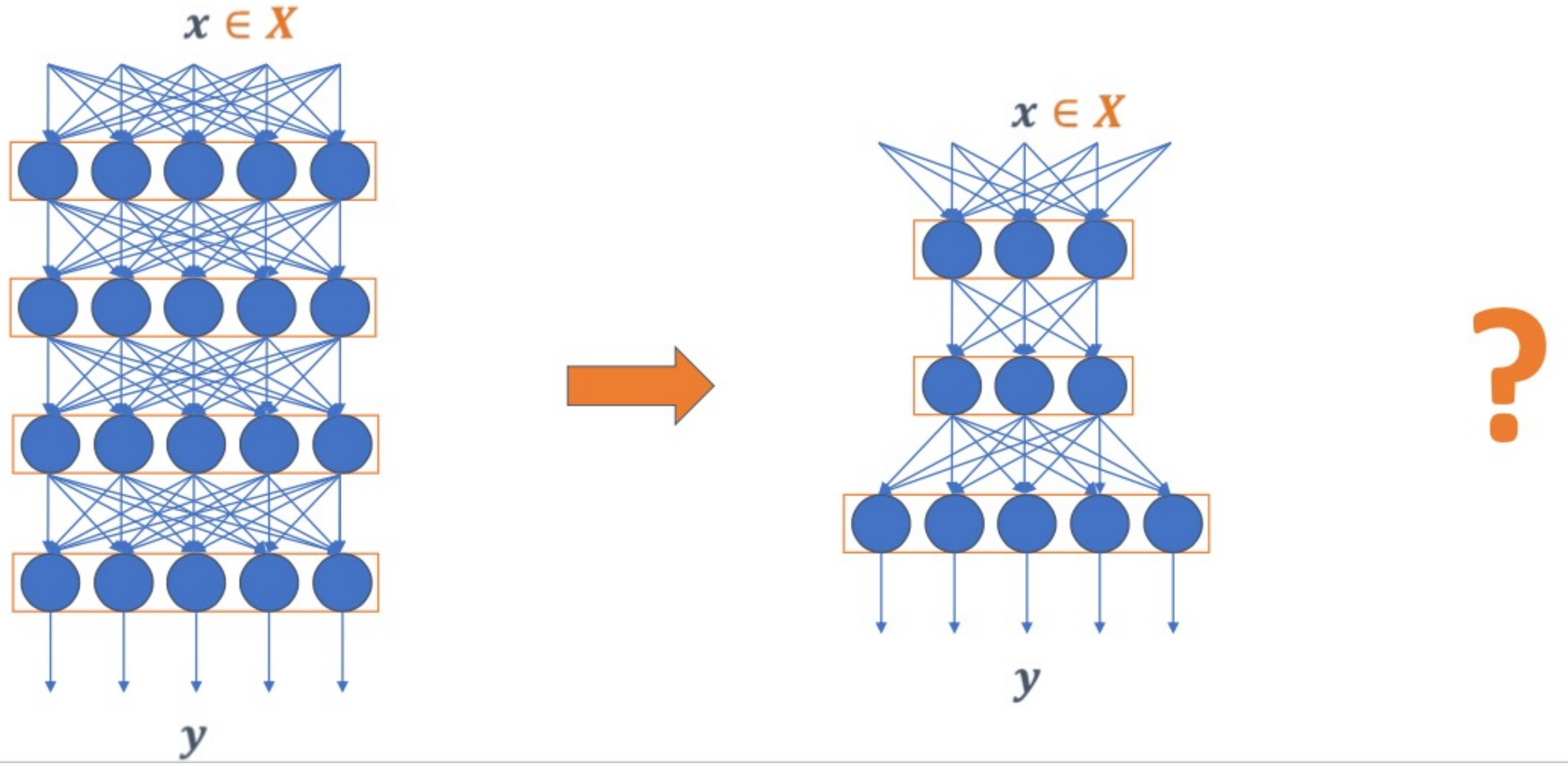
Literature Review: Network Compression

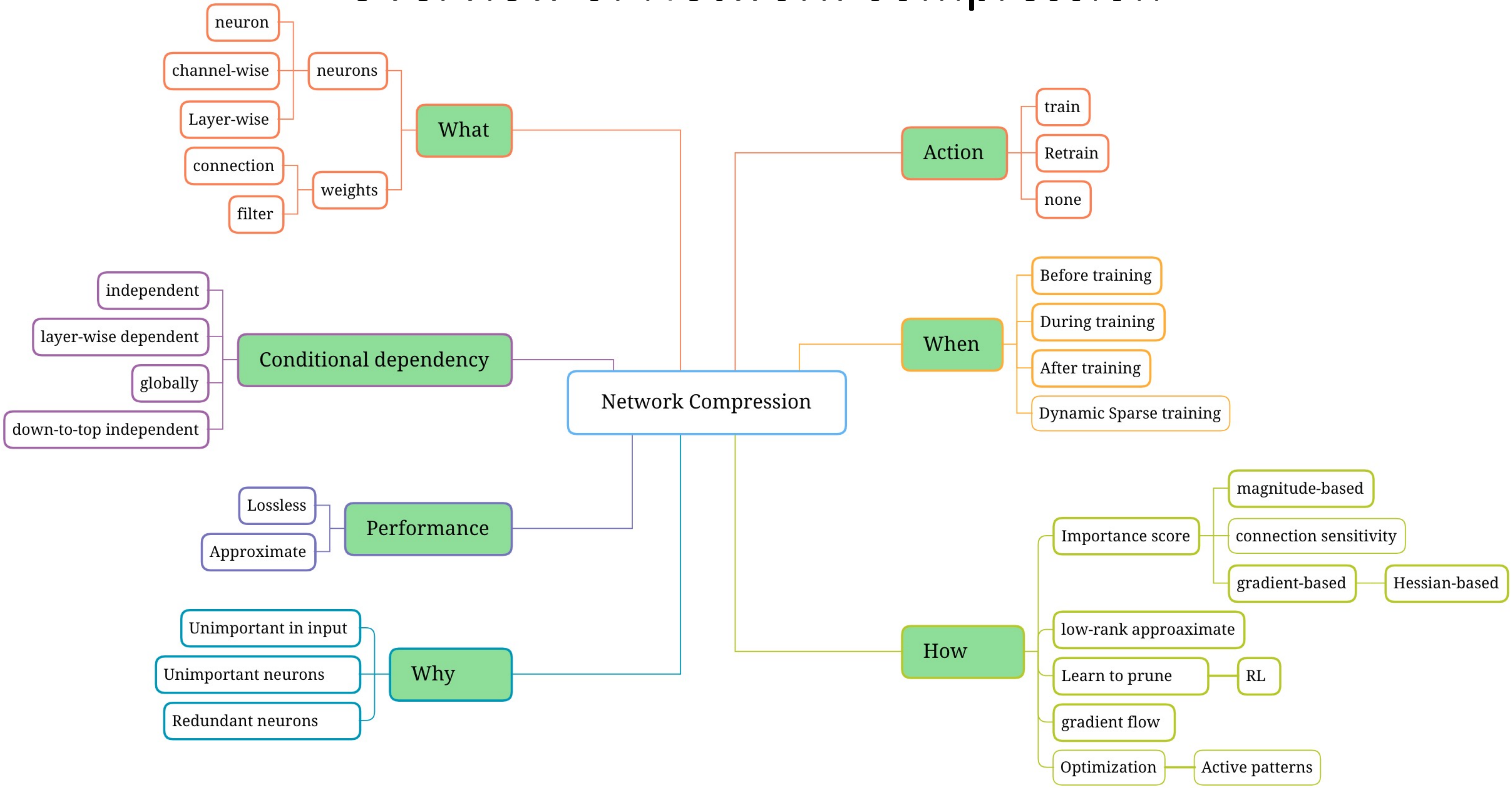
Xin Yu, 2021-8-31

Network compression: find a smaller network as good as a large one



[1] Serra, T., Kumar, A., Xin Y. and Ramalingam, S., 2021. Scaling Up Exact Neural Network Compression by ReLU Stability. *arXiv preprint arXiv:2102.07804*.

Overview of Network Compression

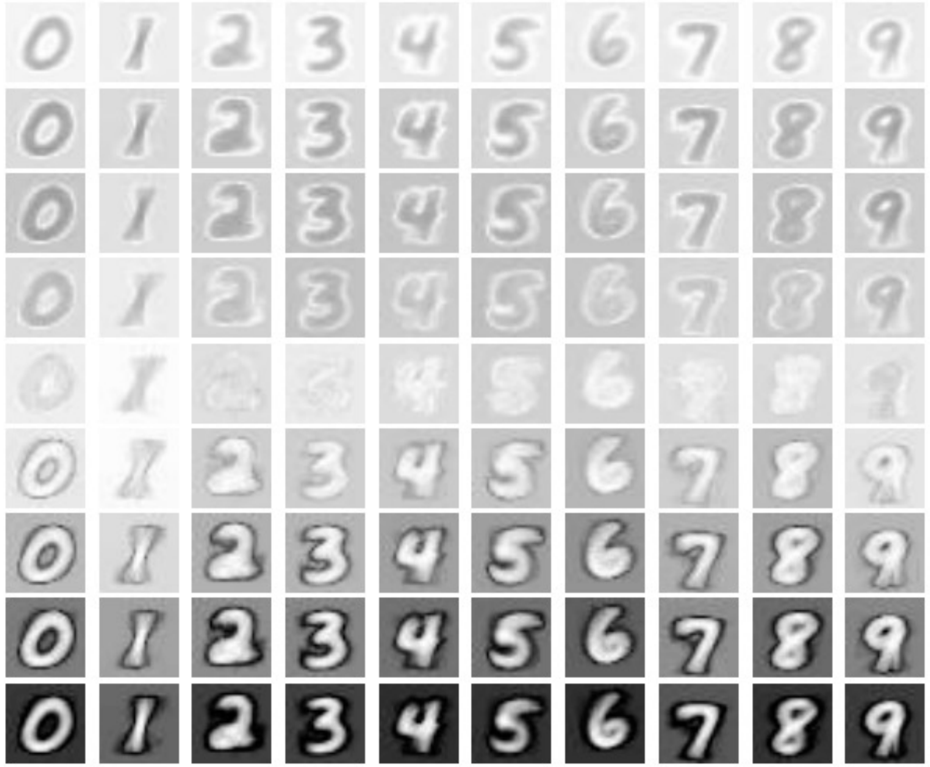


Background: why to prune networks

- Less resource consuming
  - Deploy to devices of limited resource, e.g., mobile phone, drone.
- Speedup inference
- Approximately same performance as the large network
- With quantization, it can be low-bit further
- Provide an insight into the presentation ability of a network
  - Which subnetwork contribute most to to task?

Why the network pruning is possible?

- Redundant information from the input
  - Given an image for '0' from MNIST, features of different area can used in the classification[1]



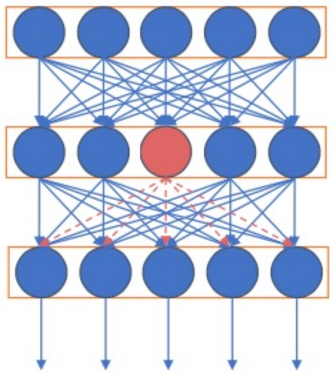
Bright pixels:
used features for the network

Dark features:
unimportant/unused features

[1] Lee, N., Ajanthan, T. and Torr, P.H., 2018. Snip: Single-shot network pruning based on connection sensitivity. **ICLR.**

Why the network pruning is possible?

- Unimportant/Redundant weights/neurons in networks
  - Stably inactive neurons: the output of the neurons are always 0



Hidden Layers	$\ell_1 = 0$	Accuracy	$\ell_1 = \bar{\ell}/2$	Accuracy	$\ell_1 = \bar{\ell}$	Accuracy
2 x 100	0%	97.93%	13%	98.14%	23%	97.89%
2 x 200	0%	98.17%	13%	98.33%	26%	98.17%
2 x 400	0%	98.25%	8%	98.35%	24%	98.24%
2 x 800	0%	98.28%	–	–	22%	98.29%

[1] Serra, T., Kumar, A., Xin Y. and Ramalingam, S., 2021. Scaling Up Exact Neural Network Compression by ReLU Stability. *arXiv preprint arXiv:2102.07804*.

How to prune: branches of Methods

- Pruning
  - After training
    - OBD: Optimal brain damage
    - OBS: Optimal brain surgeon
    - WoodFisher: Efficient Second-Order Approximation for Neural Network Compression
  - Before training
    - SNIP: Single-shot network pruning based on connection sensitivity
    - GraSP: Picking winning tickets before training by preserving gradient flow
- Quantization
- Knowledge distillation
- Low-rank decomposition
- Compact architecture design (similar as architecture search)

Methods: Optimal Brain Damage[1]

- Notation
  - Denote the dense weights by $\mathbf{w}$
  - the new weights after pruning as $\mathbf{w} + \delta\mathbf{w}$
  - The loss (the objective function) of the network
    - before pruning: $L(\mathbf{w})$
    - After pruning: $L(\mathbf{w} + \delta\mathbf{w})$
  - The target of pruning a network

$$\min_{\delta\mathbf{w} \in \mathbb{R}^d} \delta L = \min_{\delta\mathbf{w} \in \mathbb{R}^d} \Big(L(\mathbf{w} + \delta\mathbf{w}) - L(\mathbf{w}) \Big)$$

[1] LeCun, Yann, John S. Denker, and Sara A. Solla. "Optimal brain damage." NeurIPS. 1990.

Methods: Optimal Brain Damage

- Approximate the object function after pruning by a Taylor series

$$L(\mathbf{w}+\delta\mathbf{w}) \;=\; L(\mathbf{w}) + \nabla_{\mathbf{w}} L^{\top}\delta\mathbf{w} + \tfrac{1}{2}\delta\mathbf{w}^{\top}\mathbf{H}\;\delta\mathbf{w} + O(||\delta\mathbf{w}||^{3})$$

  - first derivative of the weights: $\nabla_{\mathbf{w}} L$
  - second derivative (Hessian matrix): $\mathbf{H}$

- We seek to minimize

$$\delta L = L(\mathbf{w}+\delta\mathbf{w}) - L(\mathbf{w}) \approx \nabla_{\mathbf{w}} L^{\top}\delta\mathbf{w} + \tfrac{1}{2}\delta\mathbf{w}^{\top}\mathbf{H}\;\delta\mathbf{w}$$

- Assumption1: the network is pruned at a local optimum, which eliminates the first term

$$\delta L \approx \frac{1}{2}\delta\mathbf{w}^{\top}\mathbf{H}\;\delta\mathbf{w}$$

Methods: Optimal Brain Damage

- Assumption2: Hessians tend to be diagonally- dominant
  - $f : \mathbb{R}^n \to \mathbb{R}$ has second order derivative, $\mathbf{x} = (x_1, \cdots, x_n)^T \in \mathbb{R}^n$
  - Definition of Hessian matrix: $H(\mathbf{x}) = [h_{ij}(\mathbf{x})]$

$$H(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1 \partial x_1} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n \partial x_n} \end{bmatrix} \Rightarrow H(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1 \partial x_1} & 0 & \cdots & 0 \\ 0 & \frac{\partial^2 f}{\partial x_2 \partial x_2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \frac{\partial^2 f}{\partial x_n \partial x_n} \end{bmatrix}$$

  - The equation reduce to

$$\delta L \approx \frac{1}{2} \sum_i h_{ii} \delta \mathbf{w}_i^2$$

Methods: Optimal Brain Damage

- Algorithm
  - Train the network until a reasonable solution is obtained
  - Compute the second derivatives h_{kk} for each paratmeter
  - Compute the saliencies for each parameters: $s_k = h_{kk}\delta \mathrm{w}_i^2/2$
  - Sort the parameters by saliency and delete the lowest saliency parameters.
  - Iterate to step 2 until the total pruning parameters reaches the target pruning ratio.

- Limitation:
  - Ignored the the non-diagonal elements in the hessian matrix

Methods: Optimal Brain Surgeon[1]

- **Calculate the full Hessian matrix**
- Rewrite the objective function as

$$\min_{\delta \mathbf{w} \in \mathbb{R}^d} \left(\frac{1}{2} \delta \mathbf{w}^\top \mathbf{H} \, \delta \mathbf{w} \right), \quad \text{s.t.} \quad \mathbf{e}_q^\top \delta \mathbf{w} + w_q = 0.$$

- As this is a constrained optimization problem, we can consider the Lagrange multiplier for the constraint as

$$\mathcal{L}(\delta \mathbf{w}, \lambda) = \frac{1}{2} \delta \mathbf{w}^\top \mathbf{H} \, \delta \mathbf{w} + \lambda \left(\mathbf{e}_q^\top \delta \mathbf{w} + w_q \right)$$

- We can obtain the solution by first differentiating the above equation and setting it to 0

$$\mathbf{H} \delta \mathbf{w} + \lambda \mathbf{e}_q = 0 \implies \delta \mathbf{w} = -\lambda \mathbf{H}^{-1} e_q.$$

$$g(\lambda) = \frac{\lambda^2}{2} \mathbf{e}_q^\top \mathbf{H}^{-1} \mathbf{e}_q - \lambda^2 \mathbf{e}_q^\top \mathbf{H}^{-1} \mathbf{e}_q + \lambda w_q = -\frac{\lambda^2}{2} \mathbf{e}_q^\top \mathbf{H}^{-1} \mathbf{e}_q + \lambda w_q.$$

- Maximizing the Lagrange dual function, and obtain $\delta \mathbf{w}^* = \frac{-w_q \mathbf{H}^{-1} \mathbf{e}_q}{[\mathbf{H}^{-1}]_{qq}} \quad \delta L^* = \frac{w_q^2}{2 \, [\mathbf{H}^{-1}]_{qq}}$

[1] Hassibi, B. and Stork, D.G., 1993. *Second order derivatives for network pruning: Optimal brain surgeon*. Morgan Kaufmann.

Methods: Optimal Brain Surgeon

- Calculating Hessian matrix for the network $\mathbf{o} = \mathbf{F}(\mathbf{w}, \mathbf{in})$
  - MSE $E = \frac{1}{2\mathrm{P}} \sum_{k=1}^{\mathrm{P}} (\mathbf{t}^{[k]} - \mathbf{o}^{[k]})^T (\mathbf{t}^{[k]} - \mathbf{o}^{[k]})$
  - The first derivative

$$\frac{\partial E}{\partial \mathbf{w}} = -\frac{1}{\mathrm{P}} \sum_{k=1}^{\mathrm{P}} \frac{\partial \mathbf{F}(\mathbf{w}, \mathbf{in}^{[\mathbf{k}]})}{\partial \mathbf{w}} (\mathbf{t}^{[k]} - \mathbf{o}^{[k]})$$

  - The second derivative

$$\mathbf{H} \equiv \frac{\partial^2 E}{\partial \mathbf{w}^2} = \frac{1}{\mathrm{P}} \sum_{k=1}^{\mathrm{P}} \left[\frac{\partial \mathbf{F}(\mathbf{w}, \mathbf{in}^{[\mathbf{k}]})}{\partial \mathbf{w}} \cdot \frac{\partial \mathbf{F}(\mathbf{w}, \mathbf{in}^{[\mathbf{k}]})}{\partial \mathbf{w}}^T - \frac{\partial^2 \mathbf{F}(\mathbf{w}, \mathbf{in}^{[\mathbf{k}]})}{\partial \mathbf{w}^2} \cdot (\mathbf{t}^{[k]} - \mathbf{o}^{[k]}) \right]$$

  - Use Assumption1 again: the network is pruned at a local optimum, which eliminates the second term

$$\mathbf{H} = \frac{1}{\mathrm{P}} \sum_{k=1}^{\mathrm{P}} \frac{\partial \mathbf{F}(\mathbf{w}, \mathbf{in}^{[\mathbf{k}]})}{\partial \mathbf{w}} \cdot \frac{\partial \mathbf{F}(\mathbf{w}, \mathbf{in}^{[\mathbf{k}]})}{\partial \mathbf{w}}^T$$

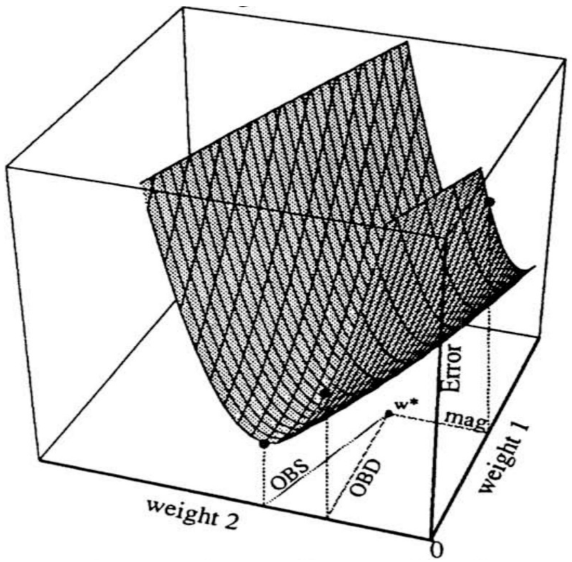
  - H is the sample covariance matrix associated with the gradient vector $\mathbf{X}^{[\mathbf{k}]} \equiv \frac{\partial \mathbf{F}(\mathbf{w}, \mathbf{in}^{[\mathbf{k}]})}{\partial \mathbf{w}}$

Methods: Optimal Brain Surgeon

- Algorithm
  1. Train the network until a reasonable solution is obtained
  2. Compute
  3. Compute $\mathbf{H}^{-1}$e saliencies for each parameters: $s_k = \frac{w_k^2}{2\,[\mathbf{H}^{-1}]_{kk}}$ and find the smallest saliency
  4. **Use the q from the previous step to update all weights.**
  5. Iterate to step 2 until the total pruning parameters reaches the target pruning ratio.
- OBS vs. OBD vs magnitude base pruning:
  - When eliminate the same weights:

    $$E(\text{mag}) \geq E(\text{OBD}) \geq E(\text{OBS})$$

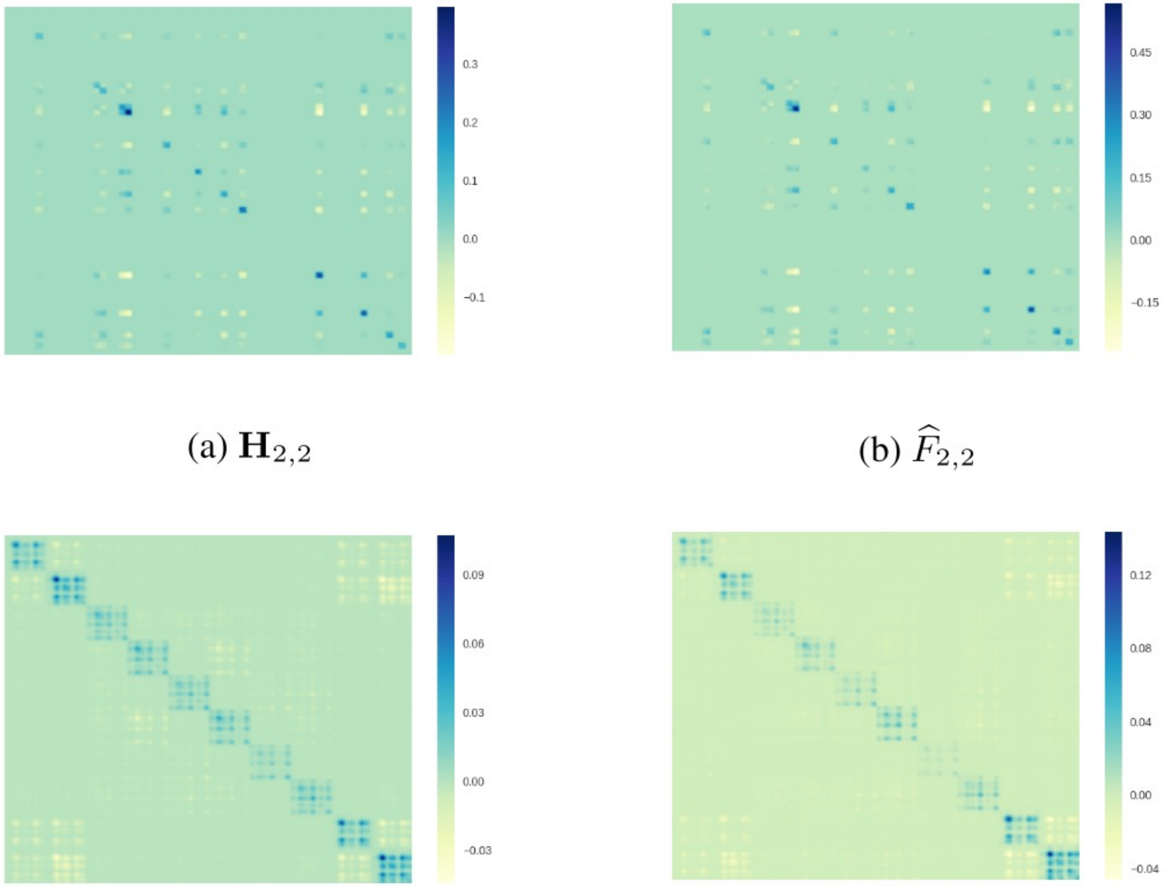
  - In many cases, OBS will eliminate different weights than those by OBD



- Limitation: inefficient to calculate
  - H: (N_w x N_w x N_data)

Methods: WoodFisher[1]

- Speedup the calculating of H by approaching it with Fisher information matrix.



(a) $\mathbf{H}_{2,2}$ (b) $\widehat{F}_{2,2}$

(c) $\mathbf{H}_{3,3}$ (d) $\widehat{F}_{3,3}$

[1] Singh, S.P. and Alistarh, D., 2020. Woodfisher: Efficient second-order approximation for neural network compression. NeurIPS.

Methods: WoodFisher

- Fisher Matrix

$$F = \mathrm{E}_{P_{\mathbf{x},\mathbf{y}}}\left[\nabla_{\mathbf{w}} \log p_{\mathbf{w}}(\mathbf{x},\mathbf{y})\, \nabla_{\mathbf{w}} \log p_{\mathbf{w}}(\mathbf{x},\mathbf{y})^{\top}\right].$$

- It can be proved that the Fisher and Hessian matrix are equivalent.

$$F = \mathrm{E}_{P_{\mathbf{x},\mathbf{y}}}\left[-\nabla^2_{\mathbf{w}} \log p_{\mathbf{w}}(\mathbf{x},\mathbf{y})\right]$$

- The empirical Fisher.

$$\hat{F} = \mathrm{E}_{\widehat{Q}_{\mathbf{x}}}\left[\mathrm{E}_{\widehat{Q}_{\mathbf{y}|\mathbf{x}}}\left[\nabla \log p_{\mathbf{w}}(\mathbf{y}|\mathbf{x}) \nabla \log p_{\mathbf{w}}(\mathbf{y}|\mathbf{x})^{\top}\right]\right] \stackrel{(a)}{=} \frac{1}{N}\sum_{n=1}^{N} \underbrace{\nabla \ell\left(\mathbf{y}_n, f\left(\mathbf{x}_n; \mathbf{w}\right)\right)}_{\nabla \ell_n} \nabla \ell\left(\mathbf{y}_n, f\left(\mathbf{x}_n; w\right)\right)^{\top}$$

- Question:
  - WoodFisher vs OBS: $\mathbf{H} = \frac{1}{\mathbf{P}}\sum_{k=1}^{\mathbf{P}} \frac{\partial \mathbf{F}(\mathbf{w}, \mathbf{in}^{[\mathbf{k}]})}{\partial \mathbf{w}} \cdot \frac{\partial \mathbf{F}(\mathbf{w}, \mathbf{in}^{[\mathbf{k}]})}{\partial \mathbf{w}}^{T}$

Prune networks vs. train a small network

- Pruning
  - Unstructured pruning for individual weights
  - Structure pruning in filter/channel/layer wise
- Does sparse network structure matter?
  - Argument1[1]: pruning a network and finetuning it can archive better performance than the small network
  - For structured pruning, training a subnetwork from scratching without keep the initialization is possible to perform as good as finetune a pruned network.
- Does the initialization of the original network matter?
  - Argument2[1] : for unstructured pruning, small learning rate can lead to winning ticket perform better than training from random intialization.
- **Can we find a sparse structure without training?**

[1] Liu, Z., Sun, M., Zhou, T., Huang, G. and Darrell, T., 2019. Rethinking the value of network pruning. *ICLR.*

Methods: before training

- Given a network initialized as W_0 and a dataset $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n$, how to obtain a subnetwork with a few data without training?
- SNIP: Single-shot network pruning based on connection sensitivity[1]
  - The objective: ($\|\cdot\|_0$ is L_0 norm.)

$$\min_{\mathbf{w}} L(\mathbf{w}; \mathcal{D}) = \min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} \ell(\mathbf{w}; (\mathbf{x}_i, \mathbf{y}_i)) \ ,$$
$$\text{s.t.} \quad \mathbf{w} \in \mathbb{R}^m, \quad \|\mathbf{w}\|_0 \leq \kappa \ .$$

  - Introduce a mask variable on the weights $\mathbf{c} \in \{0,1\}^m$

$$\min_{\mathbf{c},\mathbf{w}} L(\mathbf{c} \odot \mathbf{w}; \mathcal{D}) = \min_{\mathbf{c},\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} \ell(\mathbf{c} \odot \mathbf{w}; (\mathbf{x}_i, \mathbf{y}_i)) \ ,$$
$$\text{s.t.} \quad \mathbf{w} \in \mathbb{R}^m \ ,$$
$$\mathbf{c} \in \{0,1\}^m, \quad \|\mathbf{c}\|_0 \leq \kappa \ ,$$

  - Remove a single weight in isolation: $S(\theta_q) = \lim_{\epsilon \to 0} \left| \frac{\mathcal{L}(\boldsymbol{\theta}_0) - \mathcal{L}(\boldsymbol{\theta}_0 + \epsilon \boldsymbol{\delta}_q)}{\epsilon} \right| = \left| \theta_q \frac{\partial \mathcal{L}}{\partial \theta_q} \right|$

[1] Lee, N., Ajanthan, T. and Torr, P.H., 2019. Snip: Single-shot network pruning based on connection sensitivity. *ICLR*.

Methods: before training

GraSP: Picking winning tickets before training by preserving gradient flow[1]

- Notation:
  - Output of a neural network: $\mathcal{Z} = f(\mathcal{X};\boldsymbol{\theta}) \in \mathbb{R}^{nk\times 1}$
  - For a step of gradient decent, $\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t = -\eta \nabla_{\boldsymbol{\theta}} f(\mathcal{X};\boldsymbol{\theta}_t)$

  the change to the network's prediction can be approximated with a first-order Taylor approximation:

$$f(\mathcal{X};\boldsymbol{\theta}_{t+1}) = f(\mathcal{X};\boldsymbol{\theta}_t) + \nabla_{\boldsymbol{\theta}} f(\mathcal{X};\boldsymbol{\theta}_t)\nabla_{\mathcal{Z}}\mathcal{L}\left(\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t\right) = f(\mathcal{X};\boldsymbol{\theta}_t) - \eta\boldsymbol{\Theta}_t(\mathcal{X},\mathcal{X})\nabla_{\mathcal{Z}}\mathcal{L}$$

$$\boldsymbol{\Theta}_t(\mathcal{X},\mathcal{X}) = \nabla_{\boldsymbol{\theta}} f(\mathcal{X};\boldsymbol{\theta}_t)\nabla_{\boldsymbol{\theta}} f(\mathcal{X};\boldsymbol{\theta}_t)^\top \in \mathbb{R}^{nk\times nk}$$

  - Gradient flow:

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

  - Preserving gradient flow after training:

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

[1] Wang, C., Zhang, G. and Grosse, R., 2020. Picking winning tickets before training by preserving gradient flow. *ICLR*.

Methods: before training

GraSP: Picking winning tickets before training by preserving gradient flow[IC LR20]

- Algorithm

Algorithm 1 Gradient Signal Preservation (GraSP).

Require: Pruning ratio p, training data $\mathcal{D}$, network f with initial parameters $\boldsymbol{\theta}_0$

$\mathcal{D}_b = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{b} \sim \mathcal{D}$ ▷ Sample a collection of training examples

Compute the Hessian-gradient product $\mathbf{Hg}$ (see Eqn. (8)) ▷ See Algorithm 2

$\mathbf{S}(-\boldsymbol{\theta}_0) = -\boldsymbol{\theta}_0 \odot \mathbf{Hg}$ ▷ Compute the score of each weight

Compute p_{th} percentile of $\mathbf{S}(-\boldsymbol{\theta}_0)$ as τ

$\mathbf{m} = \mathbf{S}(-\boldsymbol{\theta}_0) < \tau$ ▷ Remove the weights with the *largest* scores

Train the network $f_{\mathbf{m}\odot\boldsymbol{\theta}}$ on $\mathcal{D}$ until convergence.

- How to calculate the Hessian-gradient Product

Algorithm 2 Hessian-gradient Product.

Require: A batch of training data $\mathcal{D}_b$, network f with initial parameters $\boldsymbol{\theta}_0$, loss function $\mathcal{L}$

$\mathcal{L}(\boldsymbol{\theta}_0) = \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}_b}[\ell(f(\mathbf{x};\boldsymbol{\theta}_0), y)]$ ▷ Compute the loss and build the computation graph

$\mathbf{g} = \mathrm{grad}(\mathcal{L}(\boldsymbol{\theta}_0), \boldsymbol{\theta}_0)$ ▷ Compute the gradient of loss function with respect to $\boldsymbol{\theta}_0$

$\mathbf{Hg} = \mathrm{grad}(\mathbf{g}^\top \mathrm{stop\_grad}(\mathbf{g}), \boldsymbol{\theta}_0)$ ▷ Compute the Hessian vector product of $\mathbf{Hg}$

Return $\mathbf{Hg}$

Comparison[1]

Table 1: Comparisons with Random Pruning with VGG19 and ResNet32 on CIFAR-10/100.

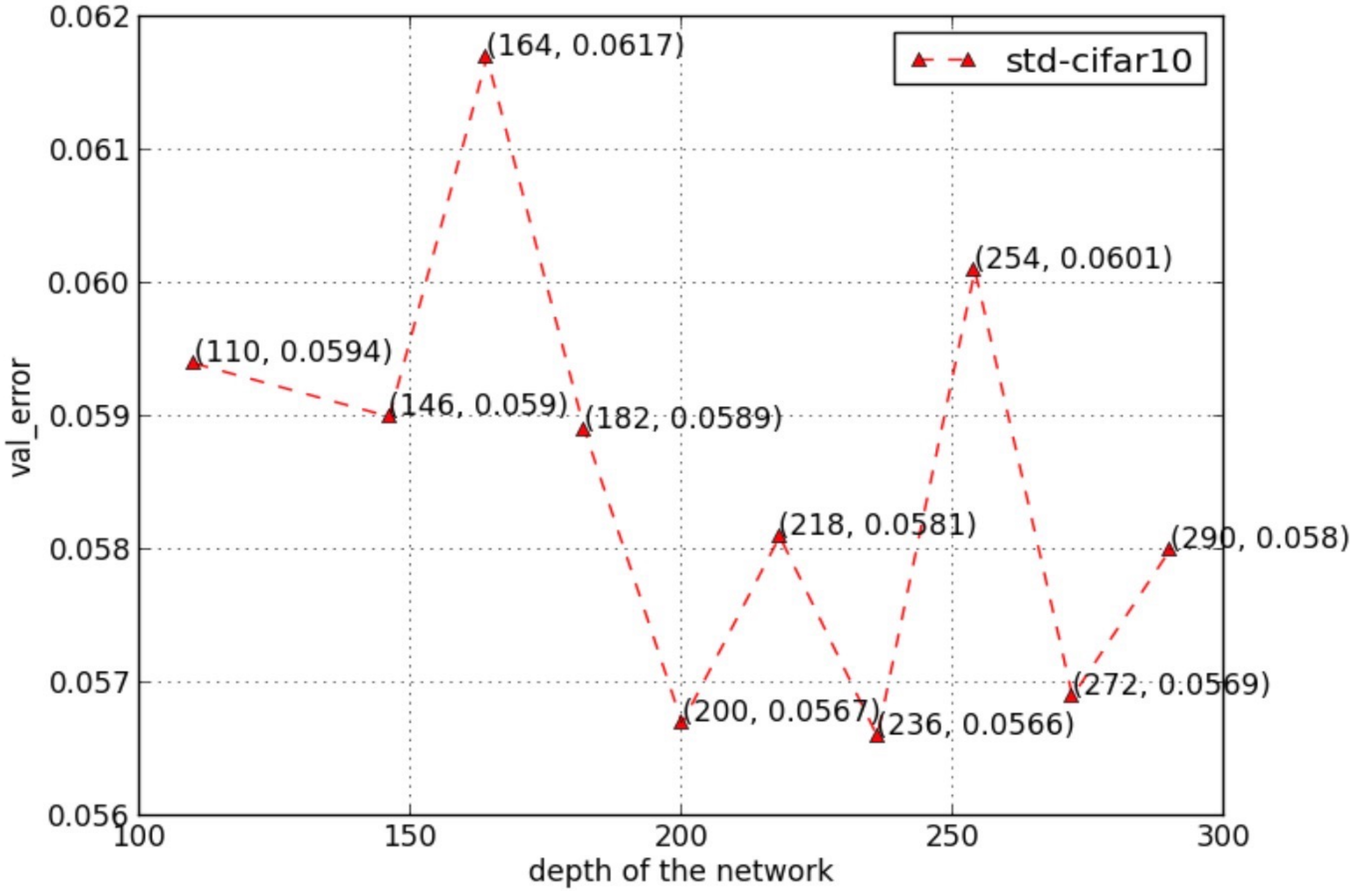
Dataset	CIFAR-10				CIFAR-100			
Pruning ratio	95%	98%	99%	99.5%	95%	98%	99%	99.5%
Random Pruning (VGG19)	89.47(0.5)	86.71(0.7)	82.21(0.5)	72.89(1.6)	66.36(0.3)	61.33(0.1)	55.18(0.6)	36.88(6.8)
GraSP (VGG19)	**93.04(0.2)**	**92.19(0.1)**	**91.33(0.1)**	**88.61(0.7)**	**71.23(0.1)**	**68.90(0.5)**	**66.15(0.2)**	**60.21(0.1)**
Random Pruning (ResNet32)	89.75(0.1)	85.90(0.4)	71.78(9.9)	50.08(7.0)	64.72(0.2)	50.92(0.9)	34.62(2.8)	18.51(0.43)
GraSP (ResNet32)	**91.39(0.3)**	**88.81(0.1)**	**85.43(0.5)**	**80.50(0.3)**	**66.50(0.1)**	**58.43(0.4)**	**48.73(0.3)**	**35.55(2.4)**

Dataset	CIFAR-10			CIFAR-100		
Pruning ratio	90%	95%	98%	90%	95%	98%
VGG19 (Baseline)	94.23	-	-	74.16	-	-
OBD (LeCun et al., 1990)	93.74	93.58	93.49	73.83	71.98	67.79
MLPrune (Zeng & Urtasun, 2019)	93.83	93.69	93.49	73.79	73.07	71.69
LT (original initialization)	93.51	92.92	92.34	72.78	71.44	68.95
LT (reset to epoch 5)	93.82	93.61	93.09	74.06	72.87	70.55
DSR (Mostafa & Wang, 2019)	93.75	93.86	93.13	72.31	71.98	70.70
SET Mocanu et al. (2018)	92.46	91.73	89.18	72.36	69.81	65.94
Deep-R (Bellec et al., 2018)	90.81	89.59	86.77	66.83	63.46	59.58
SNIP (Lee et al., 2018)	**93.63±0.06**	**93.43±0.20**	92.05±0.28	**72.84±0.22**	**71.83±0.23**	58.46±1.10
GraSP	93.30±0.14	93.04±0.18	**92.19±0.12**	71.95±0.18	71.23±0.12	**68.90±0.47**

[1] Wang, C., Zhang, G. and Grosse, R., 2020. Picking winning tickets before training by preserving gradient flow. *ICLR*.

Methods: Structure pruning

ϵ -ResNet: Learning Strict Identity Mappings

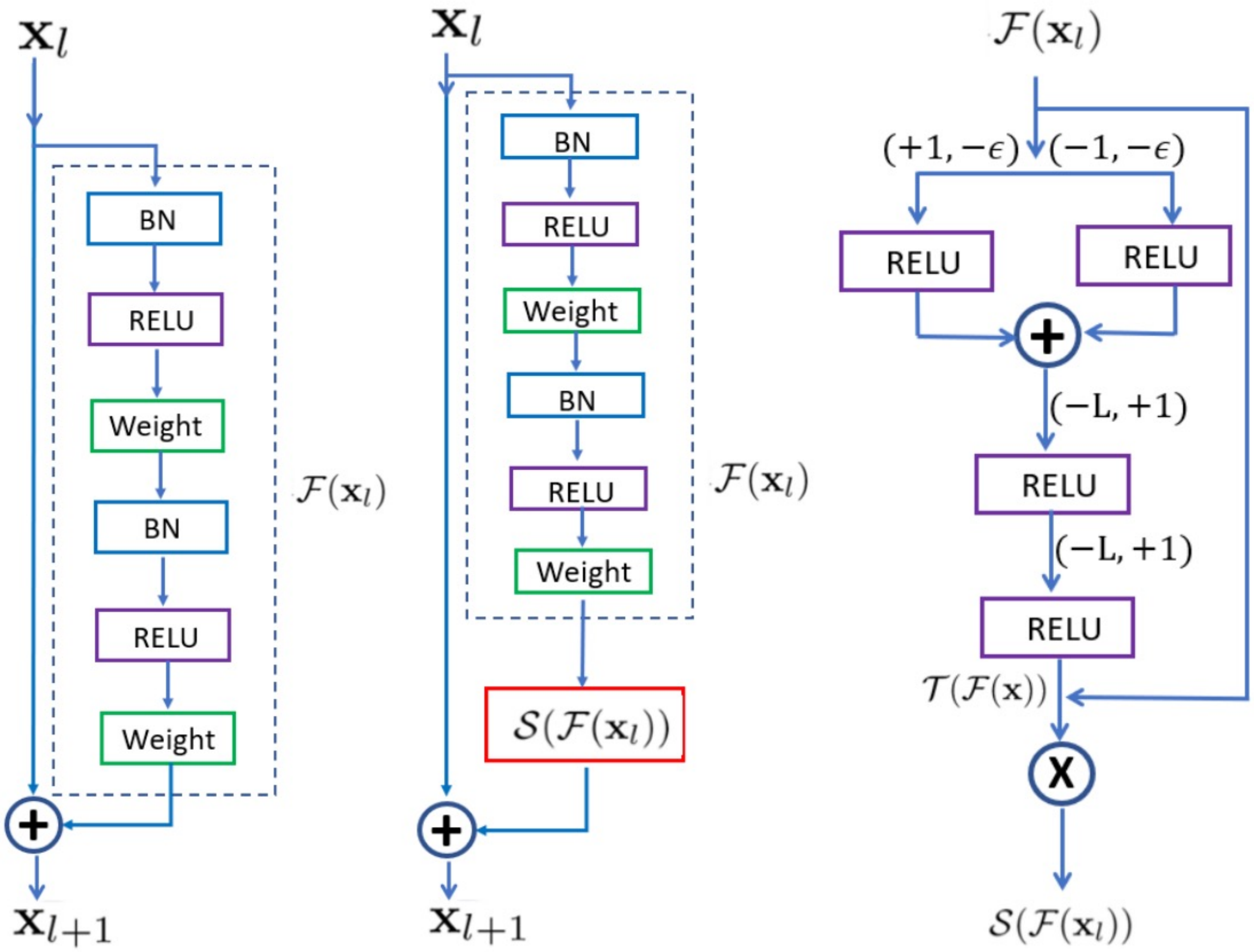


Intuition from ResNet: the performance doesn't increase with the number of layers

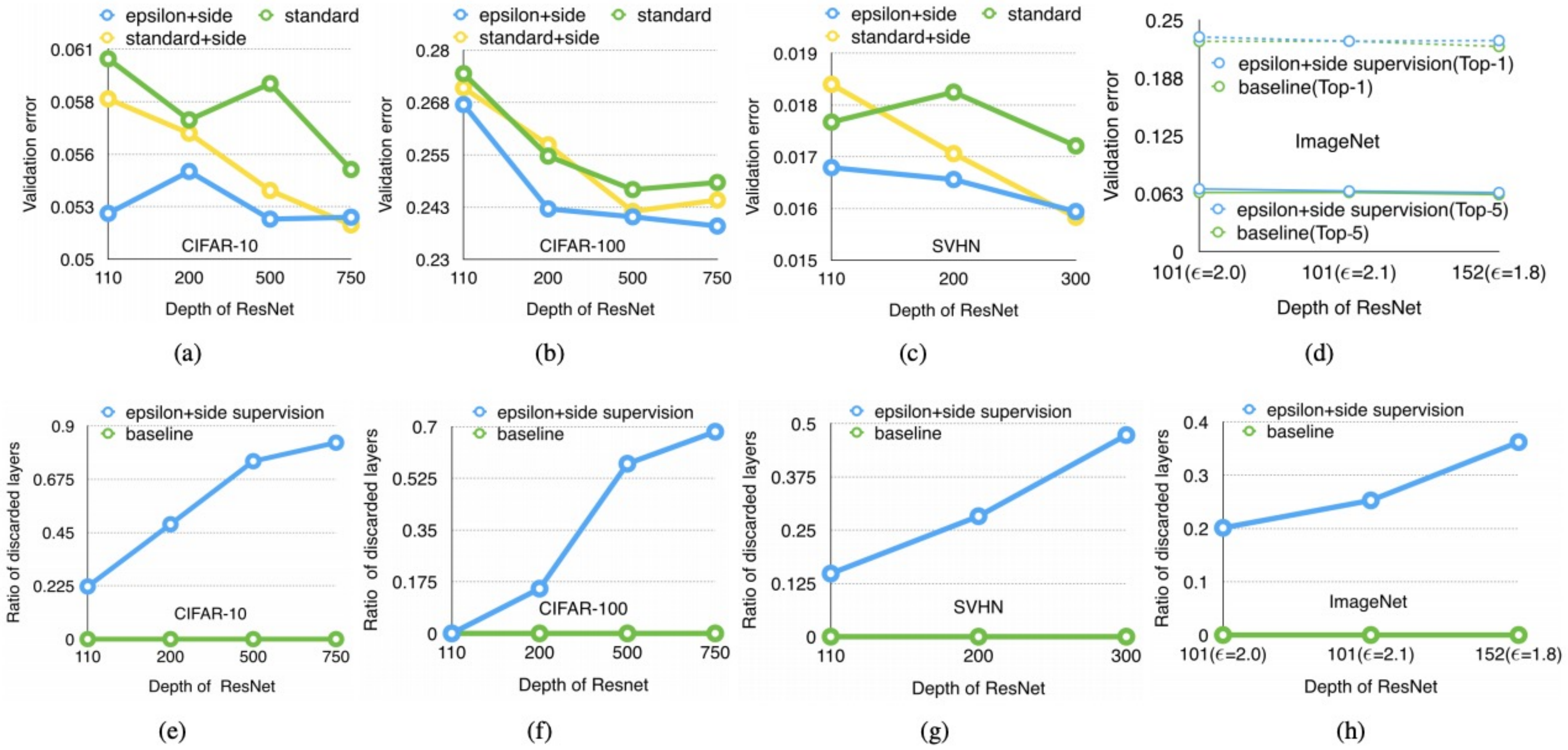
[1] Yu, X., Yu, Z. and Ramalingam, S., 2018. Learning strict identity mappings in deep residual networks. CVPR.

Methods: Structure pruning

ϵ-ResNet: Use strict identity mapping to compress the network



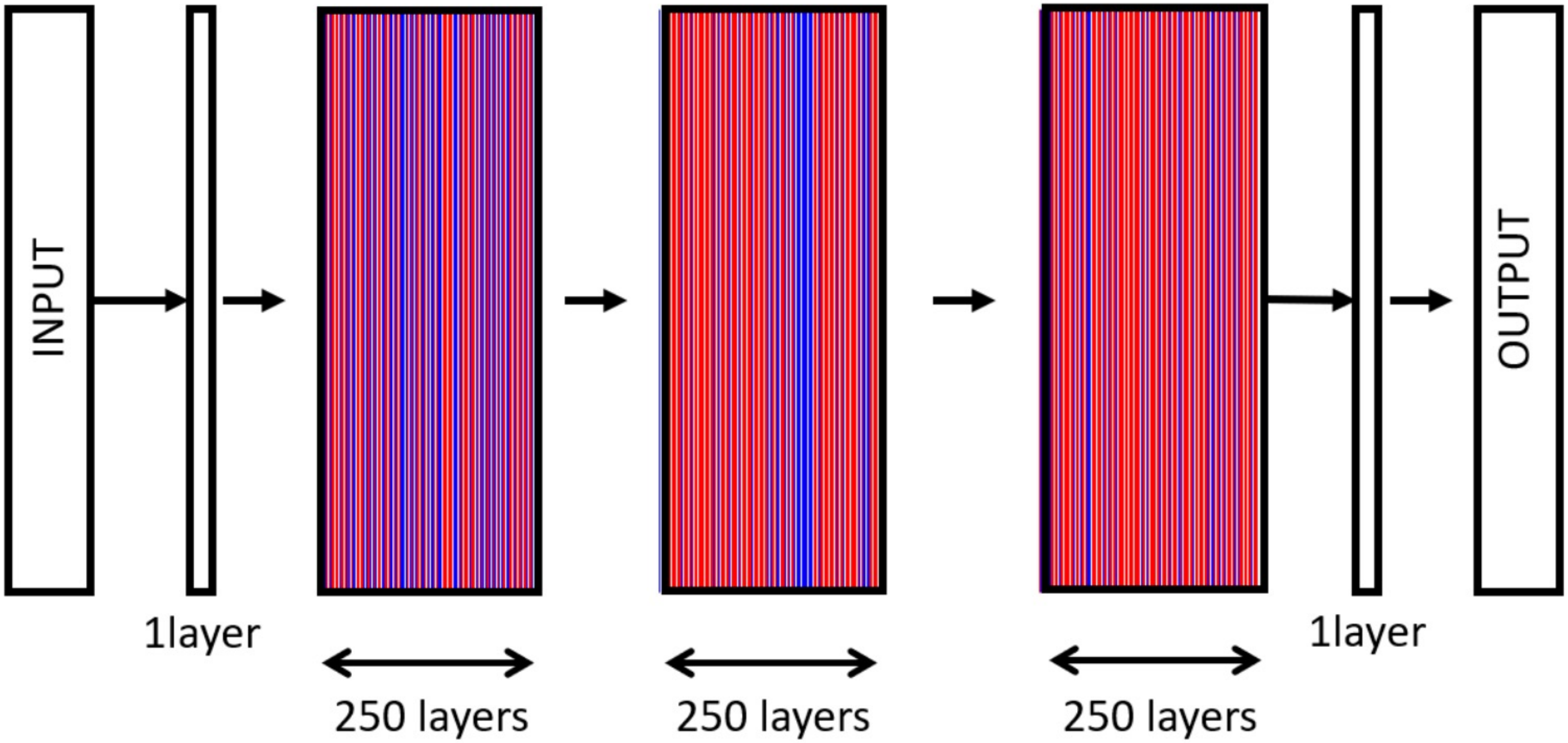
Methods: Structure pruning



Results of ϵ-ResNet

Methods: Structure pruning

ϵ-ResNet: visualization of ResNet750 on Cifar100 after pruning



Blue: retained layers. Red: pruned layers. Compression ration: 3.2 (*#all_layers/#remained_layers*)

	When	What	How	Action	Performance	Dependency	Why
Epsilon-ResNet[1]	During training	Layers	Magnitude of the feature map	prune-retrain	Approximate	-	Unimportant layers
ScalingUp[2]	After training	Neurons	MILP optimization	None	Lossless	Globally	Unimportant + redundant neurons
NISP[6]	After training	Neurons	Binary integer program	Finetune	Approximate	Down-to-top	
Surrogate[7]	After training	Weights	Hessian-based		Approximate	Independent	Weights which don’t change Loss
LotterayTicket [3]	After training	Weights	Magnitude-based	Retrain	Approximate	-	Weight initialization + Data
SNIP[4]	Before training	Weights	Connection sensitivity	Train	Approximate	-	Data
PickingWinningTickets[5]	Before training	Weights	Preseve Gradient flow	Train	Approximate	Globally	Data

Thanks! Q&A