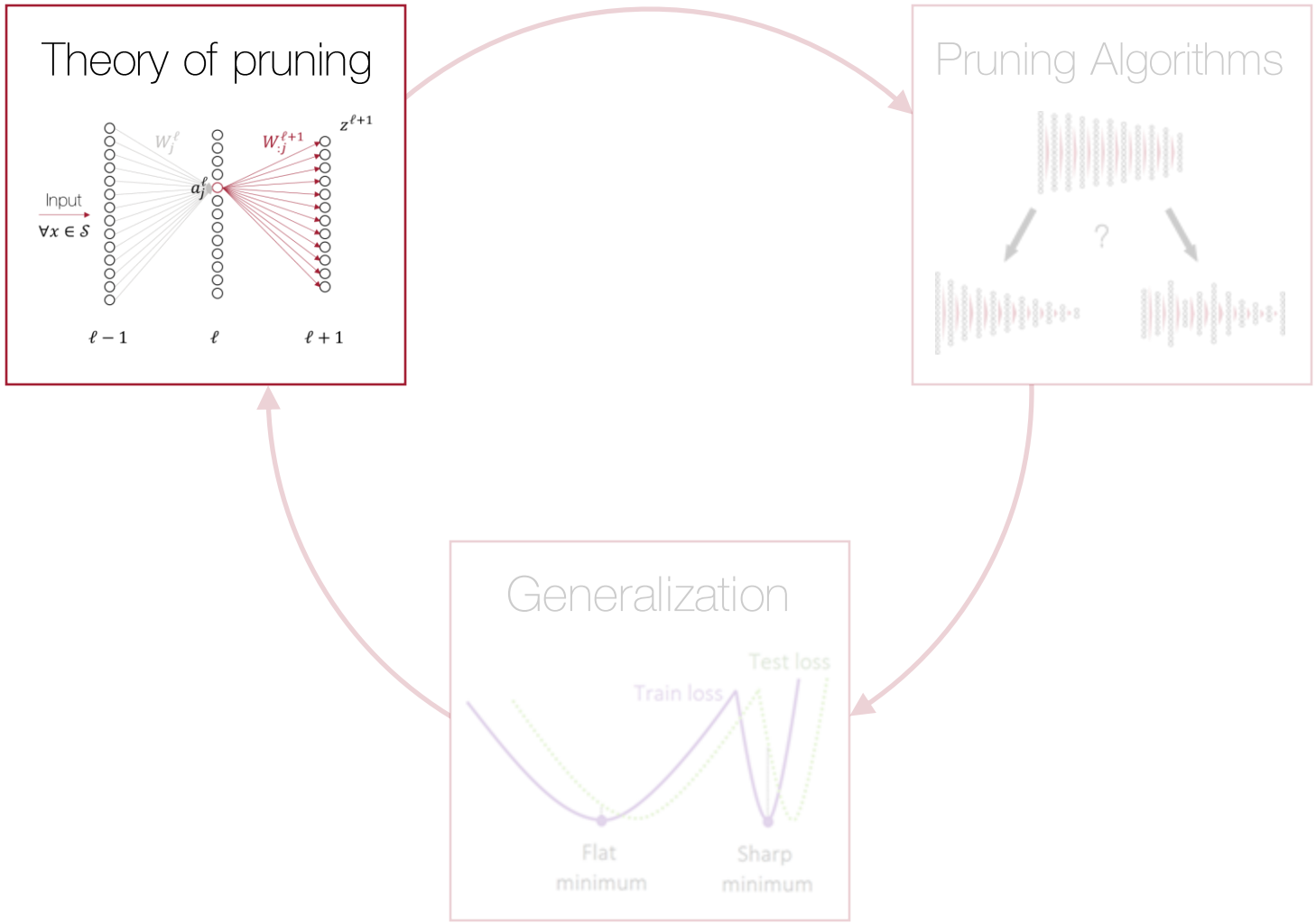
- $S^l = \sum_j s_j^\ell$
- Quantifies "spread" of importance within a layer

$\Delta^\ell :=$ "Propagation Complexity"

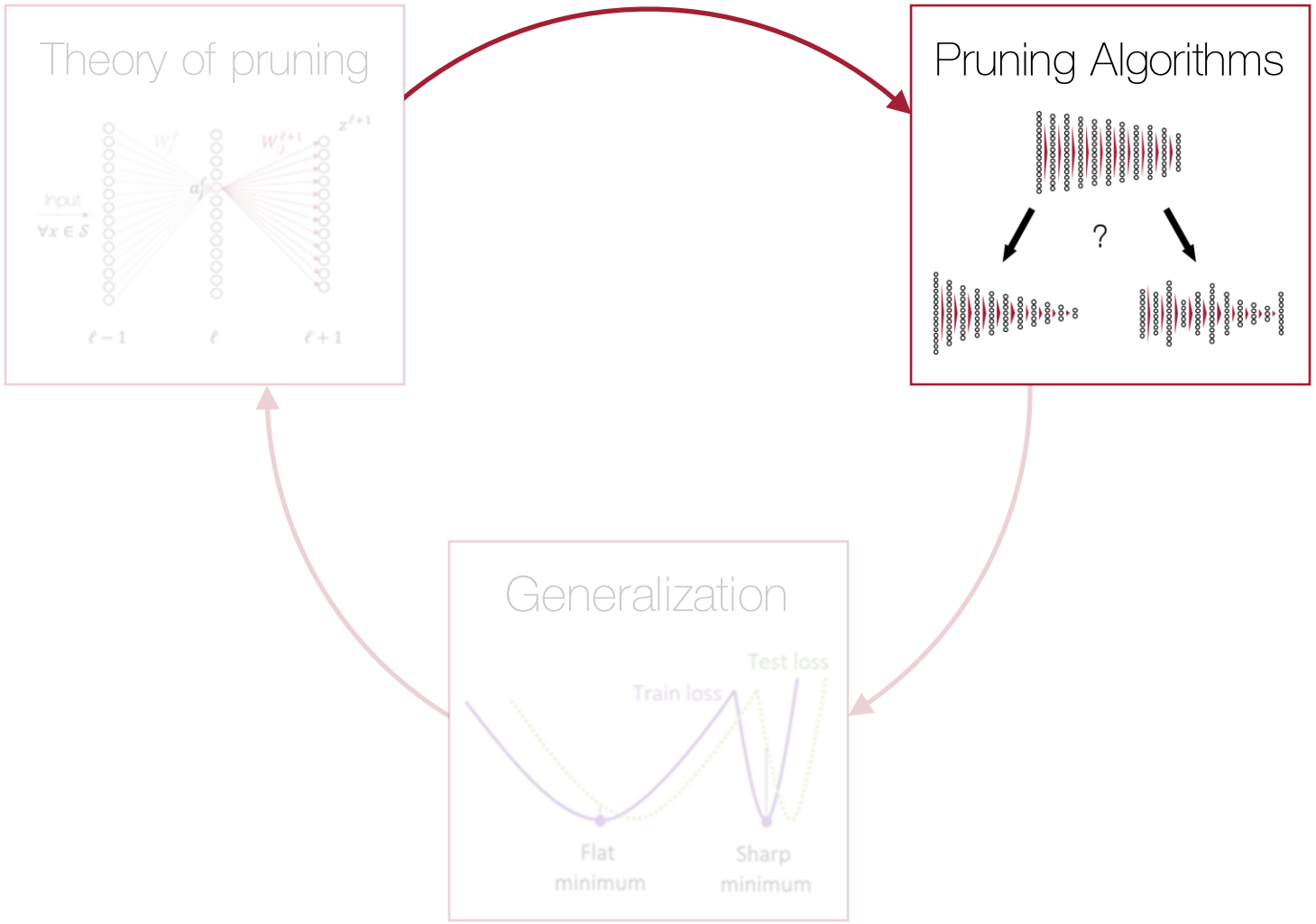
- Ensures desired relative error within layer
- Considers propagation of error across layers

lucasl@mit.edu

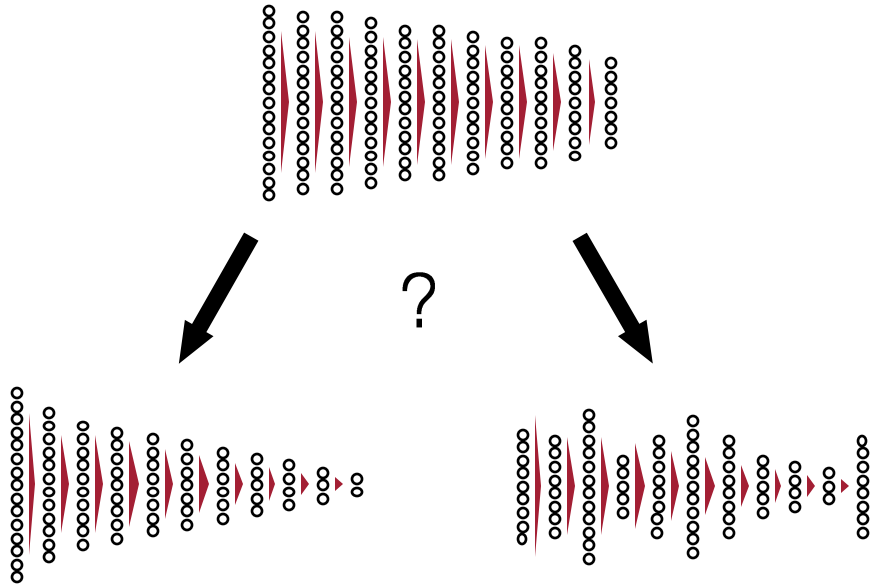
Outline


Theory of pruning
Input
$\forall x \in \mathcal{S}$
W_j^ℓ
a_j^ℓ
$W_{:j}^{\ell+1}$
$z^{\ell+1}$
$\ell - 1$
ℓ
$\ell + 1$
Pruning Algorithms
?
Generalization
Test loss
Train loss
Flat minimum
Sharp minimum

Outline


Theory of pruning
Pruning Algorithms
?
Generalization
Test loss
Train loss
Flat minimum
Sharp minimum

Budget allocation: leveraging bounds in practice



Theorem: Relative error $\epsilon^{\ell} = f(m^{\ell})$ in layer ℓ depends on number of samples m^{ℓ}

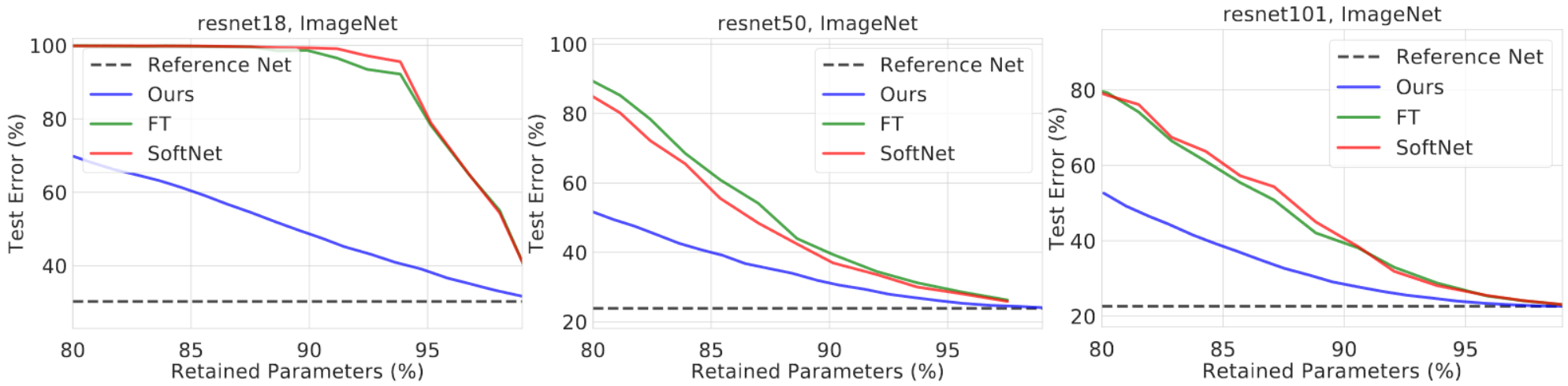
Pruned size $size(\hat{\theta}) = g(m^1, \ldots, m^L)$

Allocate budget $\mathcal{B}$:

$$m^1, \ldots, m^L = \operatorname{argmin} \max_{\ell} \epsilon^{\ell}(m^{\ell})$$
$$s.t.\ size(\hat{\theta}) = g(m^1, \ldots, m^L) \leq \mathcal{B}$$

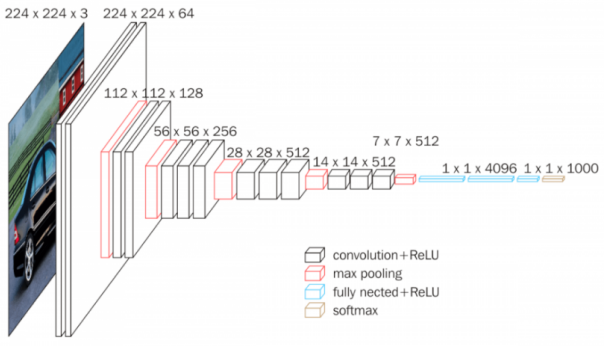
Solve efficiently via binary search

Results: prune-only

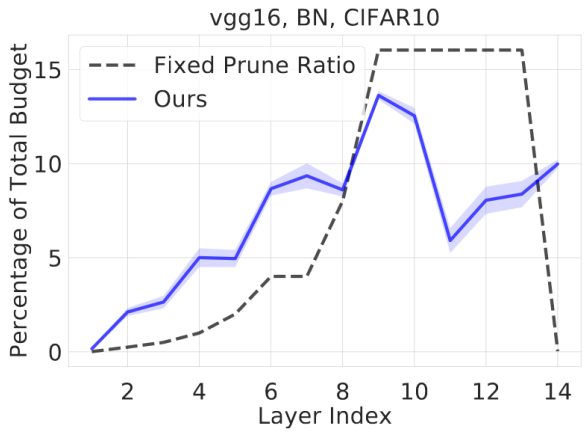


(a) ResNet18 (b) ResNet50 (c) ResNet101

Results: budget allocation



(a) VGG16 architecture



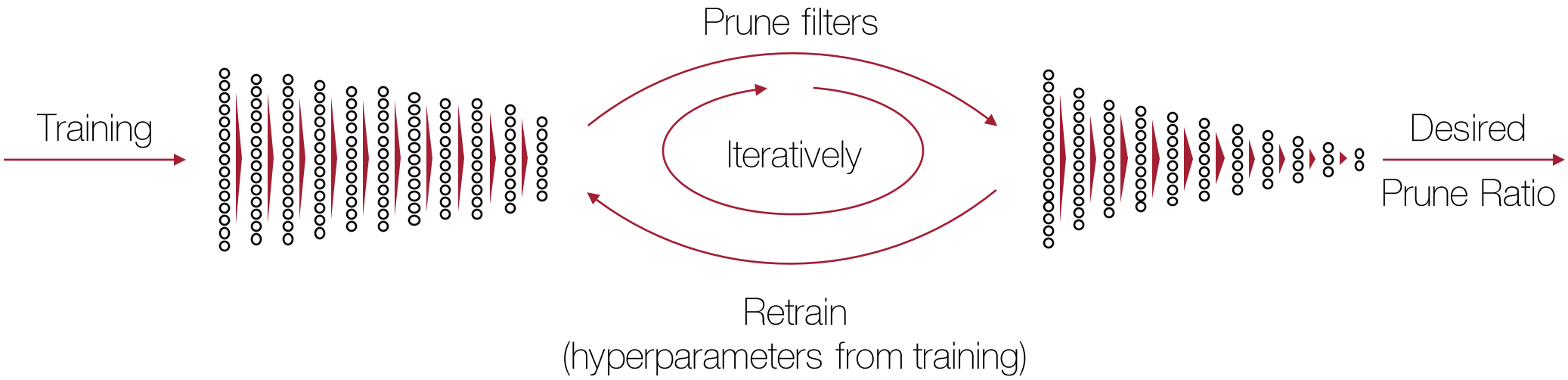
(b) Budget Allocation for VGG16

Early layers are over-sampled
→ small filters, large images

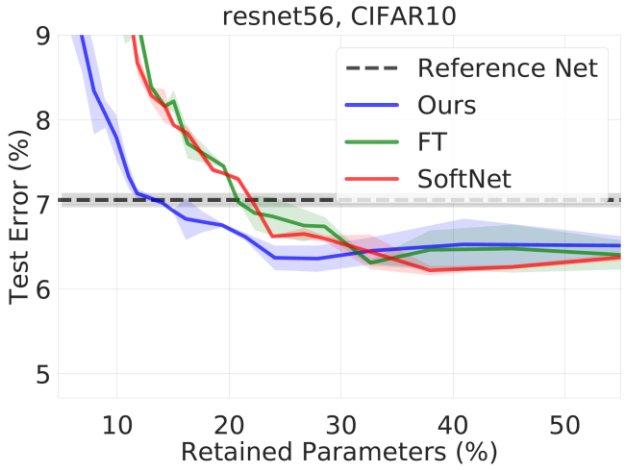
Middle layers are under-sampled
→ large filters, small images

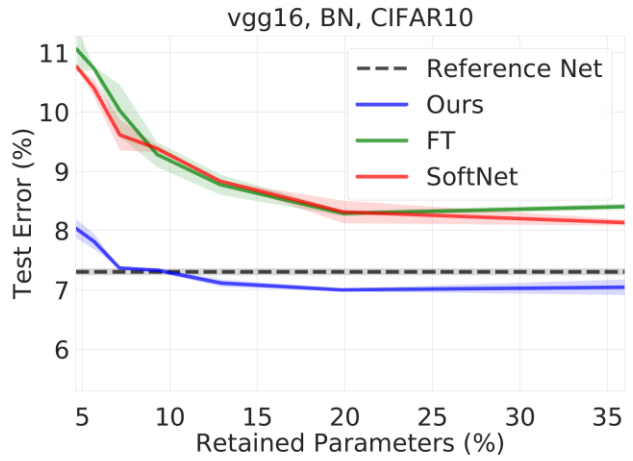
Last layer is over-sampled
→ classification layer!

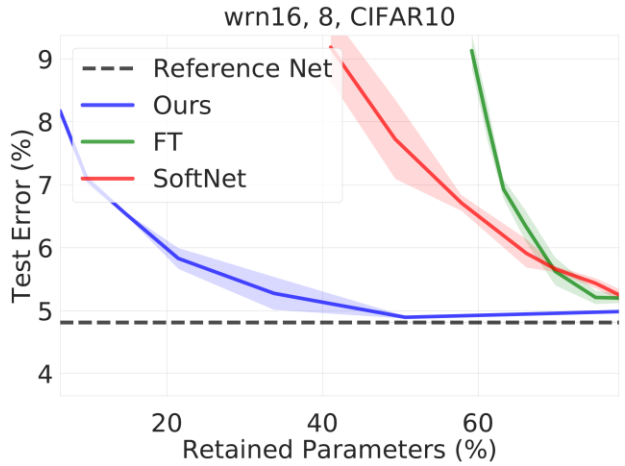
Results: iterative pruning



Results: iterative pruning



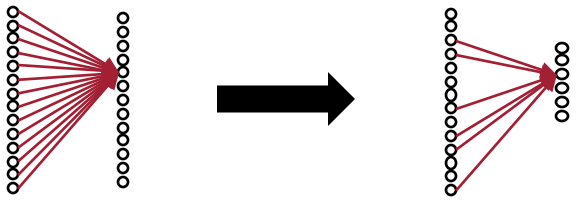




A more generic approach to pruning

1

"Local step"

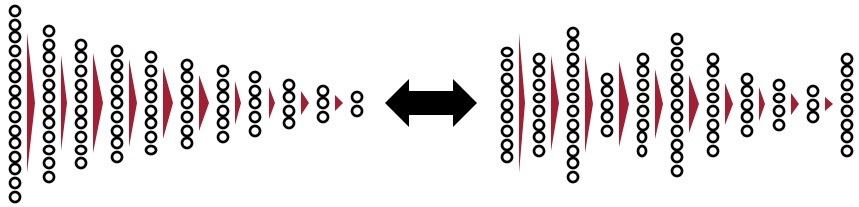


$\varepsilon^{\ell} = \varepsilon^{\ell}(\text{prune\_ratio})$

Efficiently implementable and easy to evaluate

2

"Global step"



$\|f - \hat{f}\| \leq \varepsilon \|f\|$ where $\varepsilon = \varepsilon(\varepsilon^1, \ldots, \varepsilon^L)$

$\text{minimize}\, cost(\varepsilon^1, \ldots, \varepsilon^L)\ s.t.\ size(\hat{\theta}) \leq \mathcal{B}$

Pruning via low-rank decomposition

$f \times c$ $\quad$ $f \times j$ $\quad$ $j \times c$

$$W^{\ell} = \left(\begin{array}{c}\\\\\\\\\\\\\\\\\end{array}\right) \approx \left(\begin{array}{c}\\\\\\\\\\\\\\\\\end{array}\right) * \left(\begin{array}{c}\\\\\end{array}\right) =: \hat{U}^{\ell} * \hat{V}^{\ell}$$

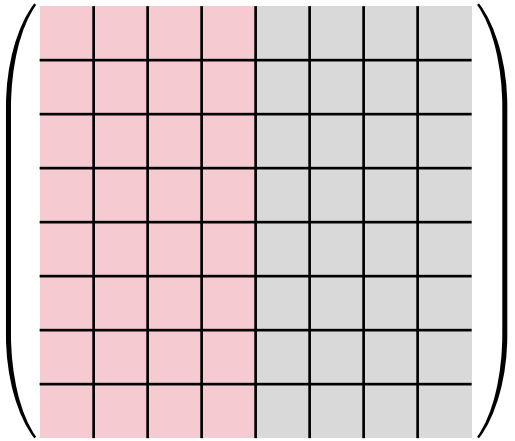
$$\#\, params = fc \longrightarrow \#\, params = j(f + c)$$

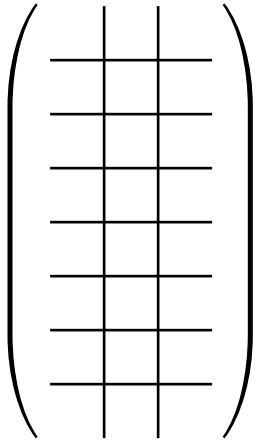
$$\varepsilon^{\ell} = \varepsilon^{\ell}(j) = \varepsilon^{\ell}(\text{"prune\_ratio"})$$

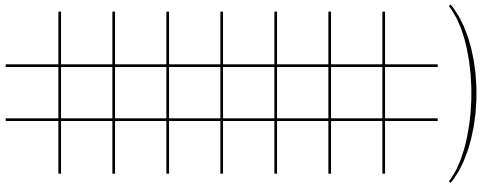
Pruning via low-rank decomposition

$$f \times \frac{c}{k} k \qquad f \times j \qquad j \times c$$

$$W^{\ell} = \left(\right) \approx \left(\right) * \left(\right) =: \hat{U}^{\ell} * \hat{V}^{\ell}$$







$$\#\, params = fc \quad \Longrightarrow \quad \#\, params = j(f + c)$$

$$\varepsilon^{\ell} = \varepsilon^{\ell}(j) = \varepsilon^{\ell}(\text{"prune\_ratio"})$$

Pruning via low-rank decomposition

$$f \times \frac{c}{k} k \qquad f \times jk \qquad k * \left(j \times \frac{c}{k}\right)$$

$$W^{\ell} = \left(\quad \right) \approx \left(\quad \right) * \begin{pmatrix} \\ \end{pmatrix} =: \hat{U}^{\ell} * \hat{V}^{\ell}$$

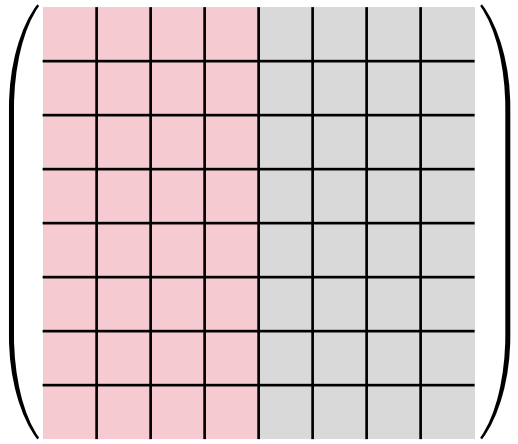
$$\#\, params = fc \longrightarrow \#\, params = j(f + c)$$

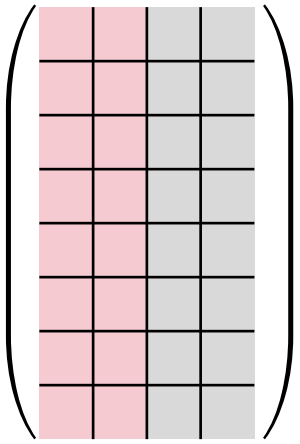
$$\varepsilon^{\ell} = \varepsilon^{\ell}(j) = \varepsilon^{\ell}(\text{"prune\_ratio"})$$

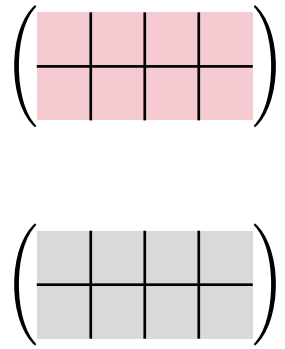
Pruning via low-rank decomposition

$$f \times \frac{c}{k} k \qquad f \times jk \qquad k * \left(j \times \frac{c}{k}\right)$$

$$W^{\ell} = \left(\right) \approx \left(\right) * \begin{pmatrix} \\ \end{pmatrix} =: \hat{U}^{\ell} * \hat{V}^{\ell}$$







$$\#\, params = fc \longrightarrow \#\, params = j(fk + c)$$

$$\varepsilon^{\ell} = \varepsilon^{\ell}(j) = \varepsilon^{\ell}(\text{"prune\_ratio"})$$

Pruning via low-rank decomposition

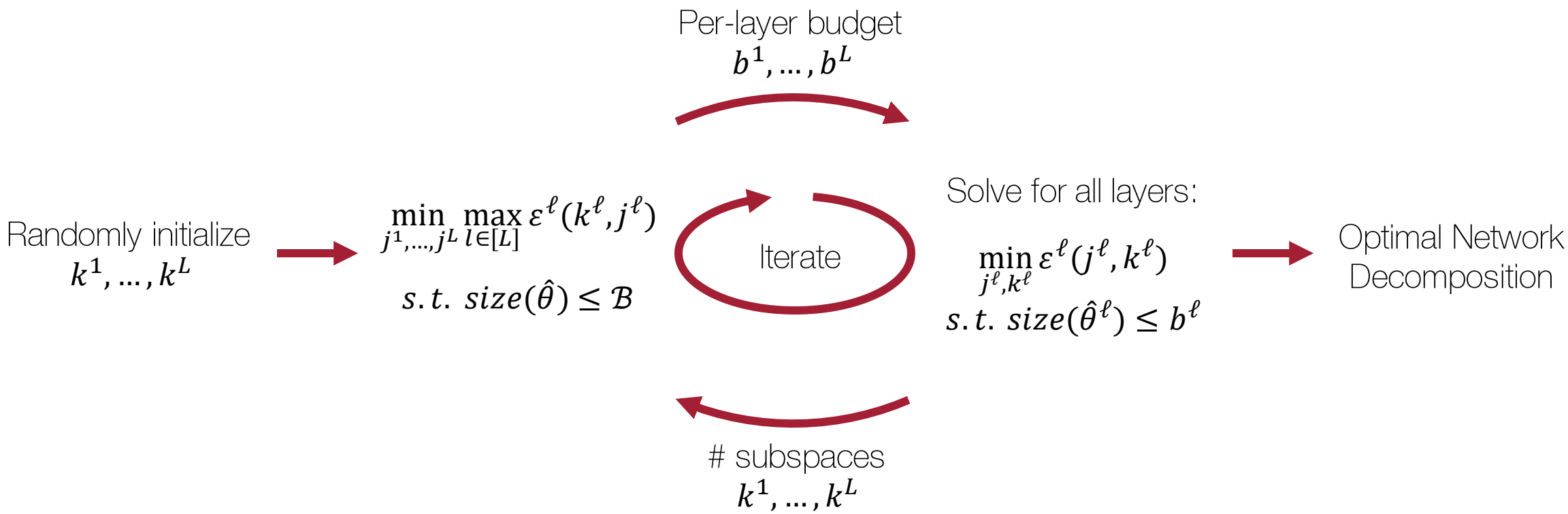
$$f \times \frac{c}{k} k \qquad f \times jk \qquad k * \left(j \times \frac{c}{k}\right)$$

$$W^{\ell} = \left(\right) \approx \left(\right) * \left(\right) =: \hat{U}^{\ell} * \hat{V}^{\ell}$$

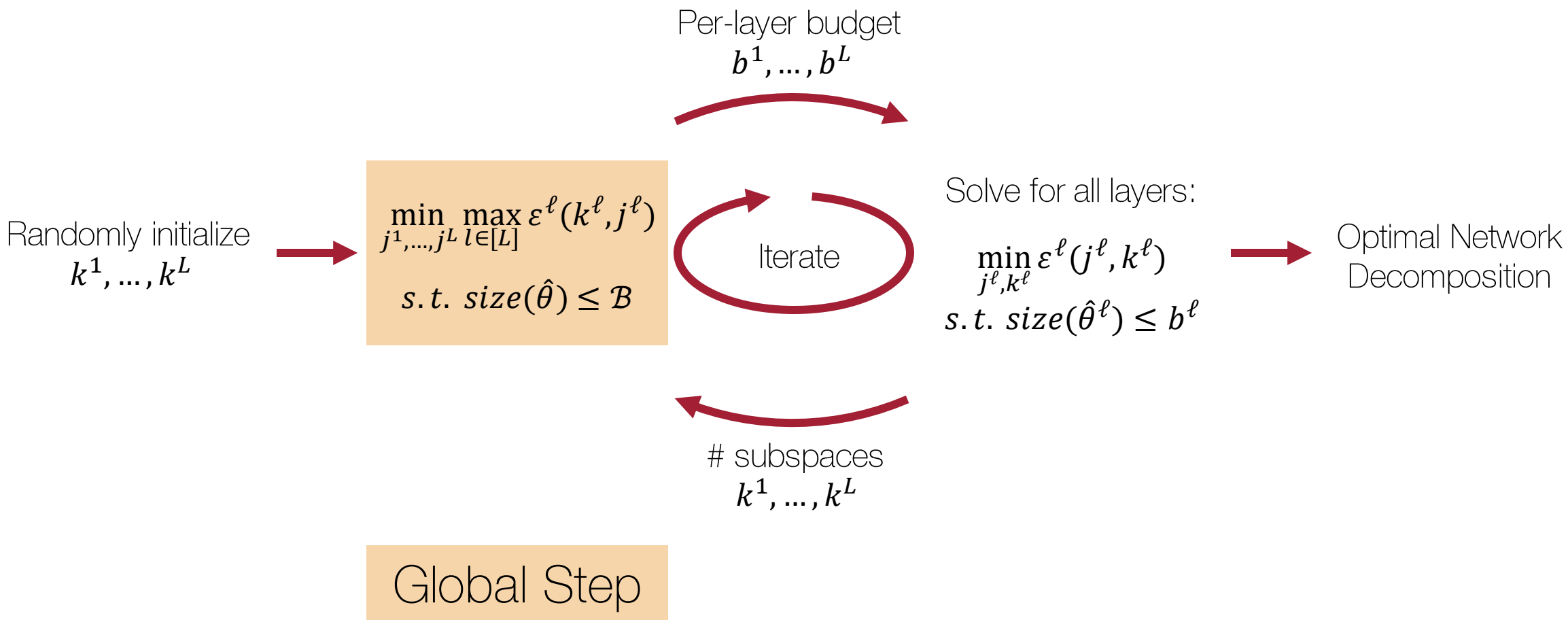
$$\#\, params = fc \longrightarrow \#\, params = j(fk + c)$$

$$\varepsilon^{\ell} = \varepsilon^{\ell}(j, k) = \varepsilon^{\ell}(\text{"prune\_ratio"})$$

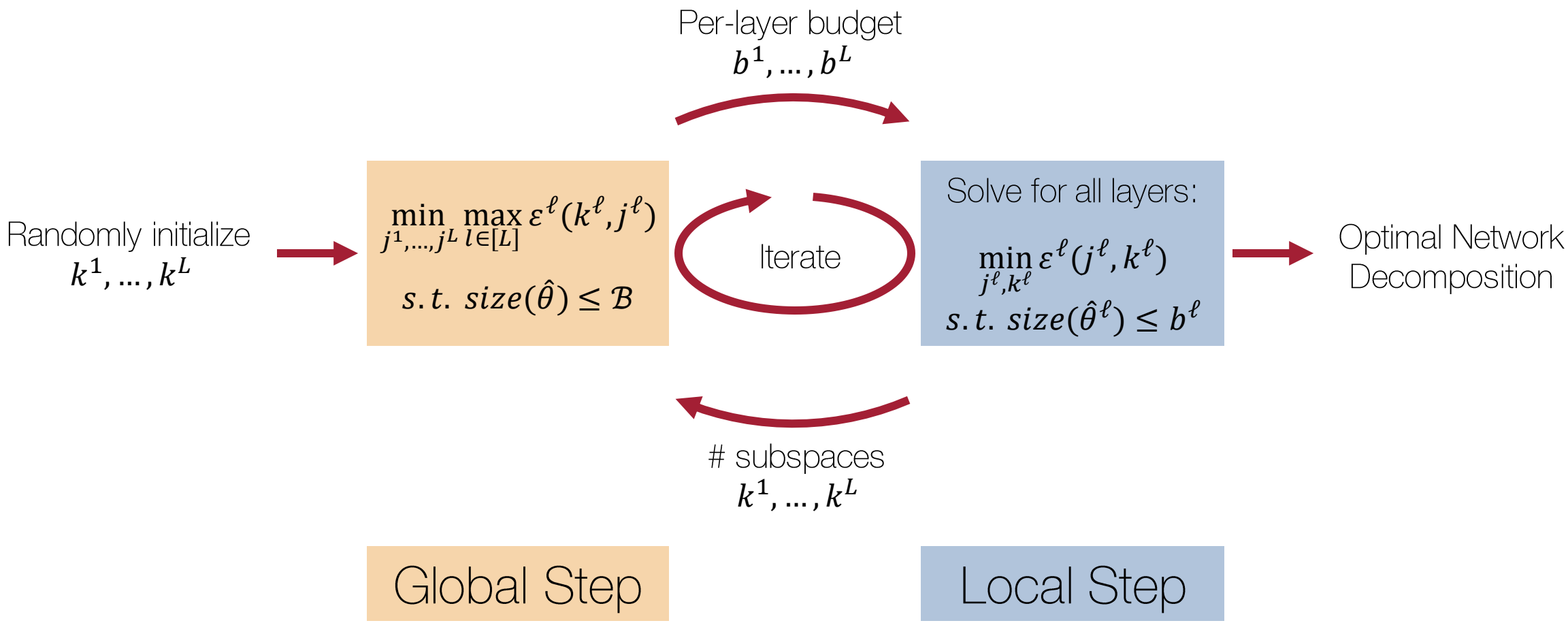
Pruning via low-rank decomposition



Pruning via low-rank decomposition



Pruning via low-rank decomposition

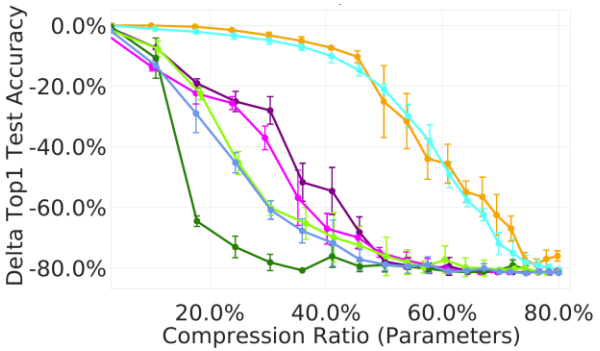


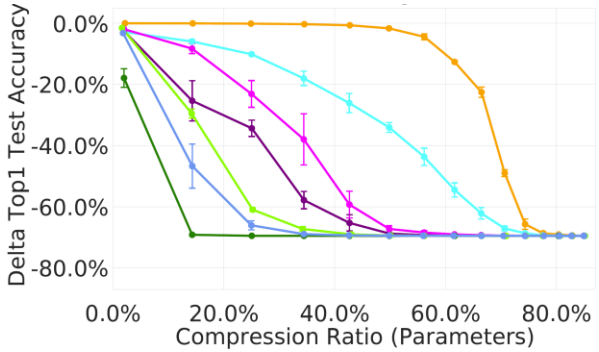
Pruning via low-rank decomposition: results

ResNet20 CIFAR10

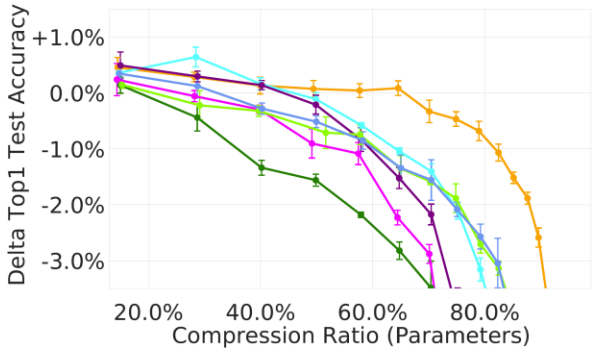
ResNet18 ImageNet

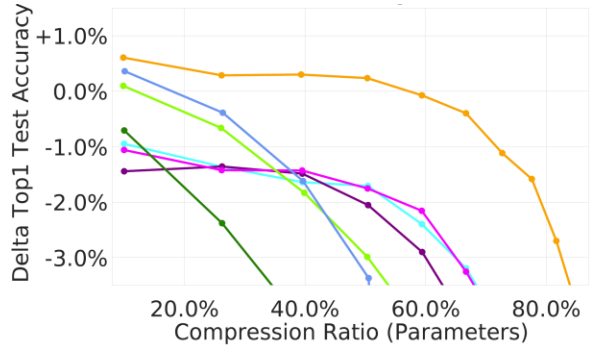
Prune-only



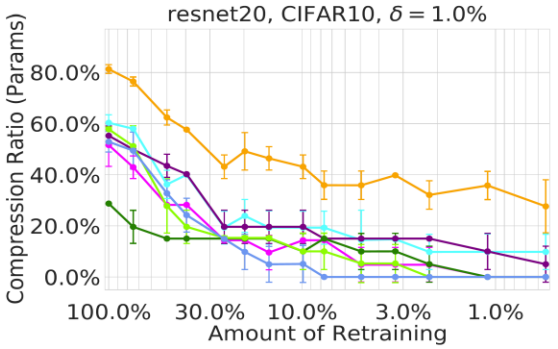


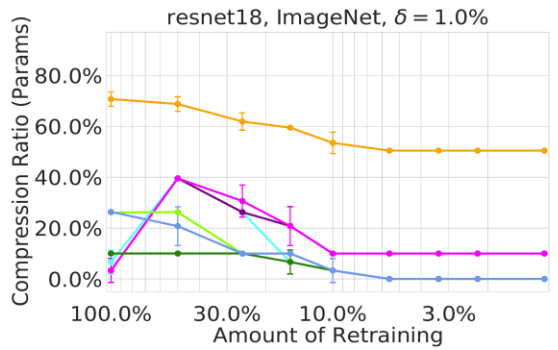
Retrain





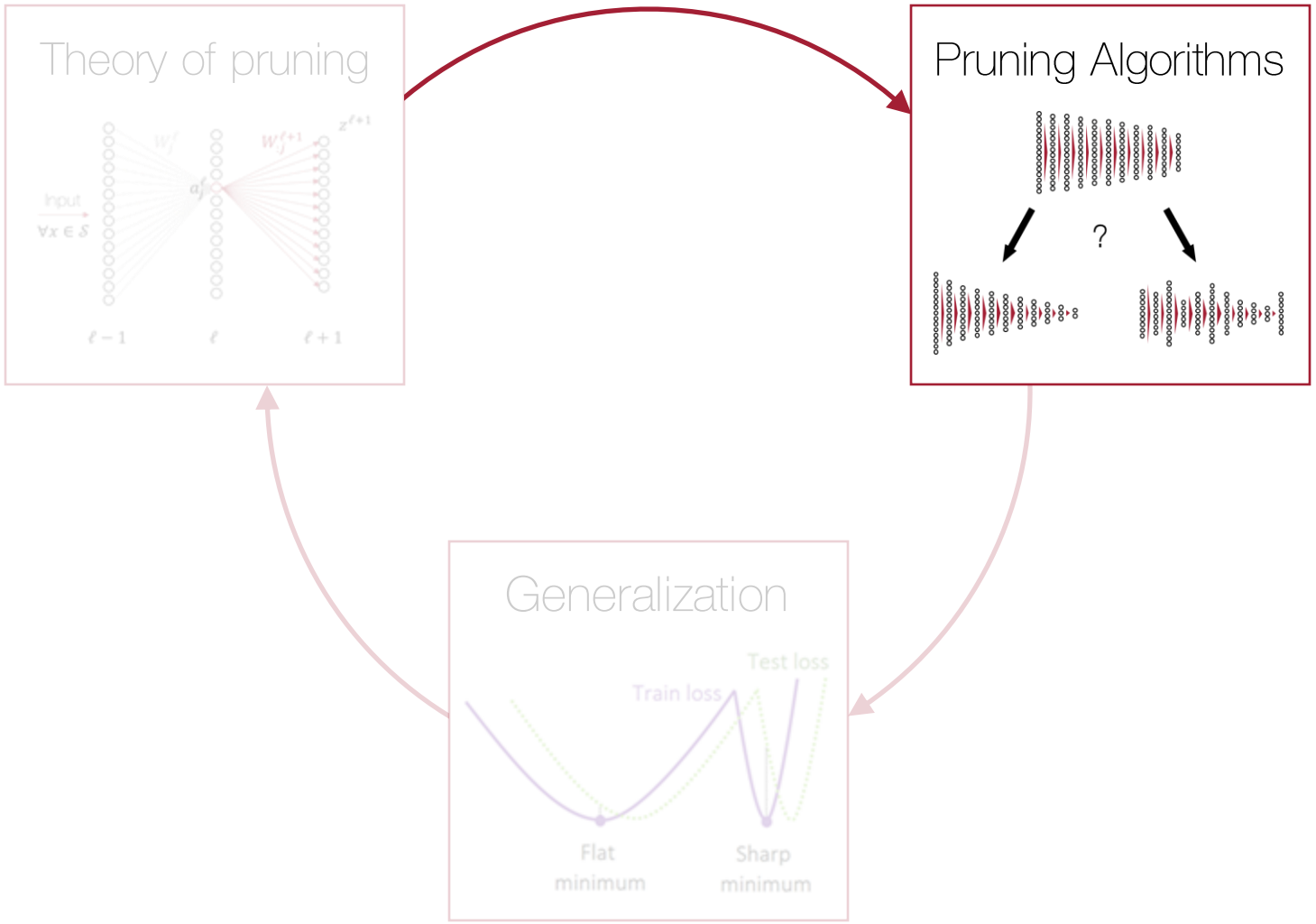
Sweep



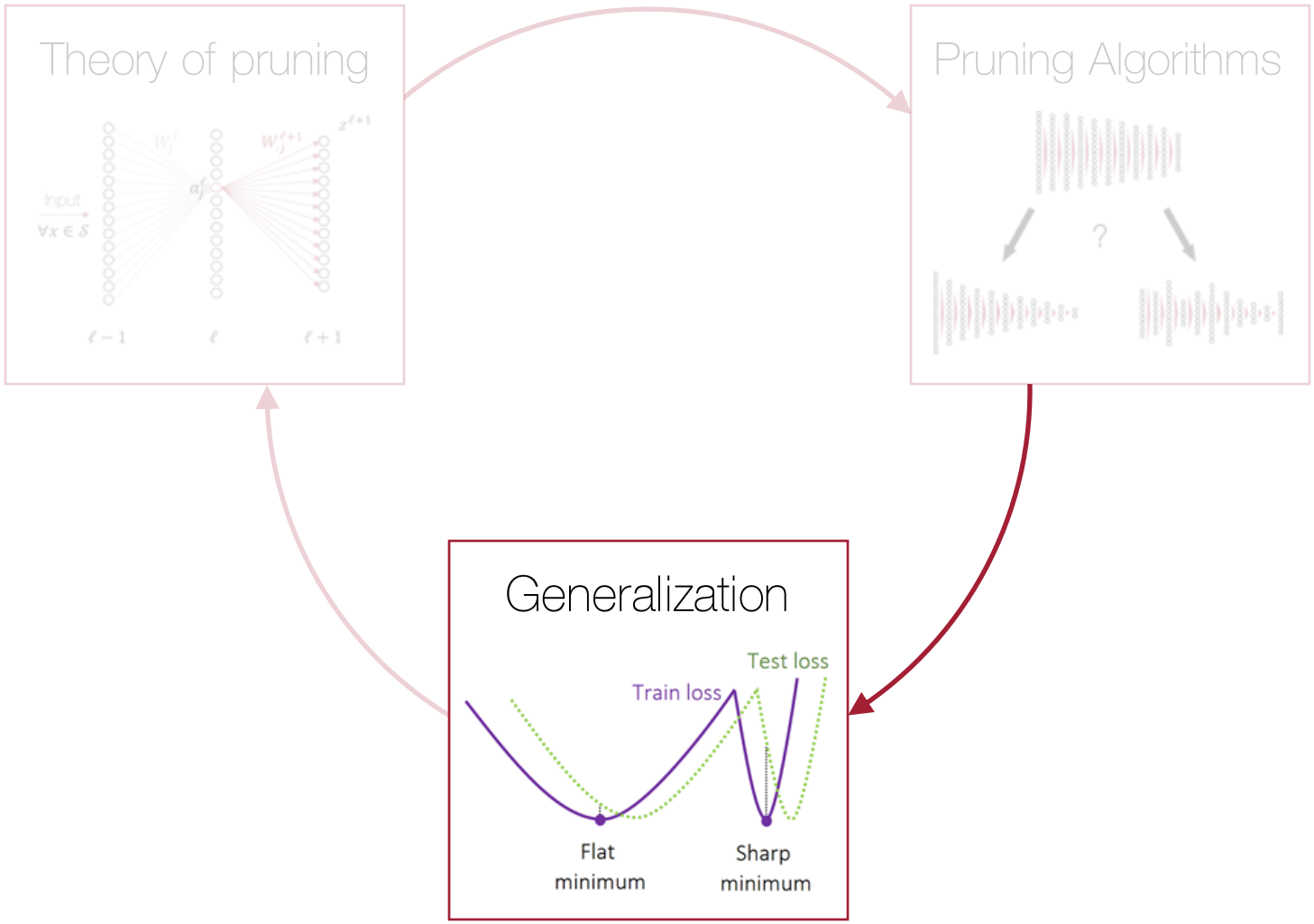


ALDS (Ours) PCA SVD-Energy SVD L-Rank FT PFP

Outline


Theory of pruning
Pruning Algorithms
?
Generalization
Test loss
Train loss
Flat minimum
Sharp minimum

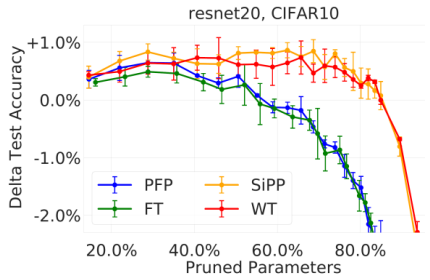
Outline


Theory of pruning
Pruning Algorithms
?
Generalization
Test loss
Train loss
Flat minimum
Sharp minimum

What are the effects of pruning?

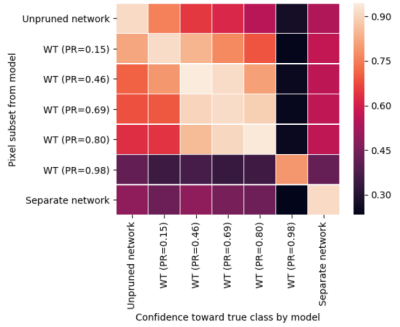
1

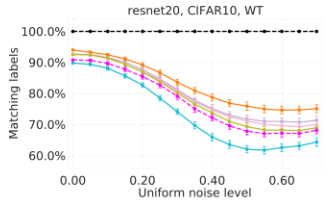
How we prune?



2

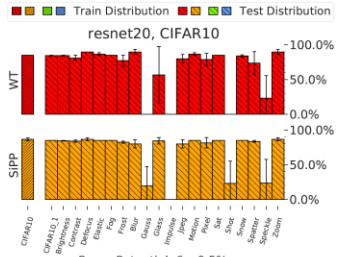
What is preserved?



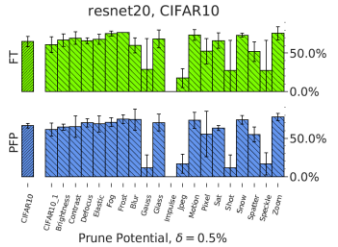


3

What is *lost*?



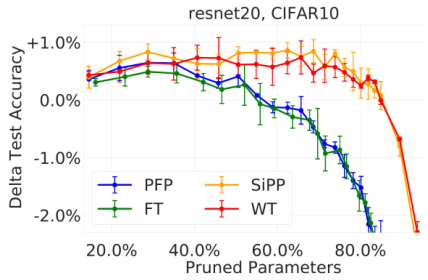
(b) Weight prune potential



What are the effects of pruning?

1

How we prune?



2

What is preserved?

3

What is *lost*?

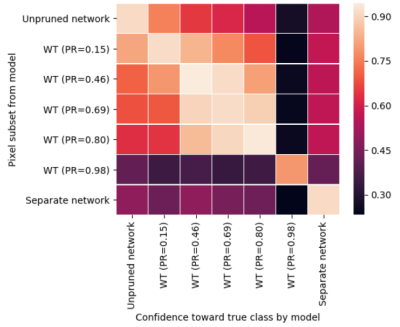
What are the effects of pruning?

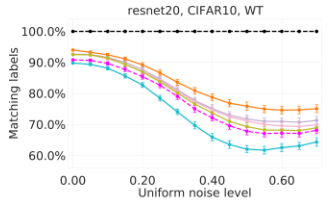
1

How we prune?

2

What is preserved?

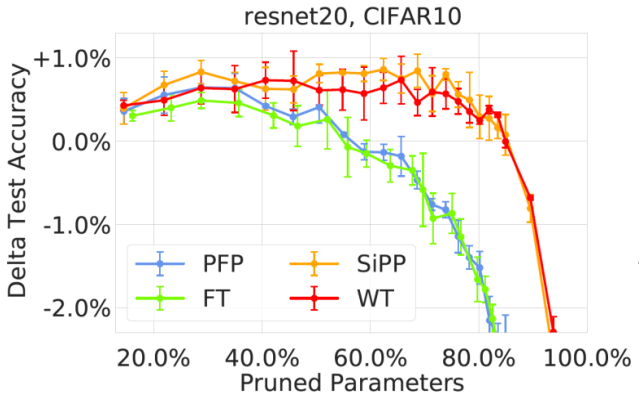




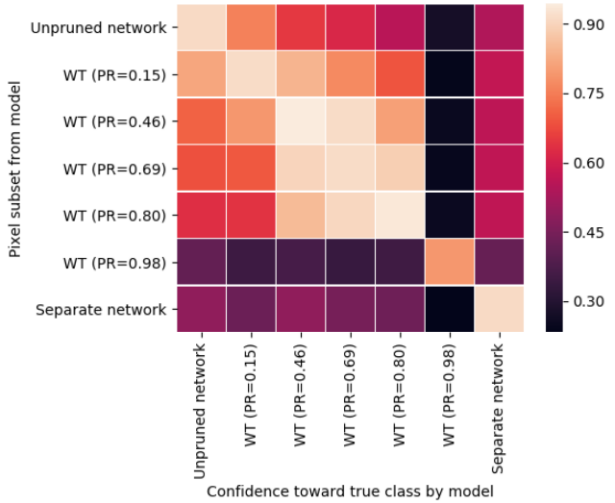
3

What is *lost*?

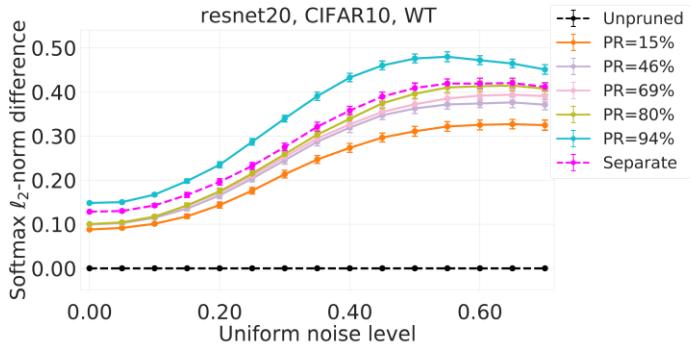
What is preserved during pruning?



Test accuracy

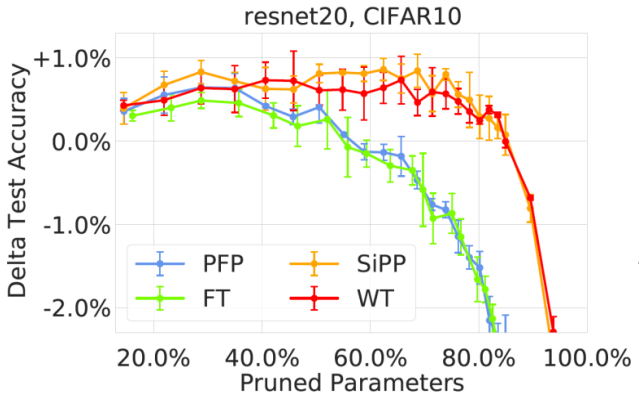


Informative features

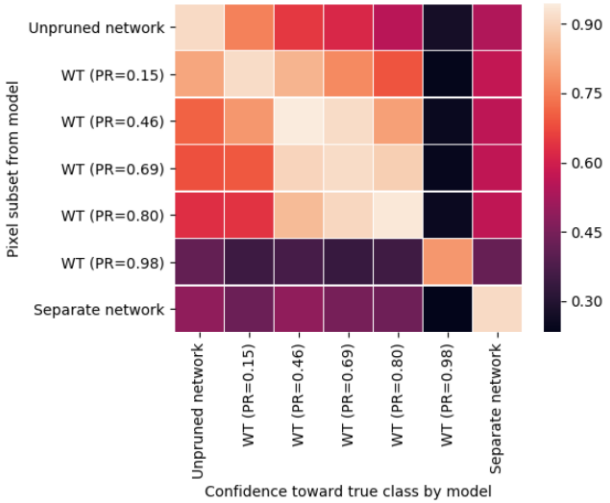


Functional similarities

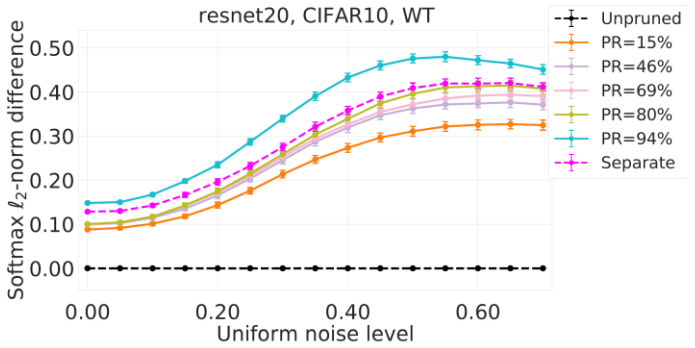
What is preserved during pruning?



Test accuracy

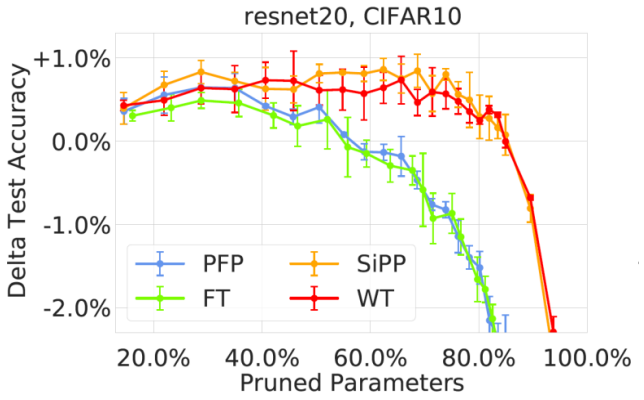


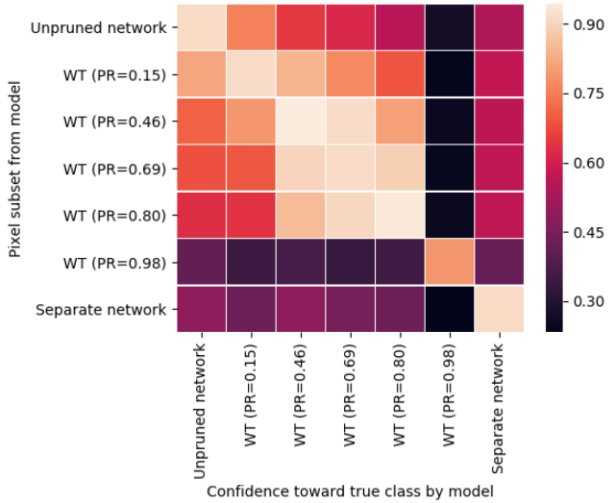
Informative features

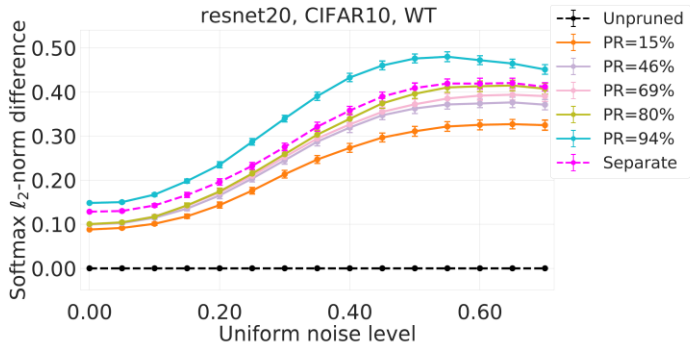


Functional similarities

What is preserved during pruning?





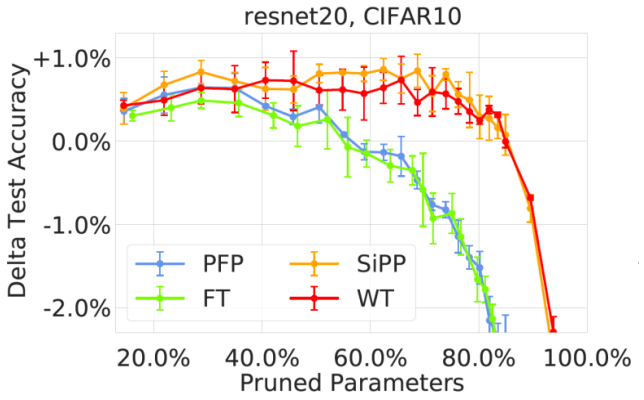


- Test accuracy

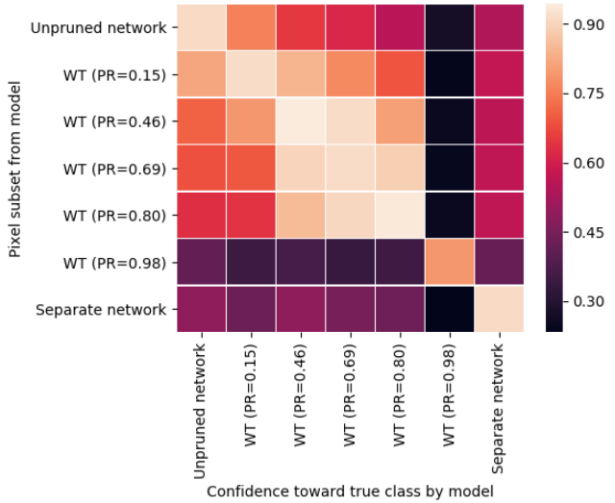
- Informative features

- Functional similarities

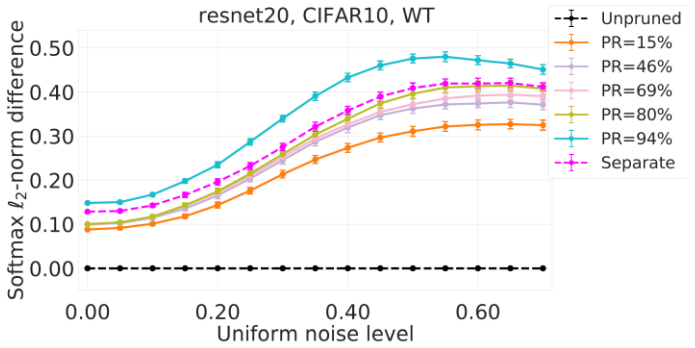
What is preserved during pruning?



- Test accuracy



- Informative features



- Functional similarities

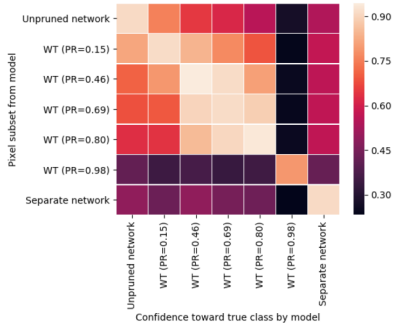
What are the effects of pruning?

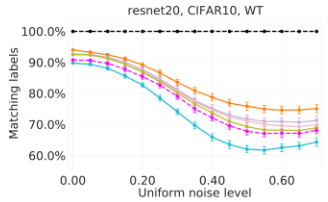
1

How we prune?

2

What is preserved?





3

What is *lost*?

What are the effects of pruning?

1

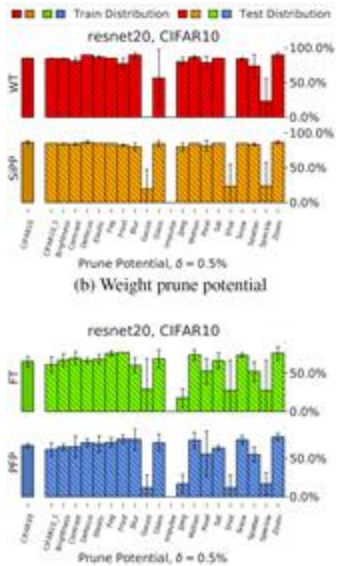
How we prune?

2

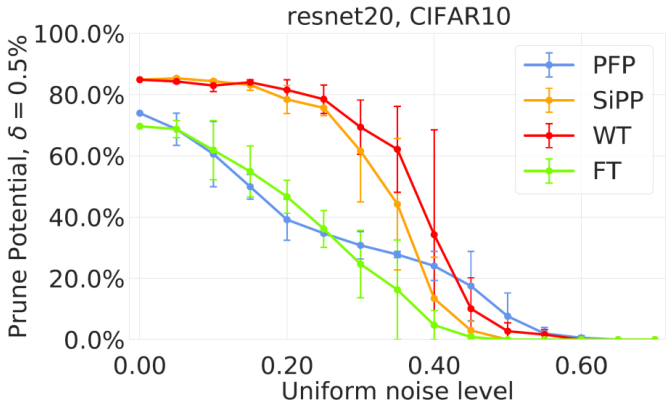
What is preserved?

3

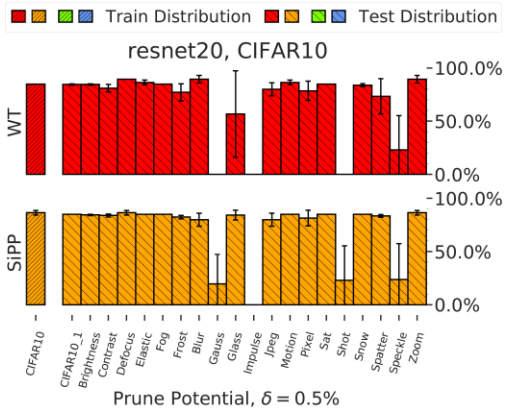
What is *lost*?



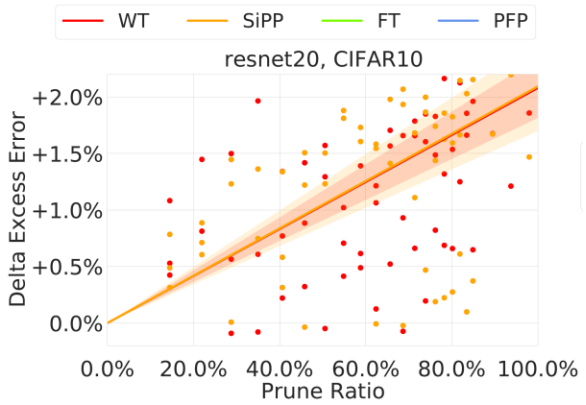
What is *lost* during pruning?



Prune potential for noise



Prune potential for corruptions



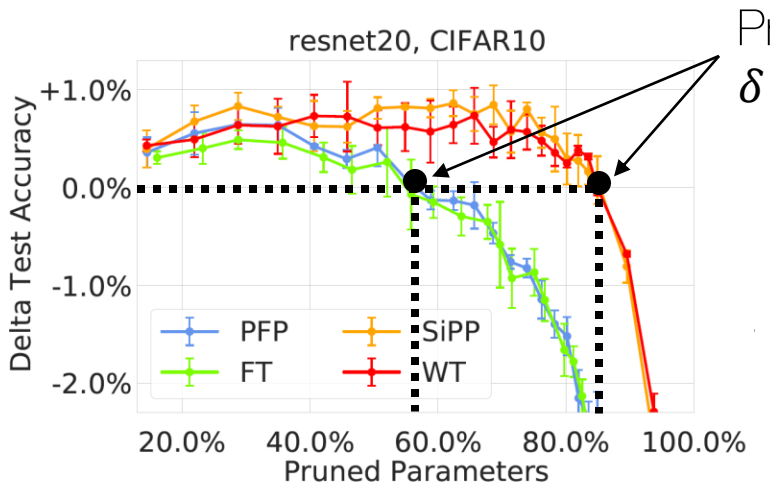
Adequate excess error

Prune potential

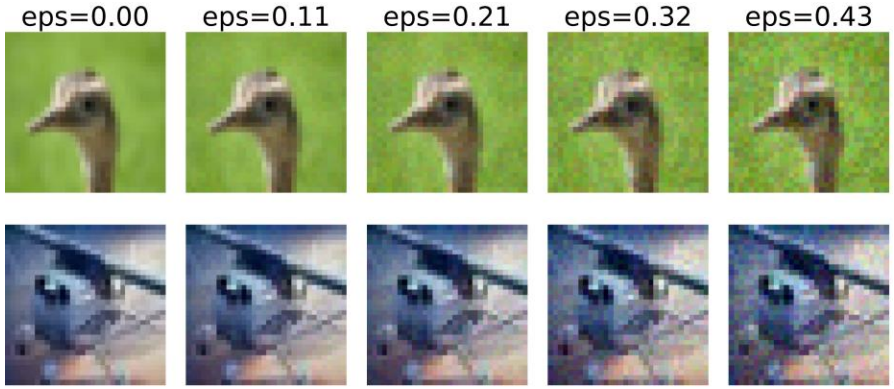
For $\delta \in [0,1)$, given network f_θ, and data distribution $\mathcal{D}$ the prune potential $P(\theta, \mathcal{D})$ is defined as

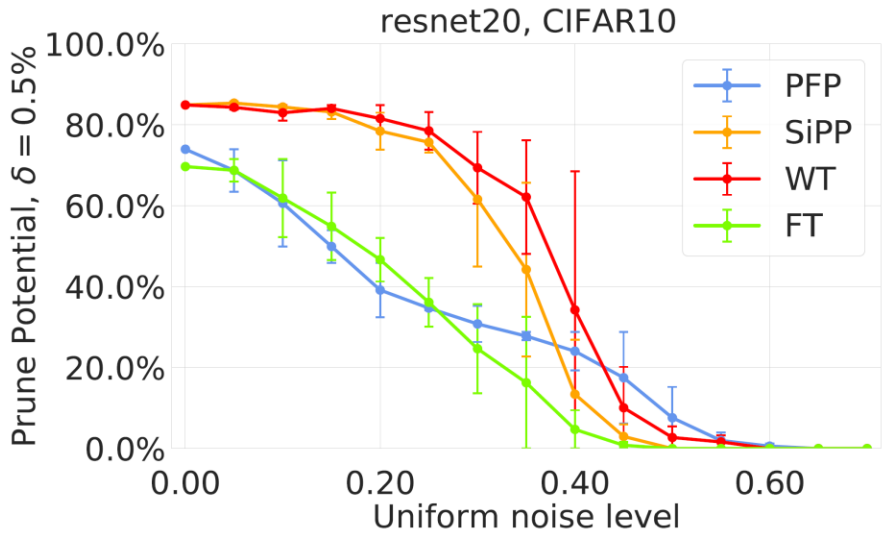
$$P(\theta, \mathcal{D}) \coloneqq \text{"}\textit{maximum prune ratio with accuracy loss at most } \delta\text{"}$$

- Quantifies "overparameterization" of network
- Approximated using many prune-retrain cycles



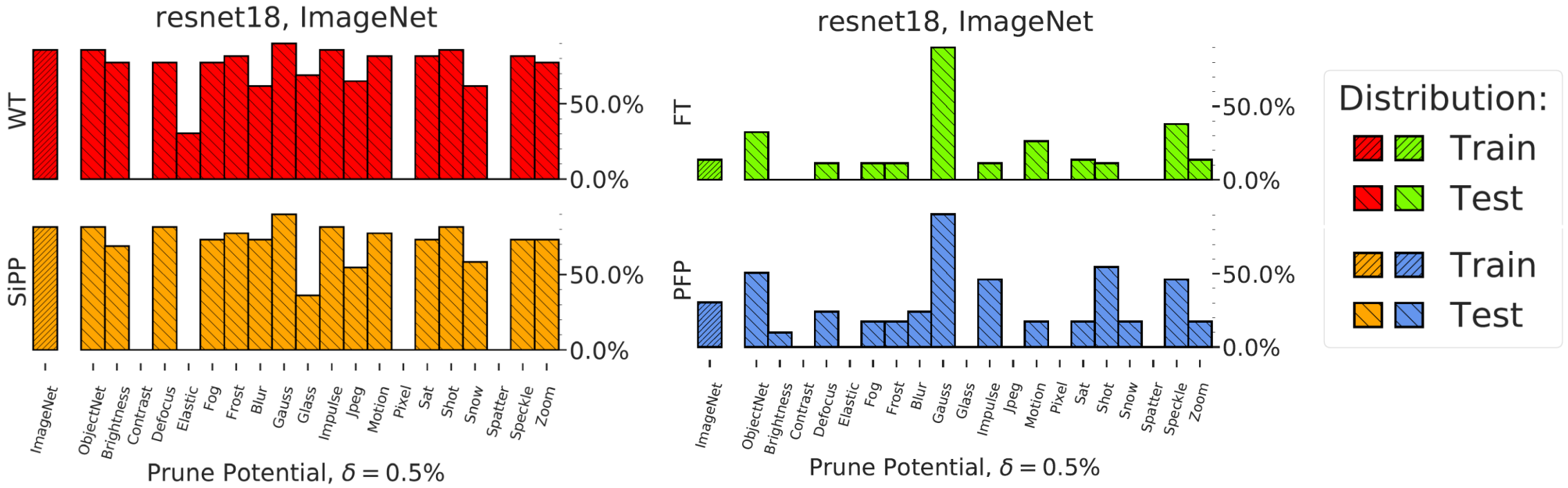
Prune potential for noise





- Prune potential after small noise injections in input
- Pruned networks are more affected by out-of-distribution
- “Robust overparameterization” vs. “nominal overparameterization”

Prune potential for corruptions



- Significantly reduced prune potential under distribution changes (!)

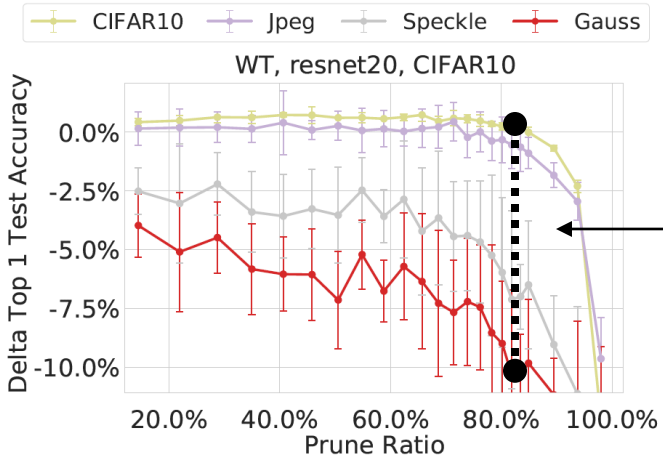
Excess error

For a network f_θ, train distribution $\mathcal{D}$, and *test distribution* $\mathcal{D}'$ the excess error is defined as

$$e(\theta, \mathcal{D}') := \text{"additional error incurred on test distribution"}$$

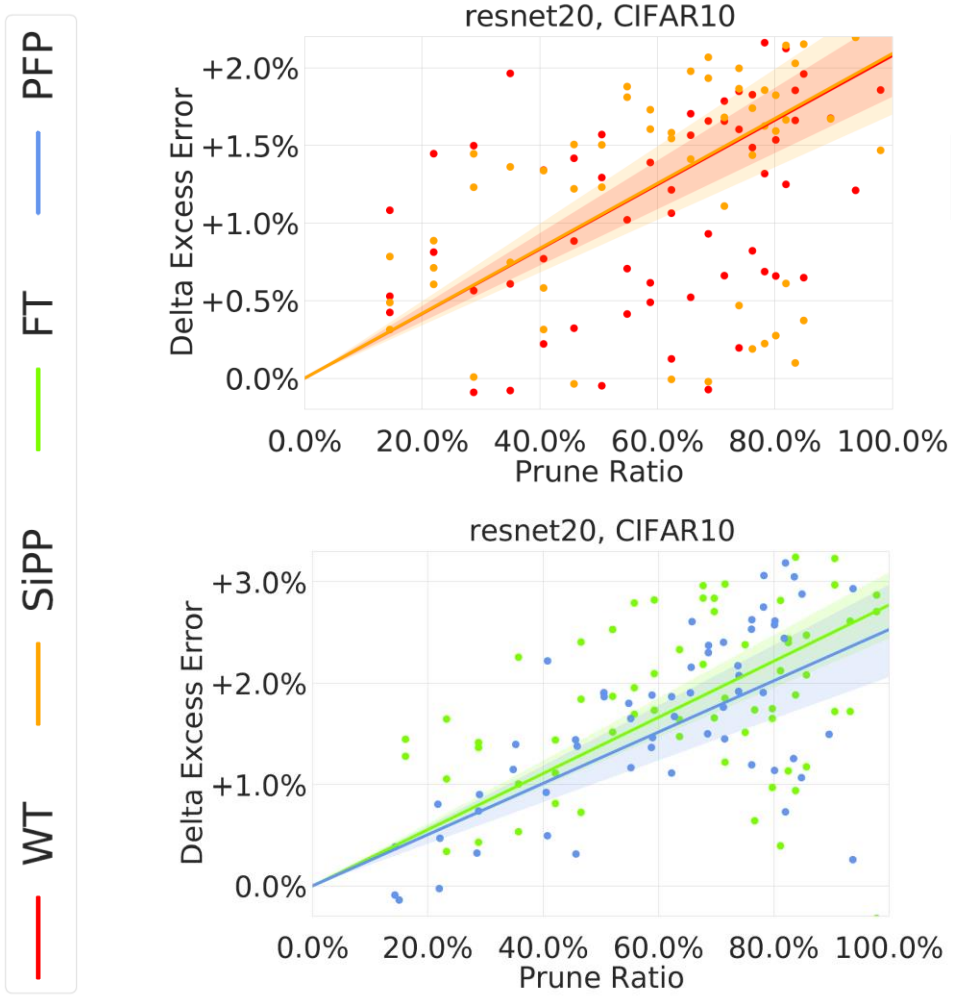
Quantifies performance drop for distribution changes

Difference in excess error for pruned and unpruned network indicates performance drop



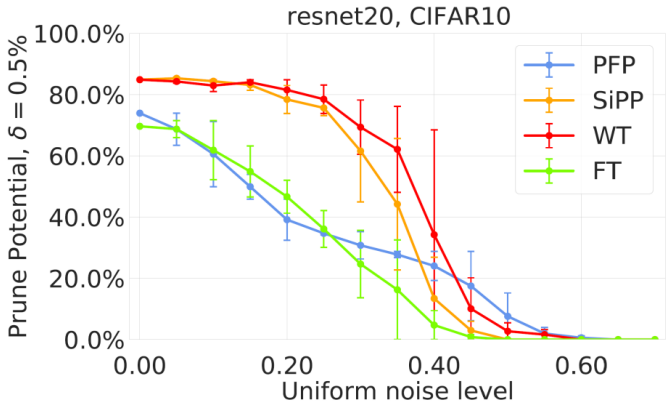
Excess error for "Gauss" test distribution and 80% prune ratio

Excess error

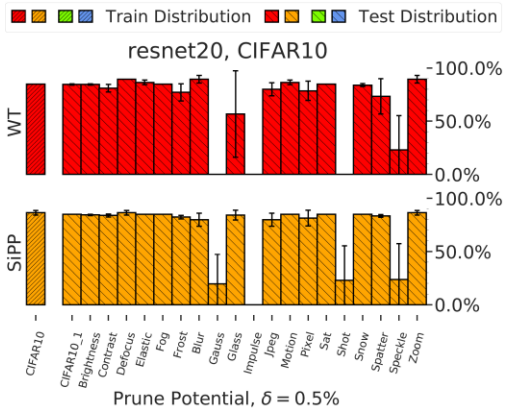


- Excess error averaged over multiple corruptions
- Pruned networks exhibit higher excess error
- Nominal prune curve is not indicative of out-of-distribution performance!

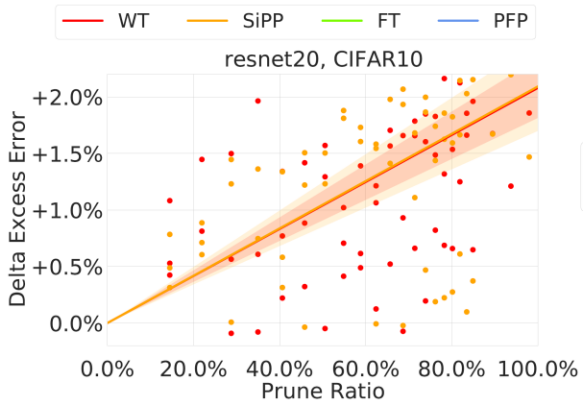
What is *lost* during pruning?



Prune potential for noise

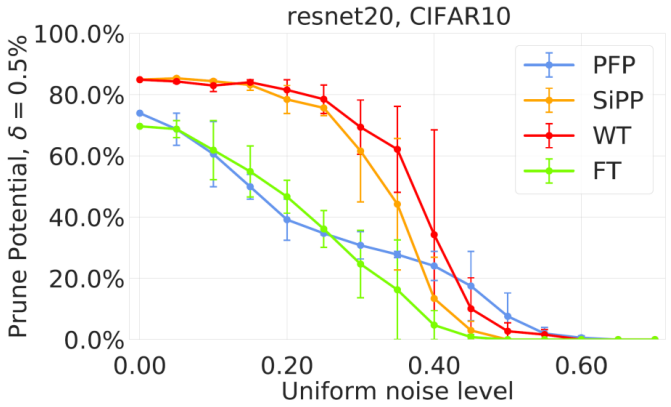


Prune potential for corruptions

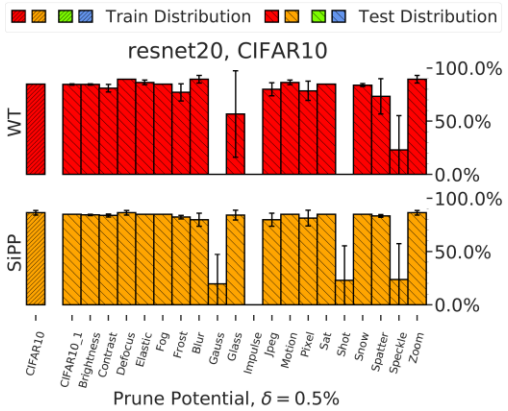


Adequate excess error

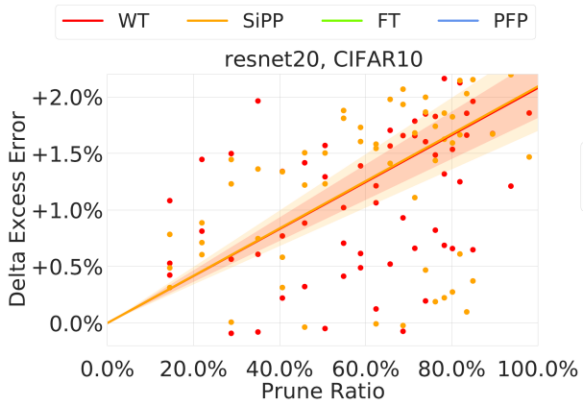
What is *lost* during pruning?



Prune potential for noise

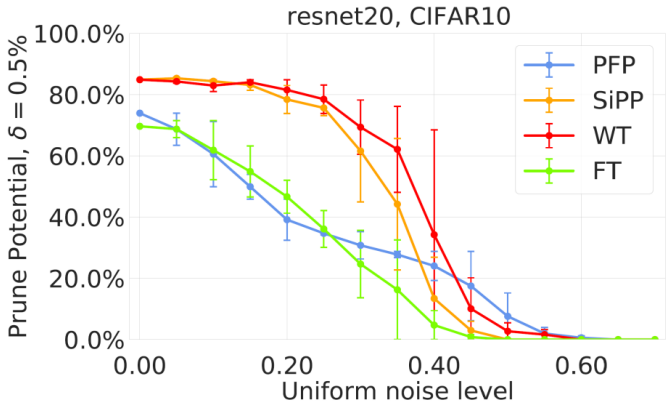


Prune potential for corruptions

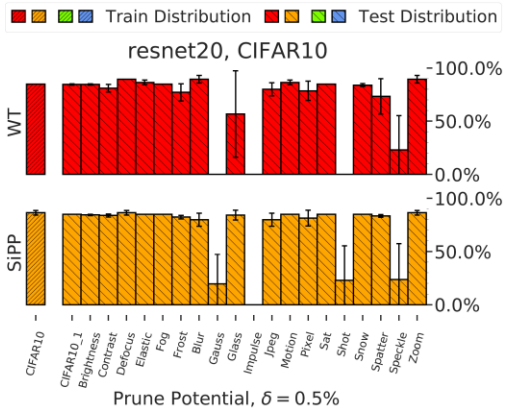


Adequate excess error

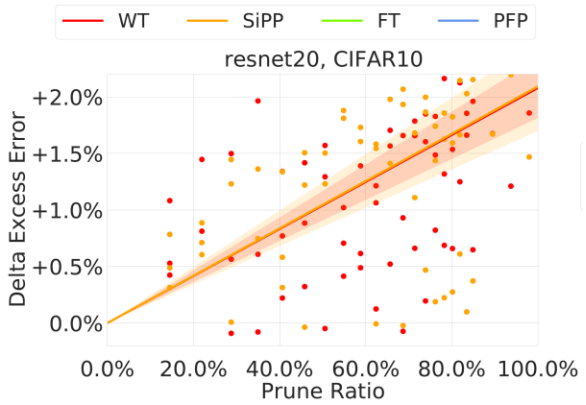
What is *lost* during pruning?



Prune potential for noise

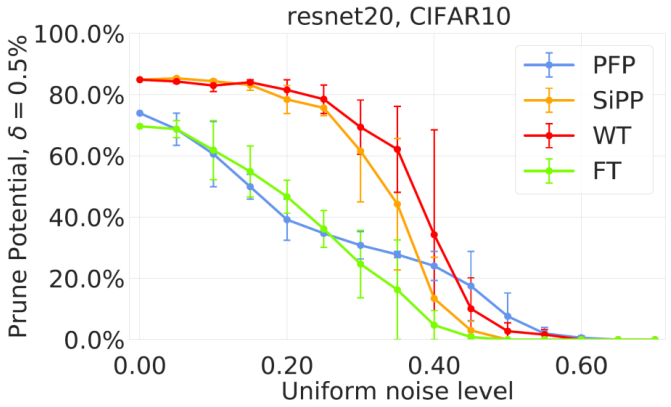


Prune potential for corruptions

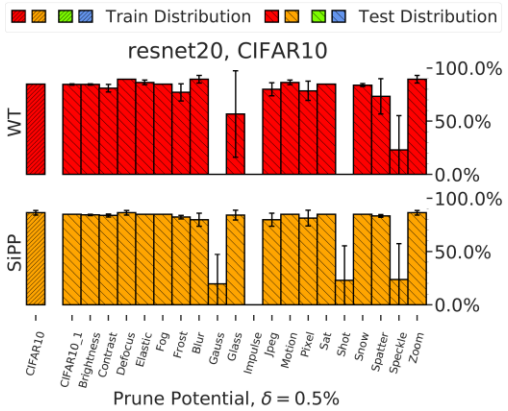


Adequate excess error

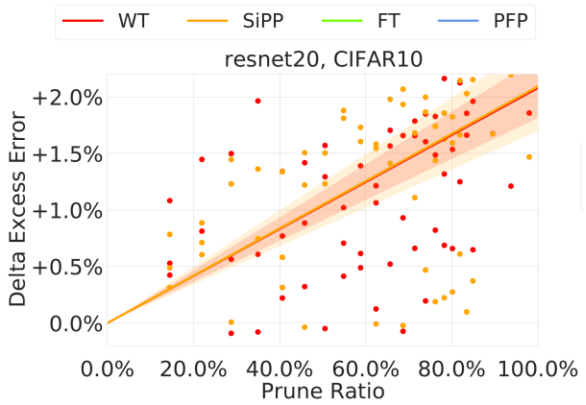
What is *lost* during pruning?



Prune potential for noise



Prune potential for corruptions



Adequate excess error

Sparse Flow: pruning continuous-depth models

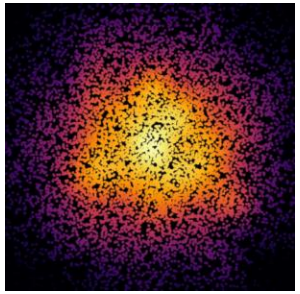
Neural ODEs
(Neural Ordinary Differential Equations)

$$\frac{\partial \boldsymbol{z}(t)}{\partial t} = f(\boldsymbol{z}(t), t, \theta), \text{ where } \boldsymbol{z}(t_0) = \boldsymbol{z}_0$$

CNFs
(Continuous Normalizing Flows)

Base distribution: $\boldsymbol{z}_0 \sim p_{z_0}(\boldsymbol{z}_0)$, Target distribution: $\boldsymbol{z}_n \sim p_{z_n}(\boldsymbol{z}_n)$

Change of variable: $\log p\big(\boldsymbol{z}(t_n)\big) = \log p\big(\boldsymbol{z}(t_0)\big) - \int_{t_0}^{t_1} \mathrm{Tr}\left(\frac{\partial f}{\partial \boldsymbol{z}(t)}\right) dt$



$\boldsymbol{z}_0 \sim p_{z_0}(\boldsymbol{z}_0)$ → Base Distribution

$$\frac{\partial \boldsymbol{z}(t)}{\partial t} = f(\boldsymbol{z}(t), t, \theta)$$

$\boldsymbol{z}_n \sim p_{z_n}(\boldsymbol{z}_n)$ → Target Distribution

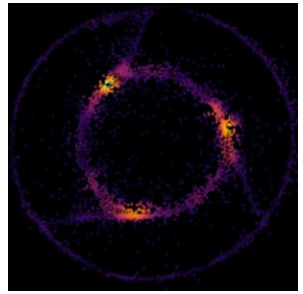


Image credit: Torchdyn code repository, https://github.com/DiffEqML/torchdyn

Sparse Flow: pruning continuous-depth models

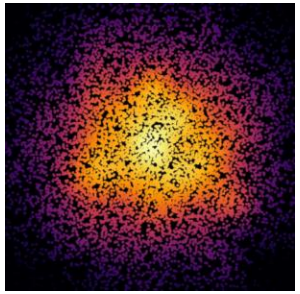
Neural ODEs
(Neural Ordinary Differential Equations)

$$\frac{\partial \boldsymbol{z}(t)}{\partial t} = f(\boldsymbol{z}(t), t, \theta), \text{ where } \boldsymbol{z}(t_0) = \boldsymbol{z}_0$$

CNFs
(Continuous Normalizing Flows)

Base distribution: $\boldsymbol{z}_0 \sim p_{z_0}(\boldsymbol{z}_0)$, Target distribution: $\boldsymbol{z}_n \sim p_{z_n}(\boldsymbol{z}_n)$

Change of variable: $\log p\big(\boldsymbol{z}(t_n)\big) = \log p\big(\boldsymbol{z}(t_0)\big) - \int_{t_0}^{t_1} \mathrm{Tr}\left(\frac{\partial f}{\partial \boldsymbol{z}(t)}\right) dt$



$\boldsymbol{z}_0 \sim p_{z_0}(\boldsymbol{z}_0)$ → Base Distribution

$$\frac{\partial \boldsymbol{z}(t)}{\partial t} = f(\boldsymbol{z}(t), t, \theta)$$

$\boldsymbol{z}_n \sim p_{z_n}(\boldsymbol{z}_n)$ → Target Distribution

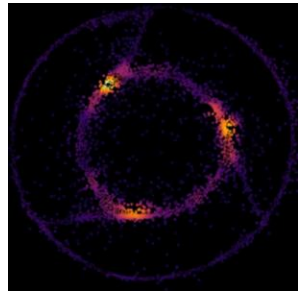


Image credit: Torchdyn code repository, https://github.com/DiffEqML/torchdyn

Sparse Flow: pruning continuous-depth models

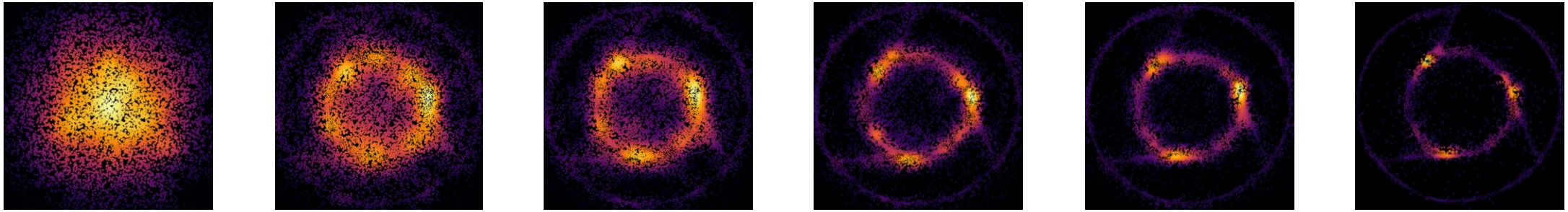
Neural ODEs
(Neural Ordinary Differential Equations)

$$\frac{\partial \boldsymbol{z}(t)}{\partial t} = f(\boldsymbol{z}(t), t, \theta), \text{ where } \boldsymbol{z}(t_0) = \boldsymbol{z_0}$$

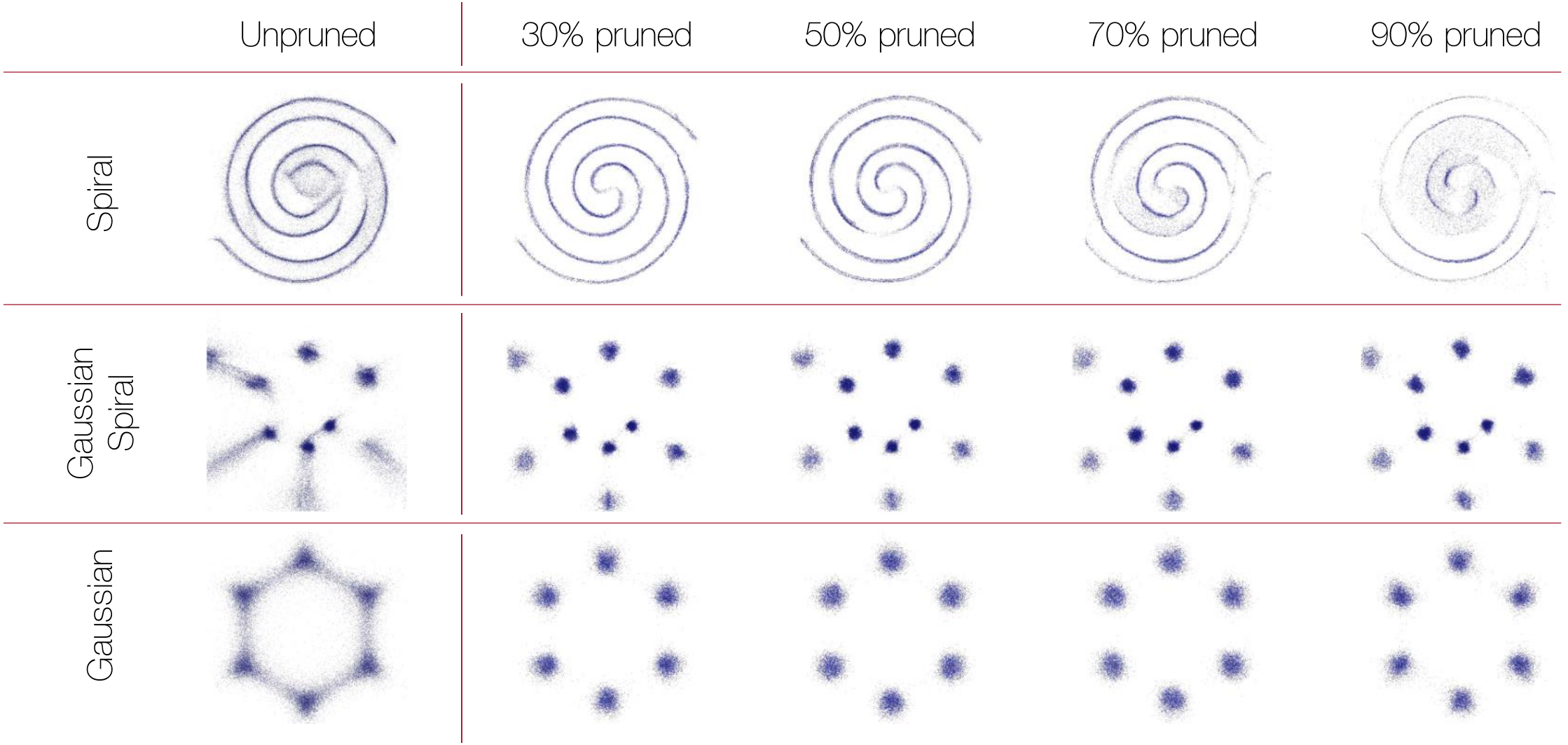
CNFs
(Continuous Normalizing Flows)

Base distribution: $\boldsymbol{z_0} \sim p_{z_0}(\boldsymbol{z_0})$, Target distribution: $\boldsymbol{z_n} \sim p_{z_n}(\boldsymbol{z_n})$

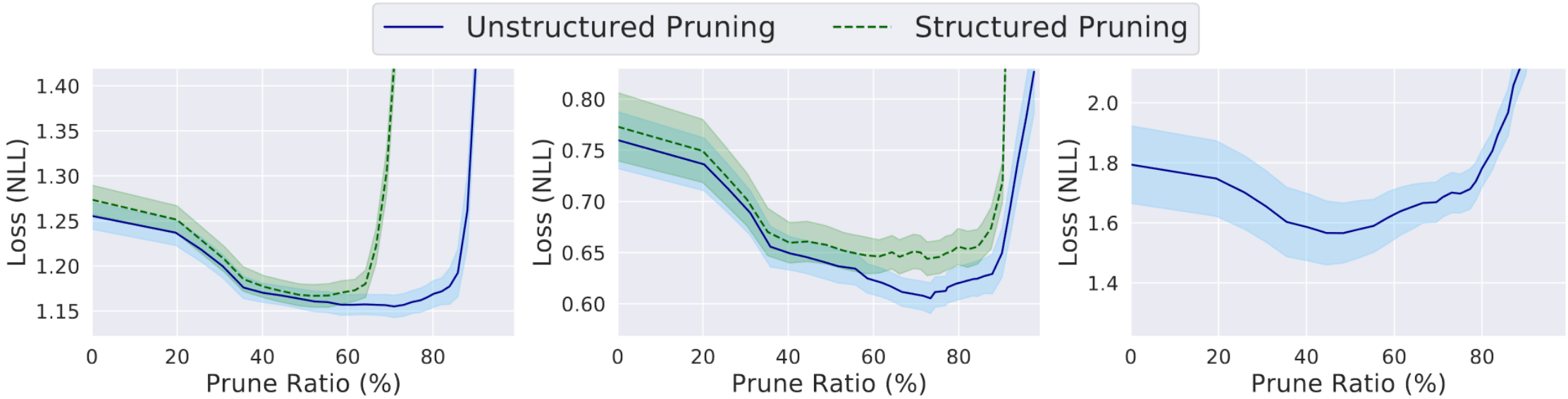
Change of variable: $\log p\big(\boldsymbol{z}(t_n)\big) = \log p\big(\boldsymbol{z}(t_0)\big) - \int_{t_0}^{t_1} \mathrm{Tr}\left(\frac{\partial f}{\partial \boldsymbol{z}(t)}\right) dt$



Sparse Flow: pruning continuous-depth models



Sparse Flow: toy datasets



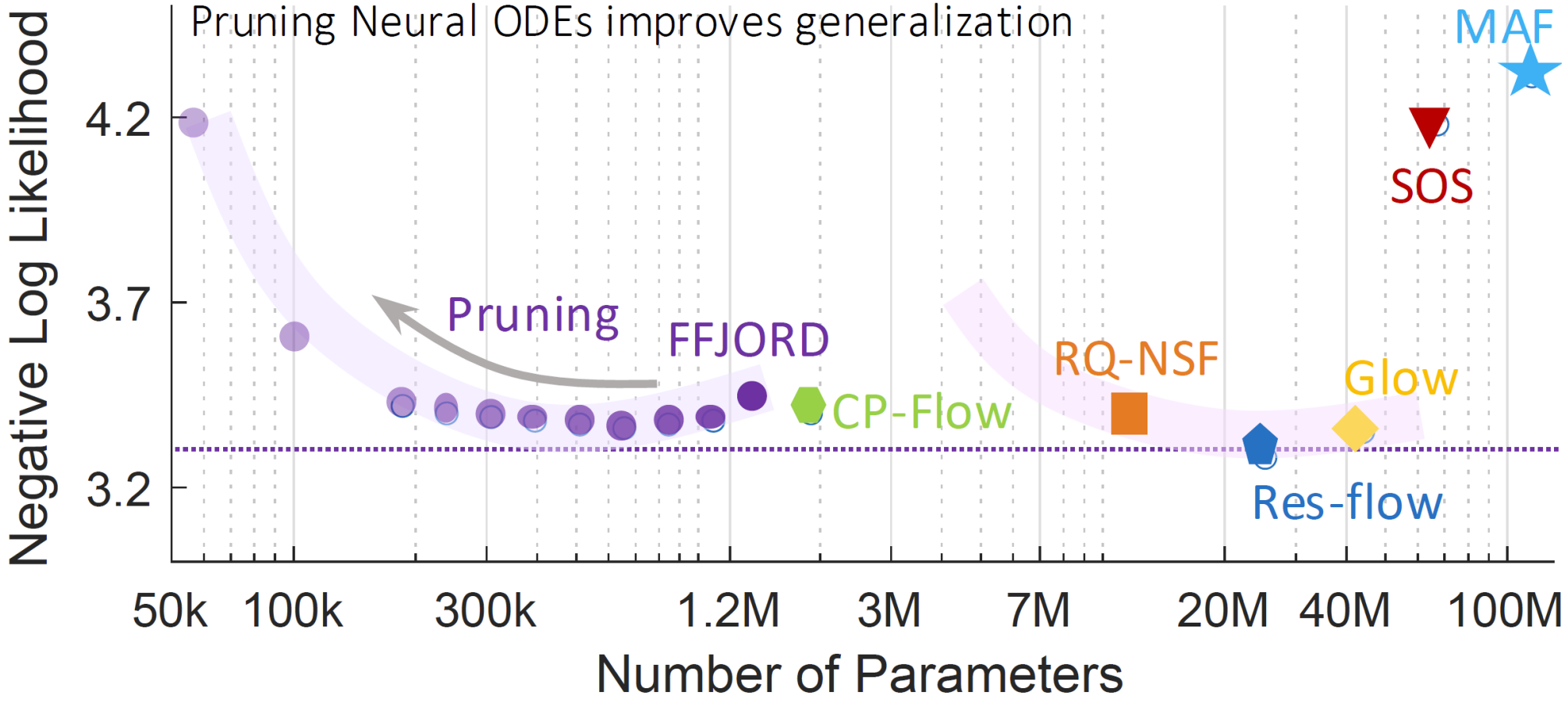
Gaussian

Gaussian Spiral

Spiral

lucasl@mit.edu

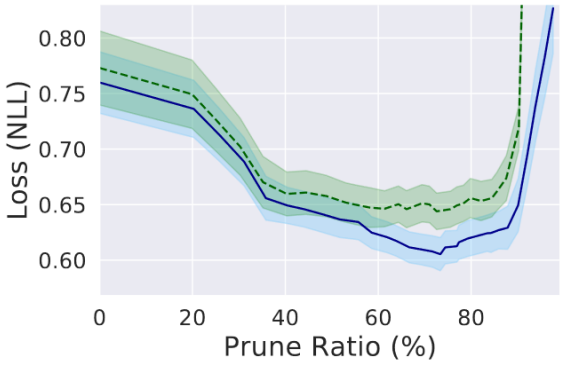
Sparse Flow: CIFAR10



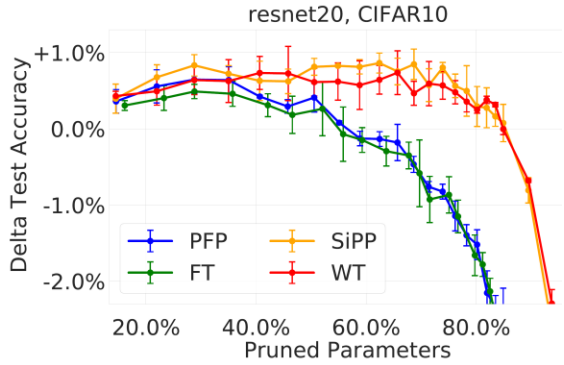
Lost or not

? Generalization ?

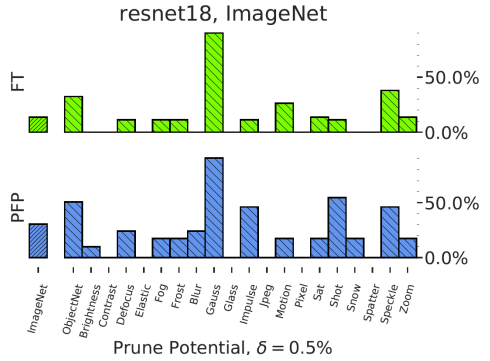
improves



persists



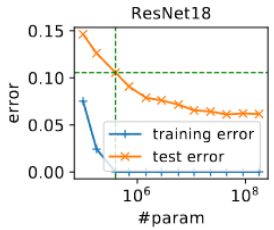
worsens



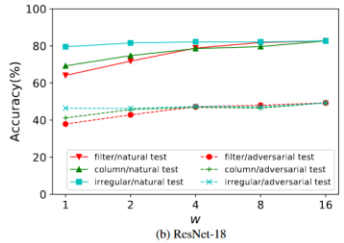
Memorization during training

Related work and discussion

Nominal vs. robust overparameterization

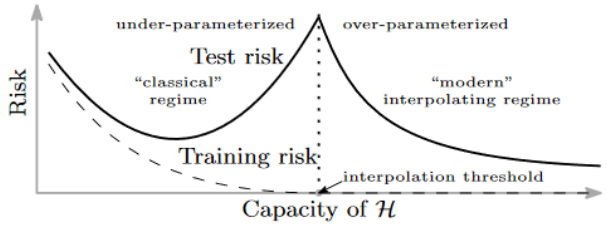


Neyshabur et al. 2019

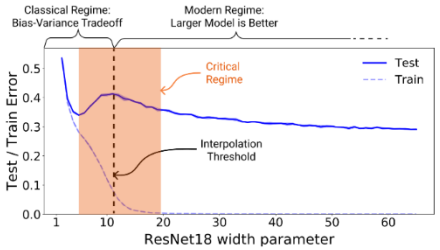


Ye et al. 2019

Implicit regularization via overparameterization

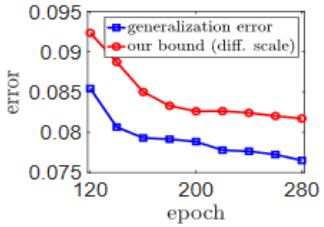


Belkin et al. 2019

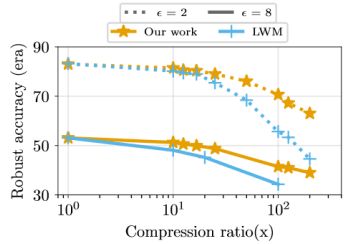


Nakkiran et al. 2020

Generalization-aware pruning

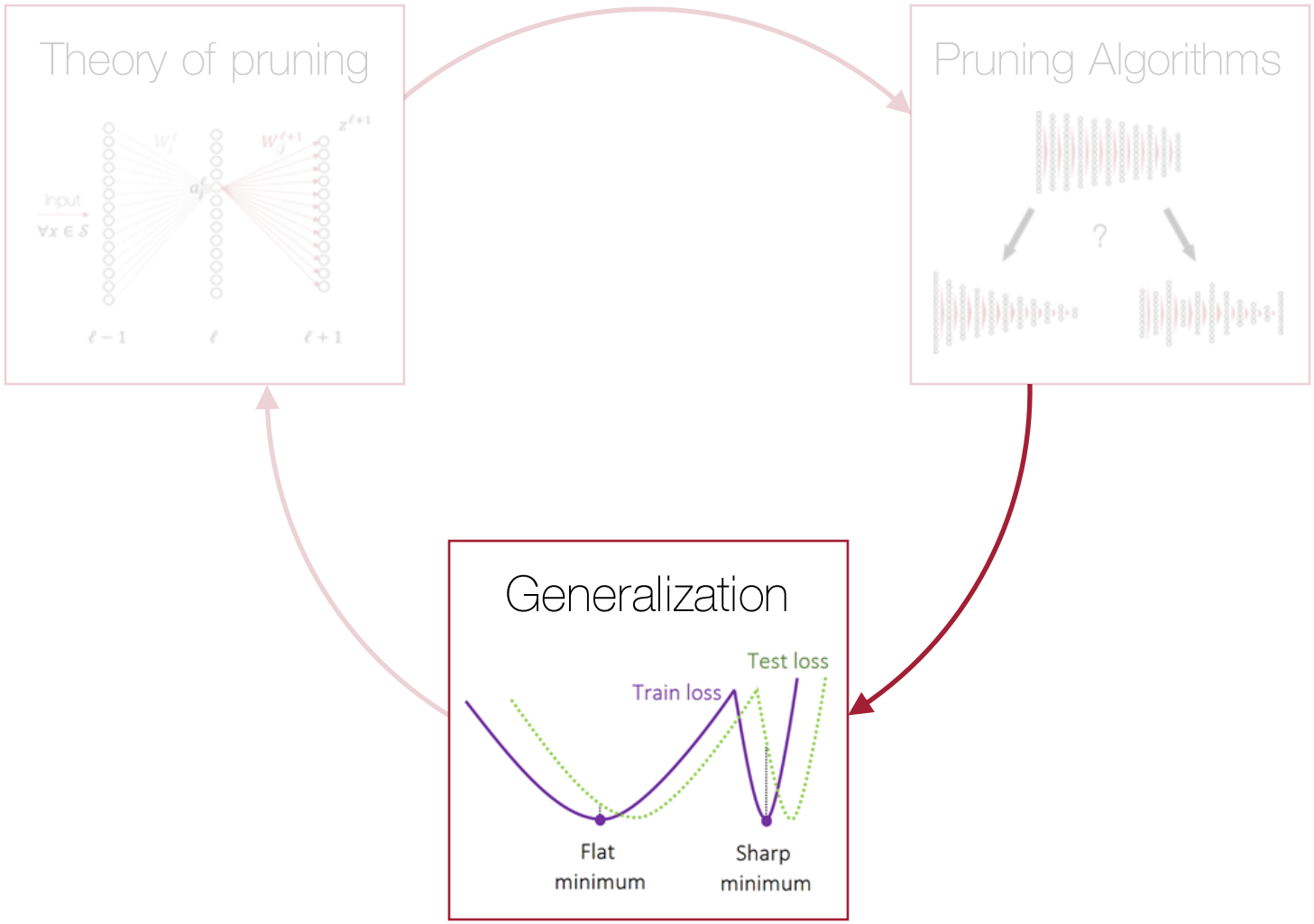


Arora et al. 2018

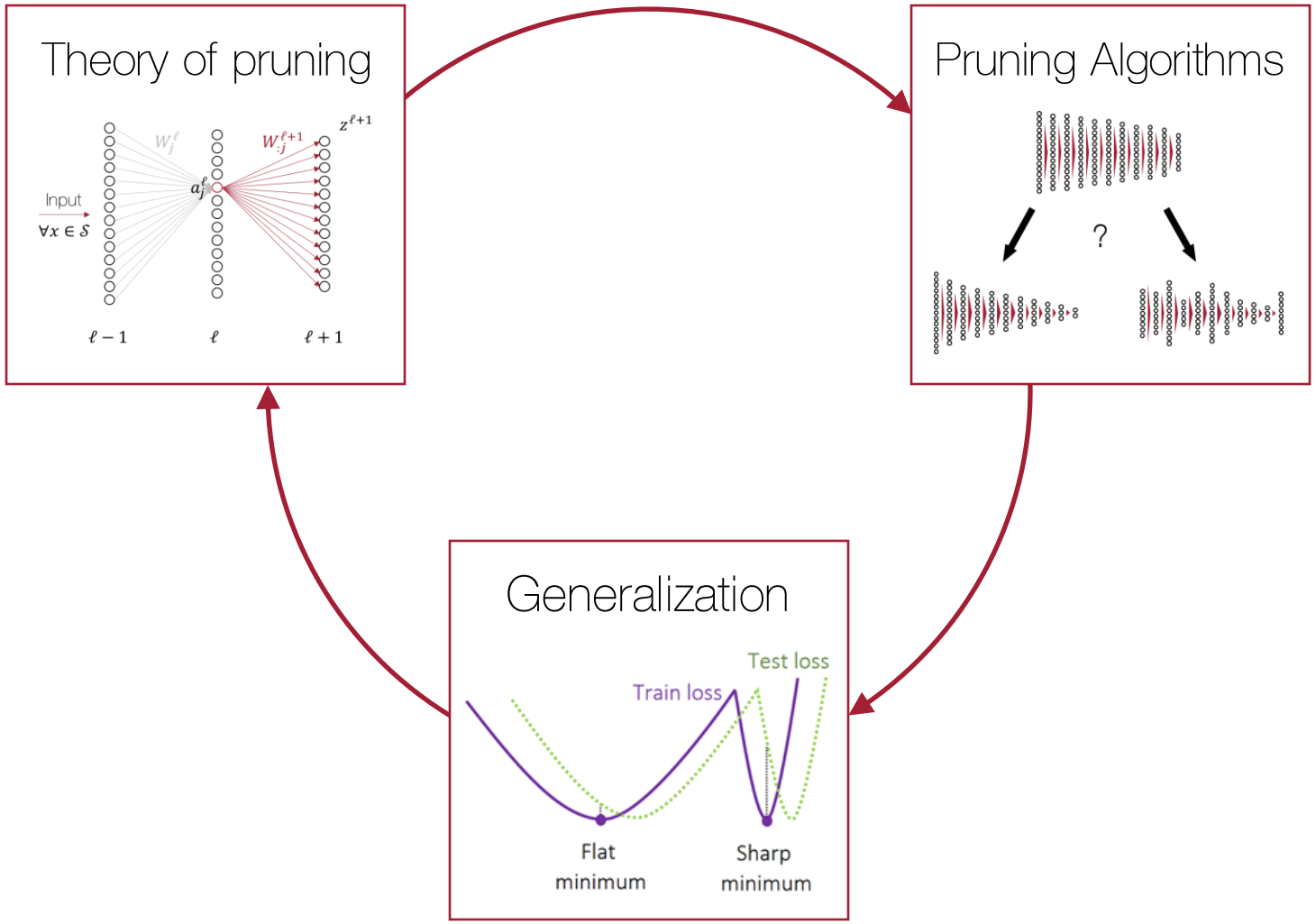


Sehwag et al. 2020

Outline


Theory of pruning
Pruning Algorithms
?
Generalization
Test loss
Train loss
Flat minimum
Sharp minimum

Outline


Theory of pruning
Input
$\forall x \in \mathcal{S}$
W_j^ℓ
a_j^ℓ
$W_{:j}^{\ell+1}$
$z^{\ell+1}$
$\ell - 1$
ℓ
$\ell + 1$
Pruning Algorithms
?
Generalization
Test loss
Train loss
Flat minimum
Sharp minimum

Conclusion

1. Built a theoretical understanding of *parameter efficiency* in deep learning.
2. Discussed the potential and limitations of our theoretical analysis.
3. Leveraged our theoretical understanding to design practical pruning algorithms.
4. Investigated the effect of pruning on the generalization capabilities of the network.

Thank you

Efficient Deep Learning: From Theory to Practice

Lucas Liebenwein, lucasl@mit.edu, http://www.mit.edu/~lucasl/

CSAIL, MIT

Thesis Supervisor: Daniela Rus, Thesis Committee: Michael Carbin, Song Han, Daniela Rus

References

[1] C. Baykal*, L. Liebenwein*, I. Gilitschenski, D. Feldman, and D. Rus. Data-dependent coresets for compressing neural networks with applications to generalization bounds. In *International Conference on Learning Representations*, 2019.

[2] L. Liebenwein*, C. Baykal*, H. Lang, D. Feldman, and D. Rus. Provable filter pruning for efficient neural networks. In *International Conference on Learning Representations*, 2020.

[3] L. Liebenwein*, A. Maalouf*, O. Gal, D. Feldman, and D. Rus. Compressing neural networks: Towards determining the optimal layer-wise decomposition. In *Advances in Neural Information Processing Systems (Under Review)*, 2021.

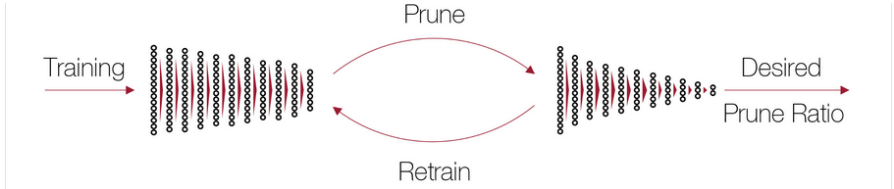
[4] L. Liebenwein, C. Baykal, B. Carter, D. Gifford, and D. Rus. Lost in pruning: The effects of pruning neural networks beyond test accuracy. In *Proceedings of Machine Learning and Systems*, 2021.

[5] L. Liebenwein*, R. Hasani*, A. Amini, and D. Rus. Sparse flows: Pruning continuous-depth models. In *Advances in Neural Information Processing Systems (Under Review)*, 2021.

Code

About

A research library for pytorch-based neural network pruning, compression, and more.



https://github.com/lucaslie/torchprune