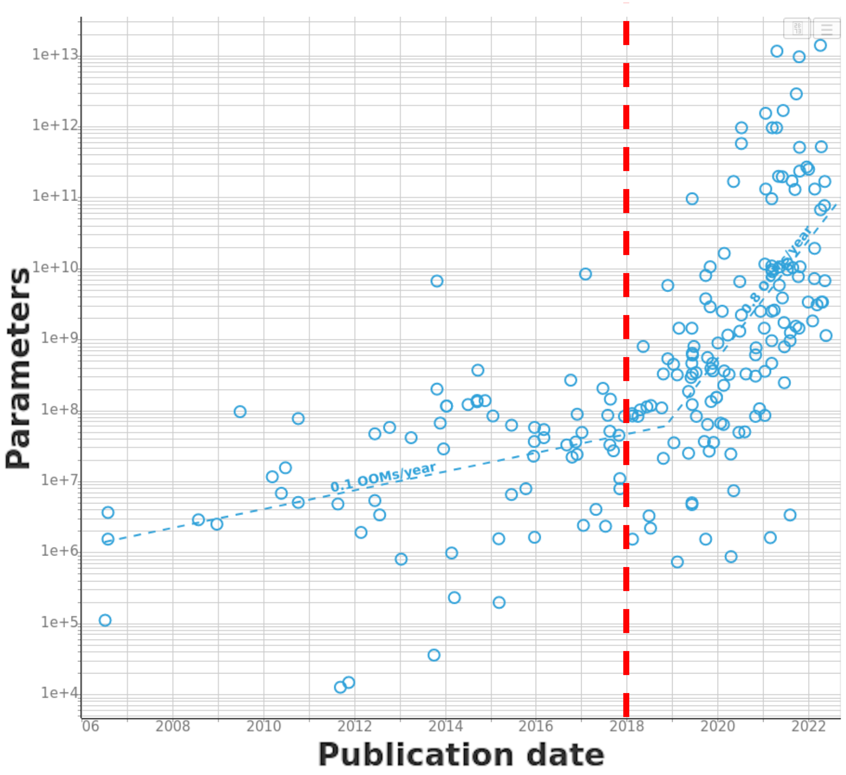
CS7150 Deep Learning

Jiaji Huang

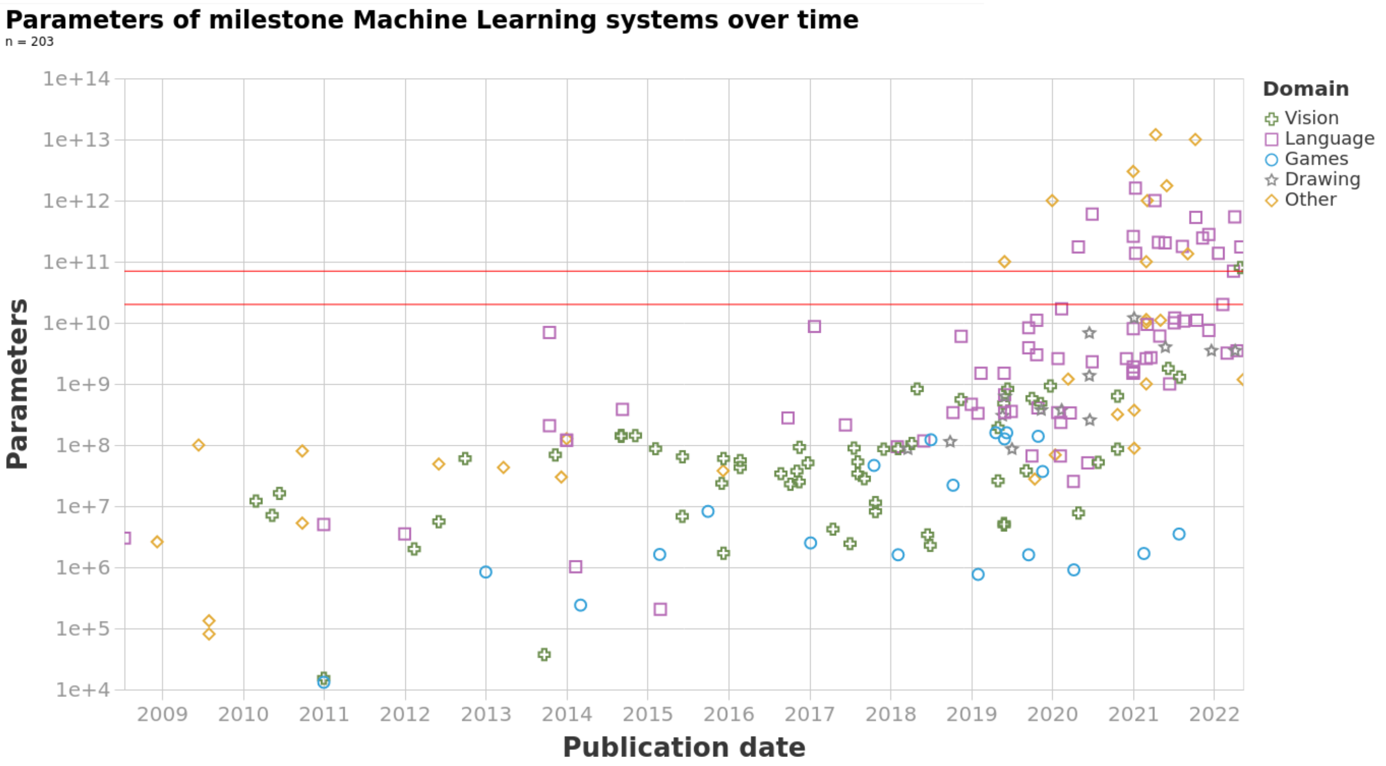
https://jiaji-huang.github.io

04/06/2024

Trend: Model Size Grows Fast



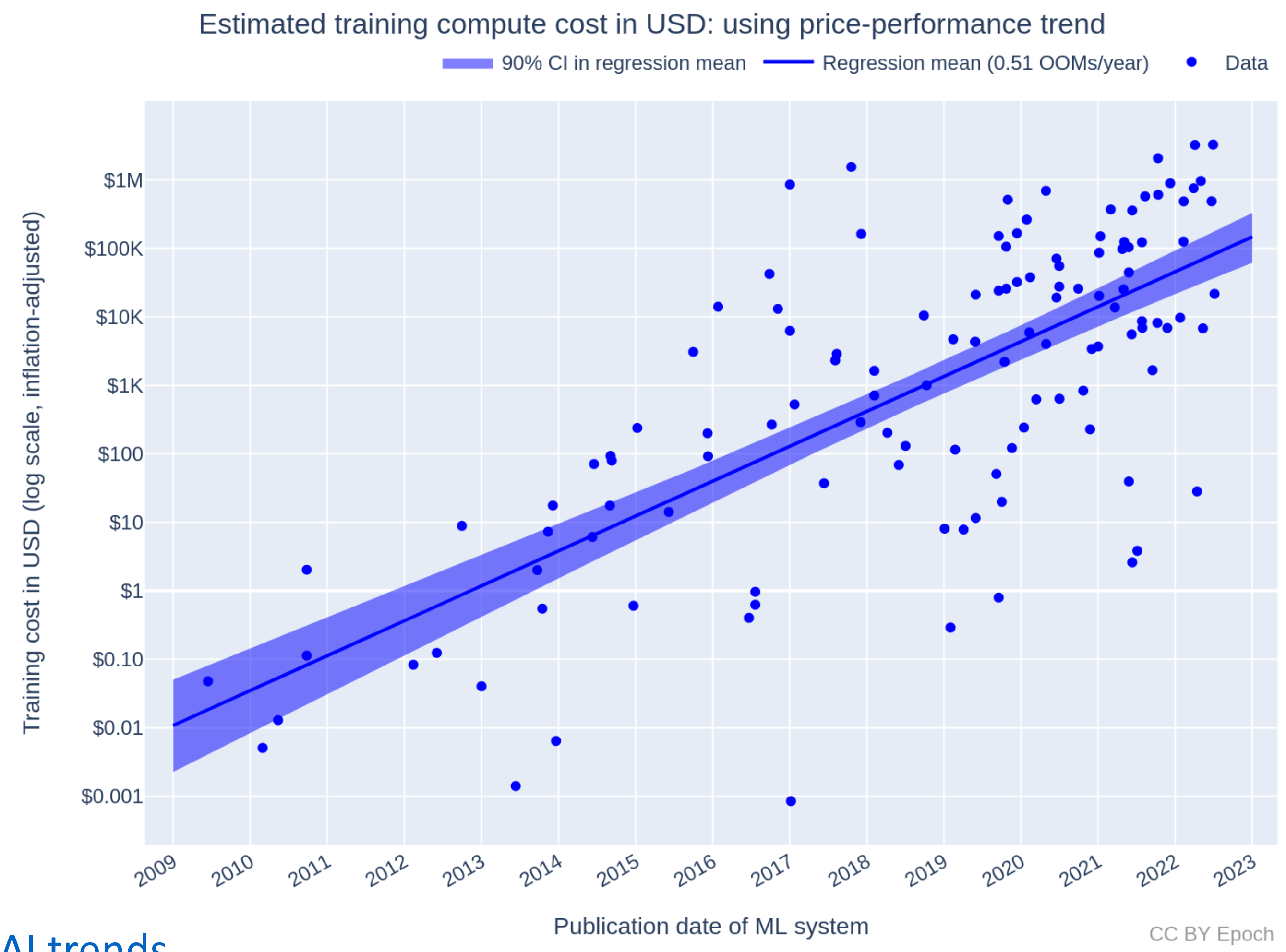
After 2018, parameter counts grow by 4 orders of magnitude, mainly due to language models



Parameter Gap: Starting in 2020, we see many models below 20B parameters and above 70B parameters, but very few in the 20B-70B range.

[EpochAI trends]

Trend: $$$$ to Train



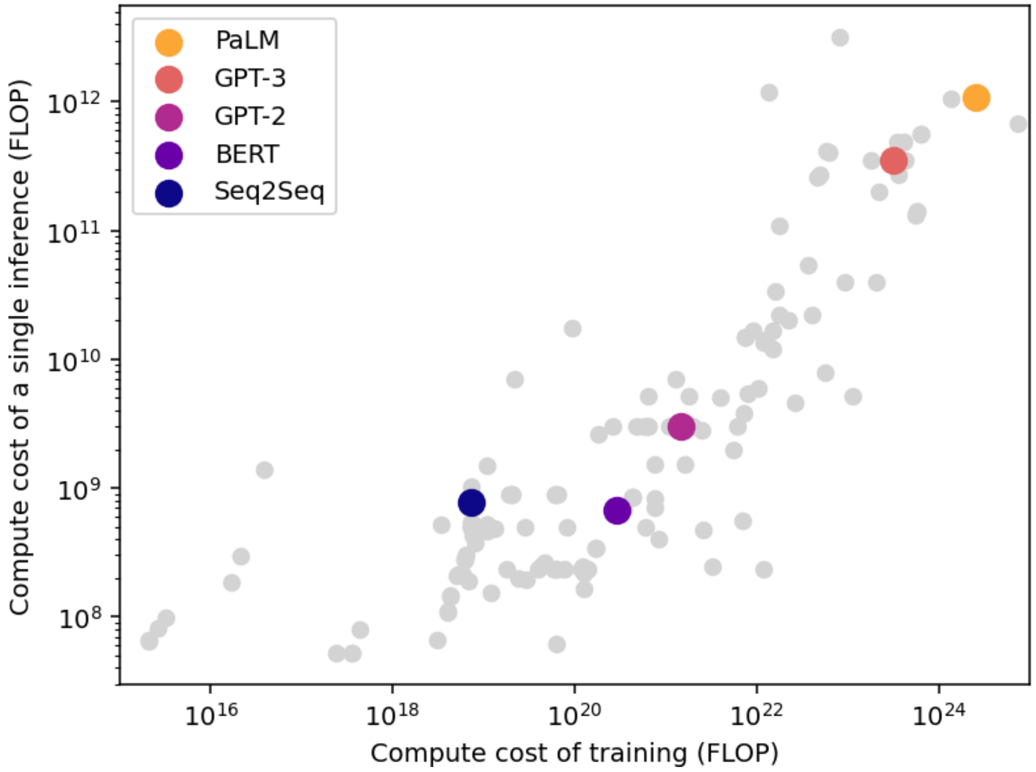
Cost consists of
- Hardware
- Electricity
- Salaries

Gemini Ultra projected to be $630 million to train (most expensive)

EpochAI trends

How about Inference cost?

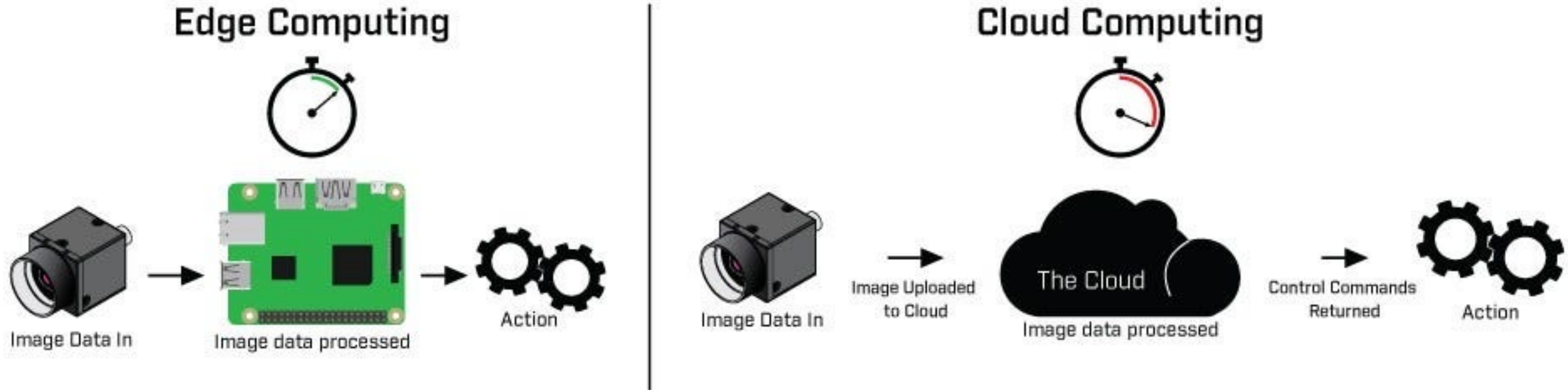
- Single inference cost $\sim\sqrt{training}$
- But over the entire lifetime of a model?
- Up to 90% of total AI cost



EpochAI trends and Desislavov et. al, 2023

Other Issues with Inference

- Choose edge computing due to latency and security
- But also less memory and computational power



Cartoon from [TowardsDataScience]

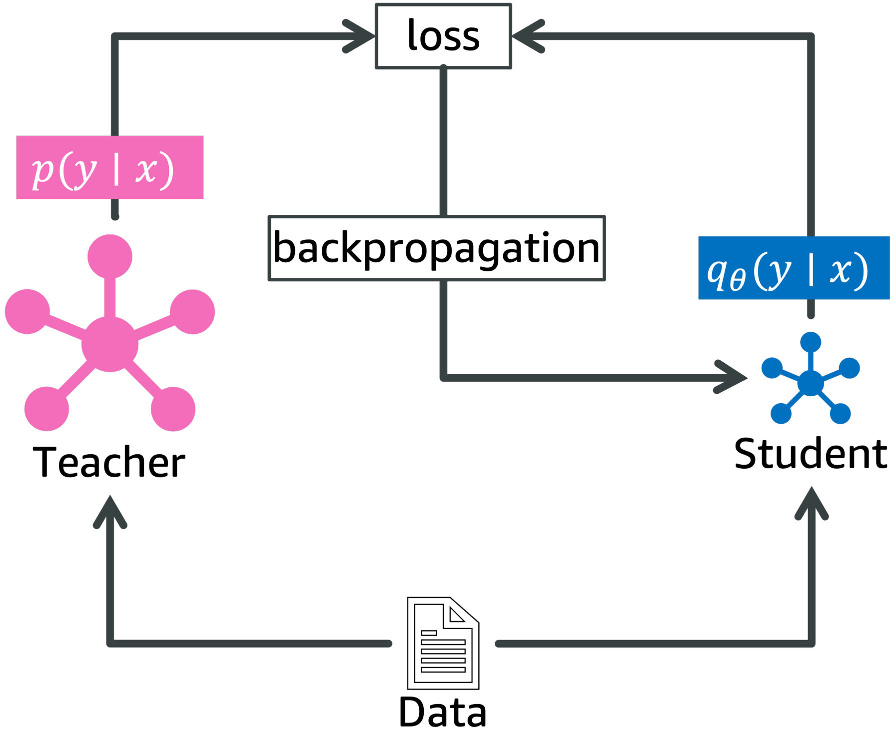
Agenda

- Knowledge Distillation
- Sparsity and Unstructured Pruning
- Structured Pruning
- One-shot Pruning

Knowledge Distillation (KD)

- Ask logits of small model ≈ logits of big model
- Terminologies:
  - Big model: teacher
  - Small model: student
- Treat $p(y|x)$ as "soft" label:

$$\min_{\theta} KL(p||q_\theta) \equiv \ell_{CE}(p, q_\theta) - H(p)$$

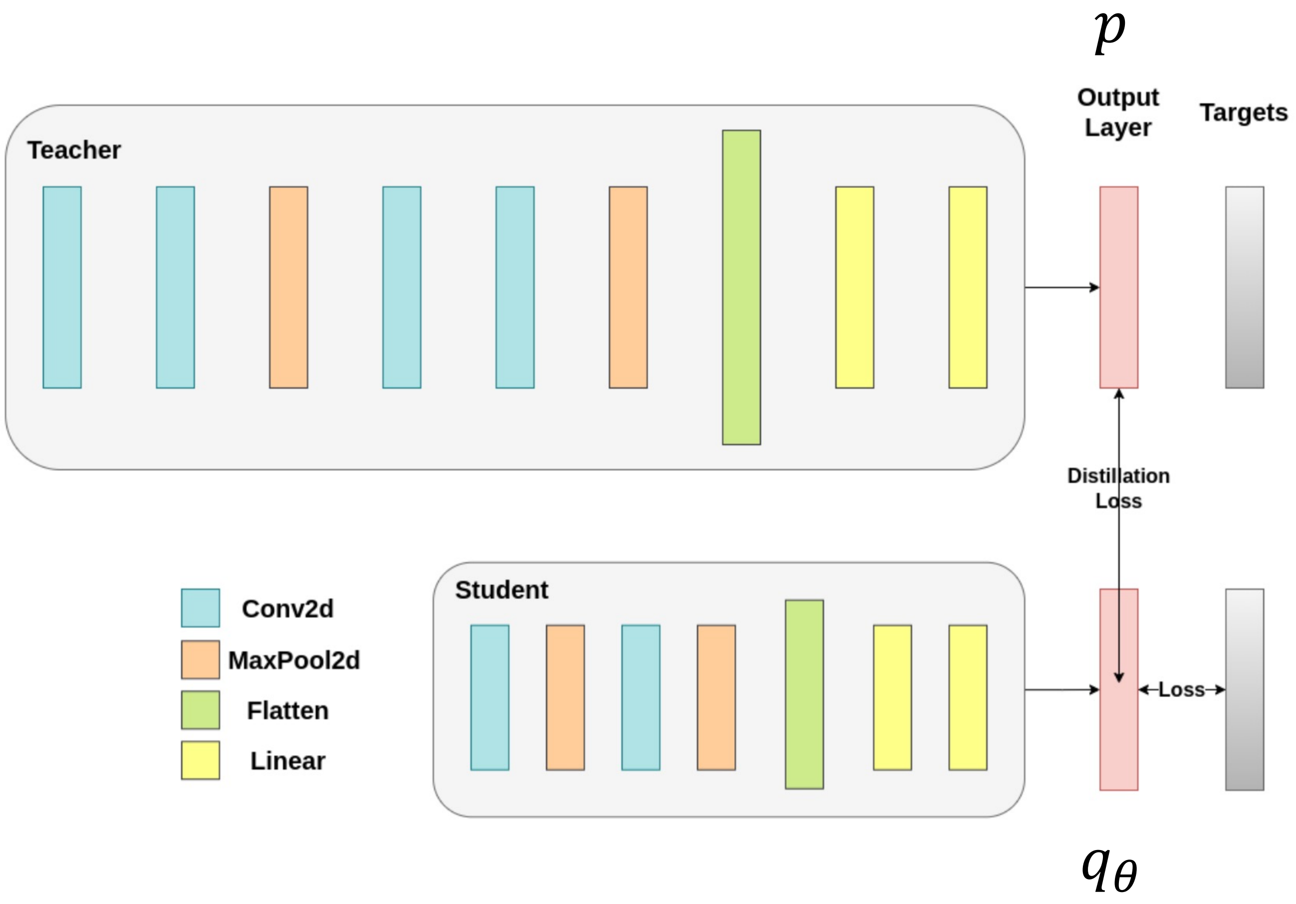


Hinton et. al, 2015

Example: Image Classification

- Treat p as soft labels
- Also use hard label (ground-truth)
- So overall loss is

$$\lambda \ell_{CE}(p, q_\theta) + (1 - \lambda)\ell_{CE}(y, q_\theta)$$
$$0 < \lambda \leq 1$$



Example from [pyTorch tutorial]()

Example: Image Classification

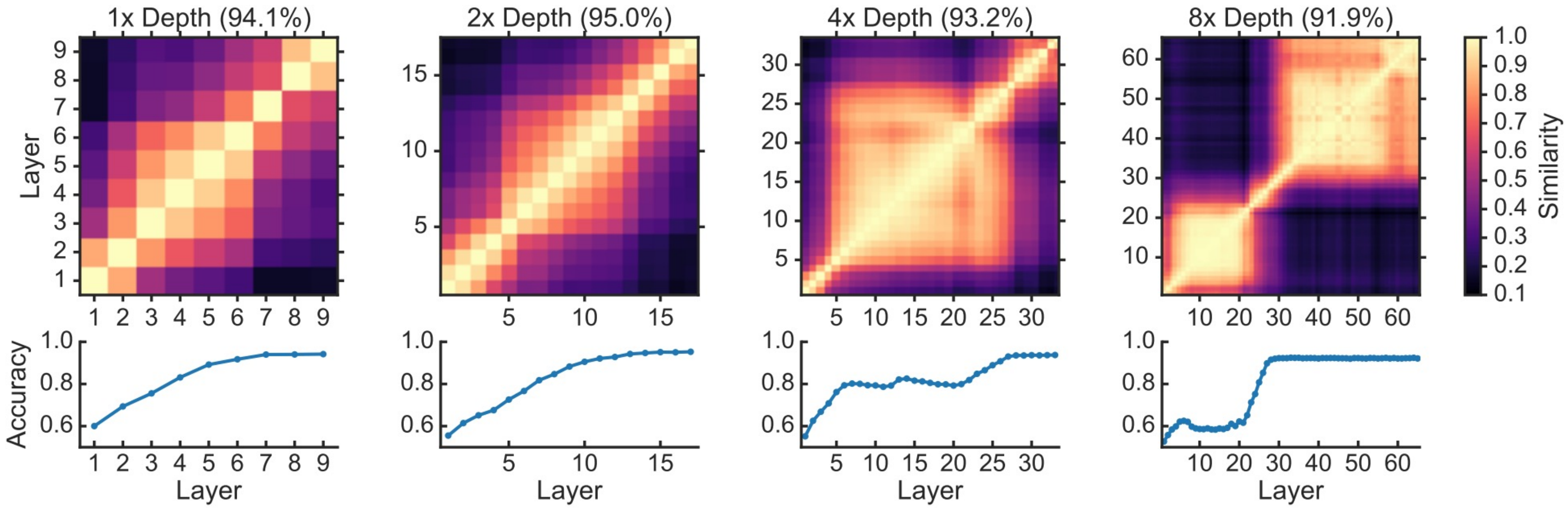
- Teacher in eval mode
- Student in training mode
- may introduce a temperature parameter $T > 0$
- Smaller T: more deterministic
- $T = 0 \equiv \arg\max$
- Optionally add “hard” loss

```python
    teacher.eval()  # Teacher set to evaluation mode
    student.train() # Student to train mode

    for epoch in range(epochs):
        running_loss = 0.0
        for inputs, labels in train_loader:
            inputs, labels = inputs.to(device), labels.to(device)

            optimizer.zero_grad()

            # Forward pass with the teacher model - do not save gradients here as we do not
change the teacher's weights
            with torch.no_grad():
                teacher_logits = teacher(inputs)

            # Forward pass with the student model
            student_logits = student(inputs)

            #Soften the student logits by applying softmax first and log() second
            soft_targets = nn.functional.softmax(teacher_logits / T, dim=-1)
            soft_prob = nn.functional.log_softmax(student_logits / T, dim=-1)

            # Calculate the soft targets loss. Scaled by T**2 as suggested by the authors of
the paper "Distilling the knowledge in a neural network"
            soft_targets_loss = torch.sum(soft_targets * (soft_targets.log() - soft_prob)) /
soft_prob.size()[0] * (T**2)

            # Calculate the true label loss
            label_loss = ce_loss(student_logits, labels)

            # Weighted sum of the two losses
            loss = soft_target_loss_weight * soft_targets_loss + ce_loss_weight * label_loss

            loss.backward()
            optimizer.step()
```

Example from [pyTorch tutorial]()

Discussion

- Benefit of soft labels by teacher? Variance reduction [(Zhou et.al, 2021)](#)
- Train student on the same data as teacher?
- Shallower or narrower student?
- How to decide student's architecture? (Layer similarity index?)



Teacher-Student Gap

- Stronger teacher doesn’t necessarily benefit student
- Teacher label becomes “harder”, not much additional information than ground- truth label
- Too complicated decision boundary for student to mimic

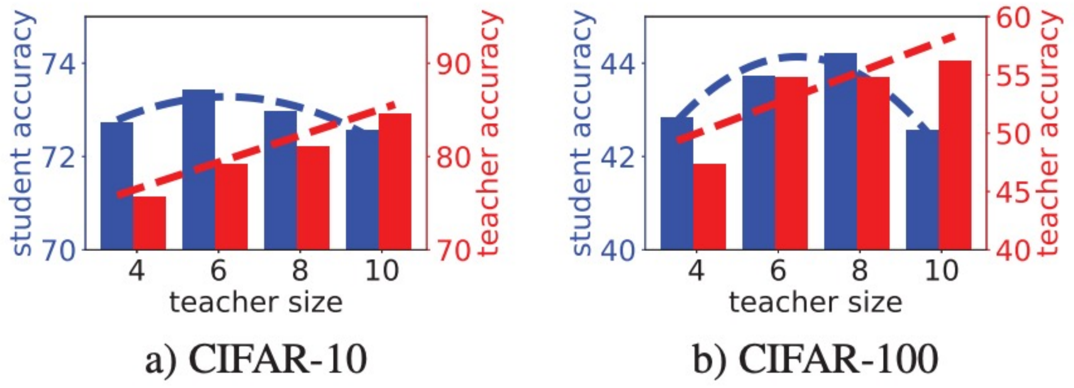


Figure 2: Distillation performance with increasing teacher size. The number of convolutional layers in student is 2.

[Mirzadeh et. al, 2020]

Teacher-Student Gap

- Given the teacher, there is a "best" choice of student architecture

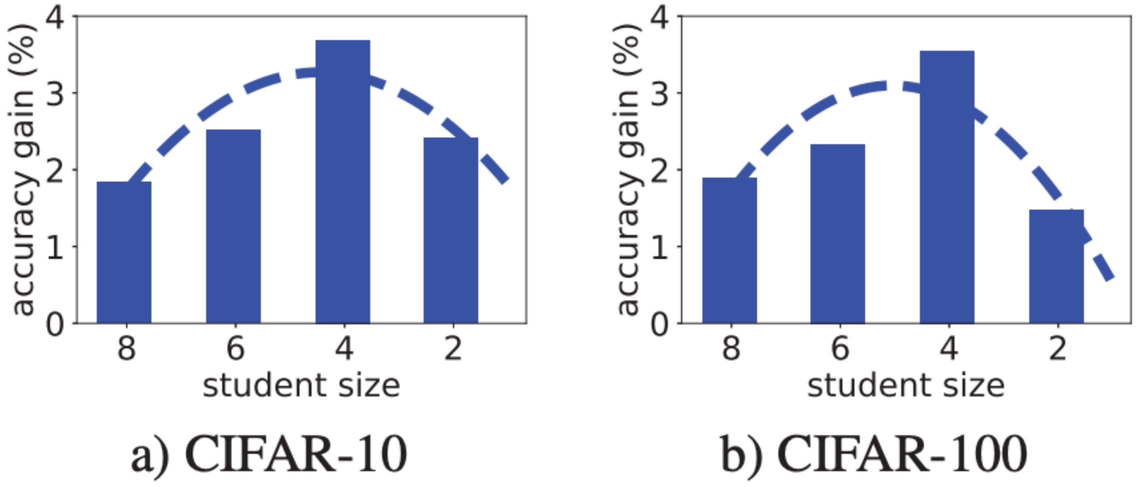


Figure 3: Percentage of distilled student performance increase over the performance when it learns from scratch with varying student size. The teacher has 10 layers.

[Mirzadeh et. al, 2020]

Teaching Assistant

- When the teacher-student gap is too big

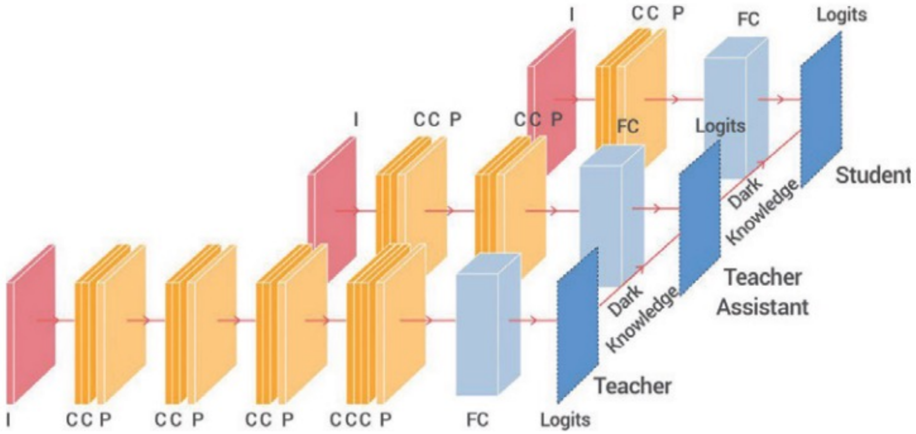


Figure 1: TA fills the gap between student & teacher

Table 2: Student's accuracy given varied TA sizes for (S=2, T=10)

Model	Dataset	TA=8	TA=6	TA=4
CNN	CIFAR-10	72.75	73.15	73.51
	CIFAR-100	44.28	44.57	44.92

Table 3: Student's accuracy given varied TA sizes for (S=8, T=110)

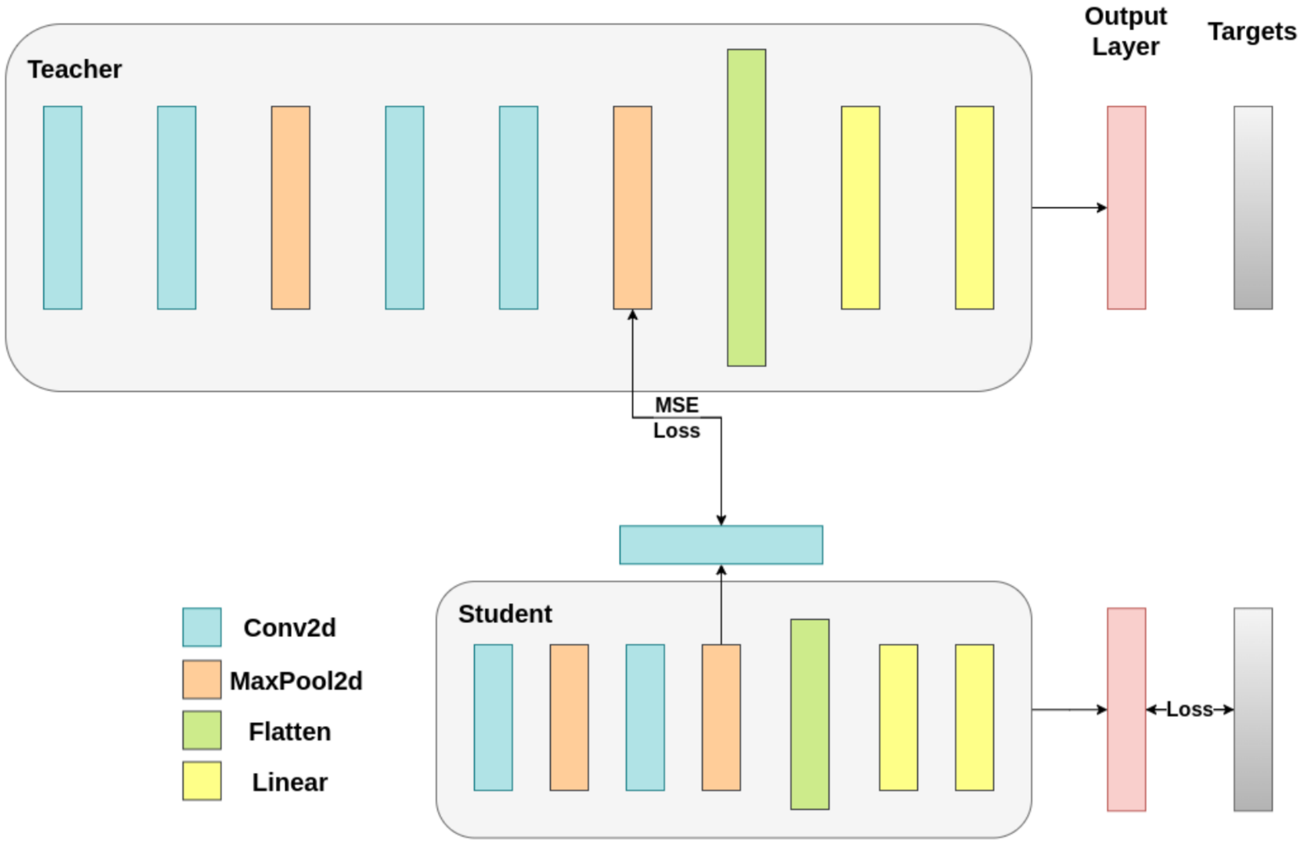
Model	Dataset	TA=56	TA=32	TA=20	TA=14
ResNet	CIFAR-10	88.70	88.73	88.90	88.98
	CIFAR-100	61.47	61.55	61.82	61.5

Mirzadeh et. al, 2020

Other Distillation Criteria

- Optionally ask intermediate features to be close
- May introduce a projector if feature dimensions differ

$$\min_{\theta,W} ||Wf_t - f_s(\theta)||^2$$



KD for Pretrained LMs: Distilbert

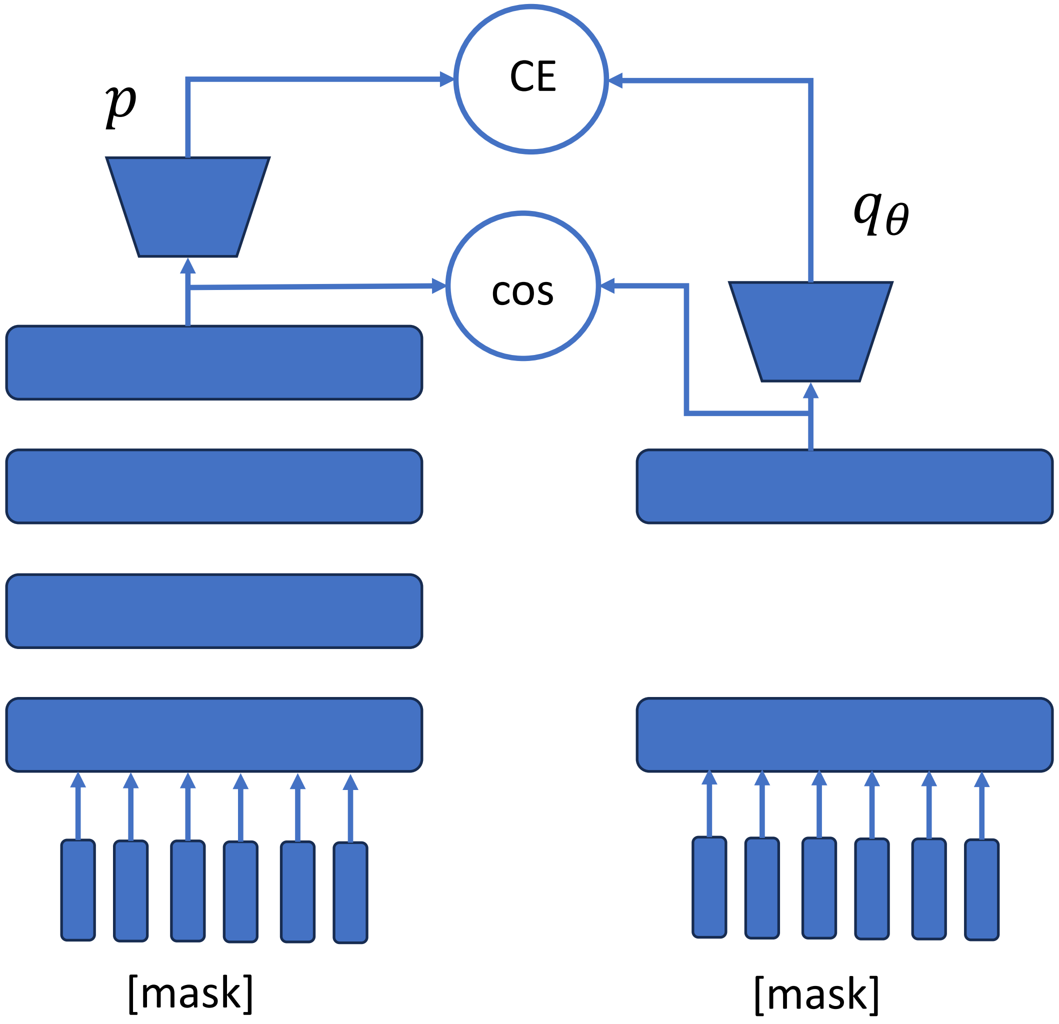
- Student architecture
  - half as deep as teacher
  - Initialized from every other layer of teacher
  - Keep same hidden dimensions
- Training loss

$$\ell_{CE}(p, q_\theta) - \alpha \cos(h_t, h_s)$$

- Training data:

  Same pretraining material as BERT

[Sanh, et. al, 2020]()



KD for Pretrained LMs: Distilbert

- Evaluation 1: Retains accuracy

Table 1: **DistilBERT retains 97% of BERT performance.** Comparison on the dev sets of the GLUE benchmark. ELMo results as reported by the authors. BERT and DistilBERT results are the medians of 5 runs with different seeds.

Model	**Score**	CoLA	MNLI	MRPC	QNLI	QQP	RTE	SST-2	STS-B	WNLI
ELMo	68.7	44.1	68.6	76.6	71.1	86.2	53.4	91.5	70.4	56.3
BERT-base	79.5	56.3	86.7	88.6	91.8	89.6	69.3	92.7	89.0	53.5
DistilBERT	77.0	51.3	82.2	87.5	89.2	88.5	59.9	91.3	86.9	56.3

- Evaluation 2: faster

Table 3: **DistilBERT is significantly smaller while being constantly faster.** Inference time of a full pass of GLUE task STS-B (sentiment analysis) on CPU with a batch size of 1.

Model	# param. (Millions)	Inf. time (seconds)
ELMo	180	895
BERT-base	110	668
DistilBERT	66	410

Sanh, et. al, 2020

A few Variants

- Patient KD (Sun, et. al, 2019): Some variations on training loss
  - Cos loss for intermediate hidden features
  - Downstream task-specific training loss
- Tiny BERT (Jiao, et. al, 2020): task specific distillation

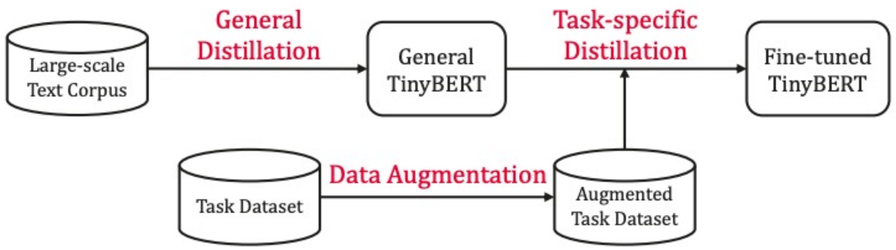
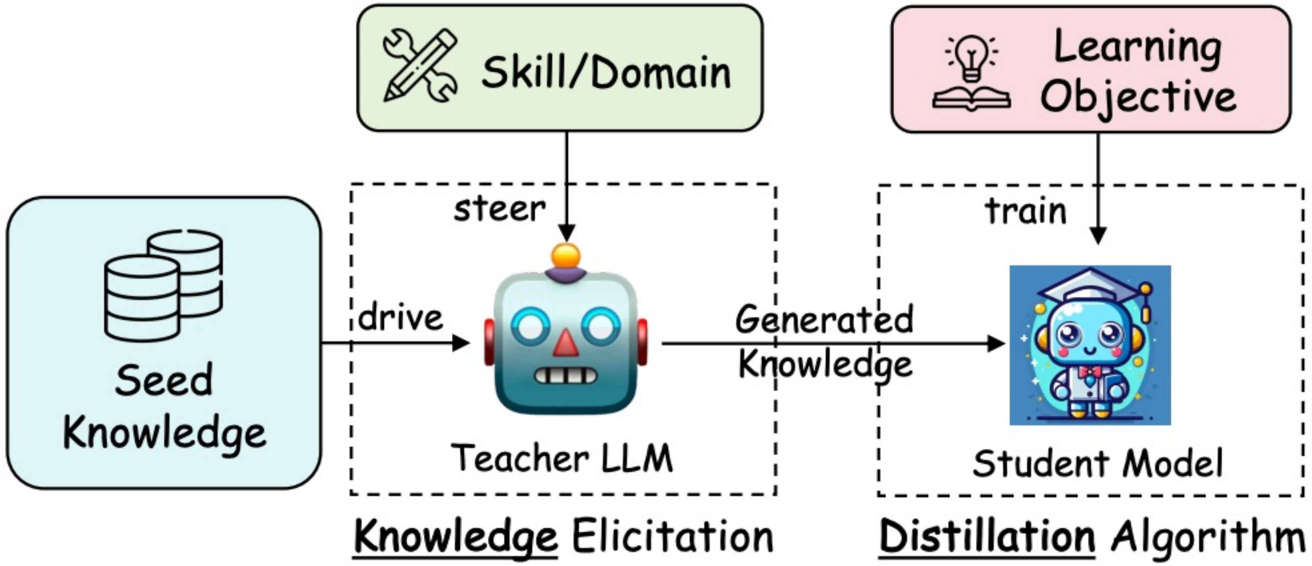


Figure 1: The illustration of TinyBERT learning.

- DistilGPT-2: Same as distilbert but for decoder model

KD for LLM

- LLM's training data is proprietary



[Xu, et. al, 2024]

More on KD loss

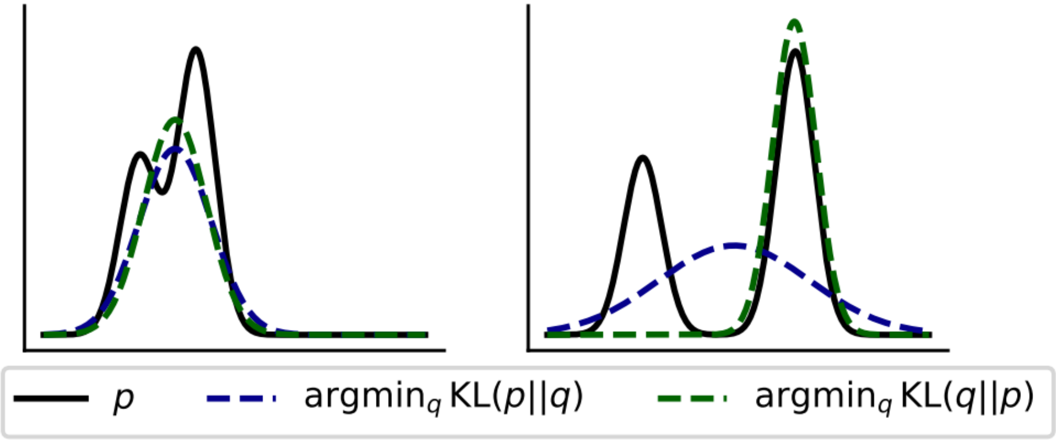
- Forward KL is commonly seen

$$\min_\theta KL(p||q_\theta) \equiv \ell_{CE}(p, q_\theta) - H(p)$$

- Backward KL

$$\min_\theta KL(q_\theta||p) \equiv \ell_{CE}(q_\theta, p) - H(q_\theta)$$

- Coverage vs. preciseness
- Choice should be task-dependent
    - Machine translation, less modes, forward KL
    - Dialog, more modes, backward KL



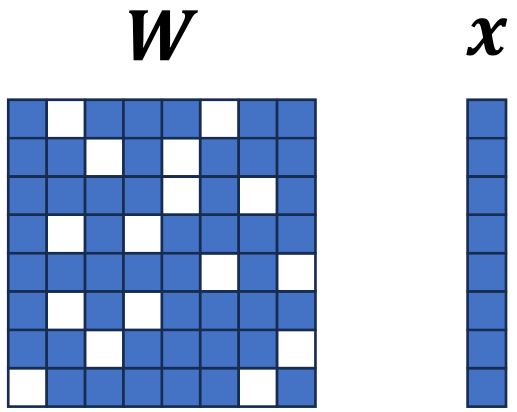
Forward KL: covers all mode of p, less precise
Backward KL: mode seeking

Agenda

- Knowledge Distillation
- Sparsity and Unstructured Pruning
- Structured Pruning
- One-shot Pruning

Unstructured pruning

- Force weights to be 0. Sparsity pattern is unstructured
- e.g., $\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x}$ prune A to 20% sparsity but in a unstructured way



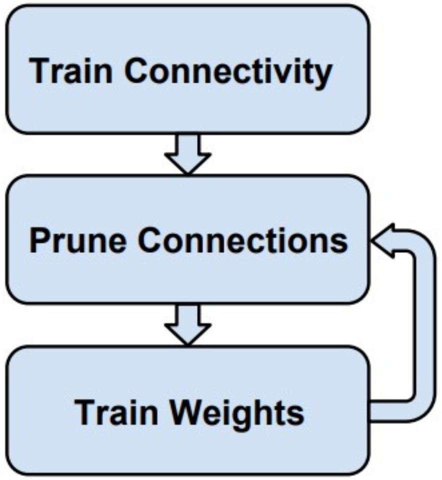
- Saves storage, but not necessarily speedup
- As GPU is good at dense matrix operations

Magnitude Based Approach

- Recipe:

1. Train as usual
2. Set w_i to 0 if $|w_i|$ small
3. Keep the unpruned weights, and further train (don't re-initialize!)

2-3 can be repeated for multiple rounds (suggested)



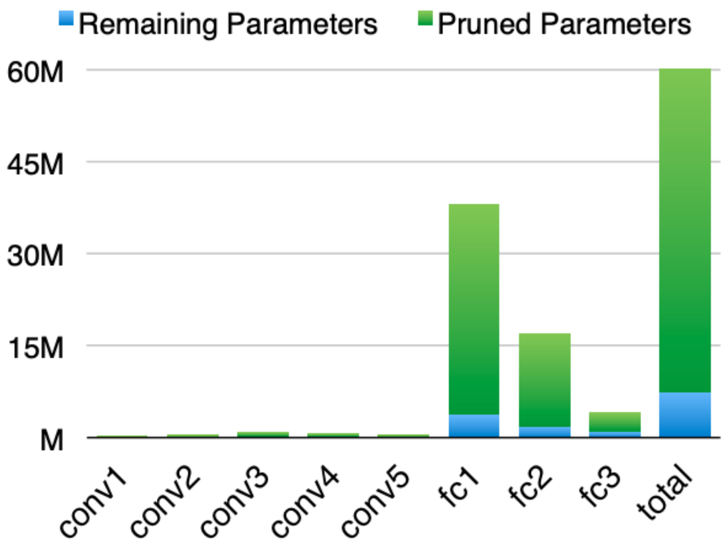
- Q: why not just impose ℓ_1 regularization at training?

Han, et. al, 2015

Magnitude Based Approach

- Reduced number of parameters Significantly

Layer	Weights	FLOP	Act%	Weights%	FLOP%
conv1	35K	211M	88%	84%	84%
conv2	307K	448M	52%	38%	33%
conv3	885K	299M	37%	35%	18%
conv4	663K	224M	40%	37%	14%
conv5	442K	150M	34%	37%	14%
fc1	38M	75M	36%	9%	3%
fc2	17M	34M	40%	9%	3%
fc3	4M	8M	100%	25%	10%
Total	61M	1.5B	54%	**11%**	**30%**

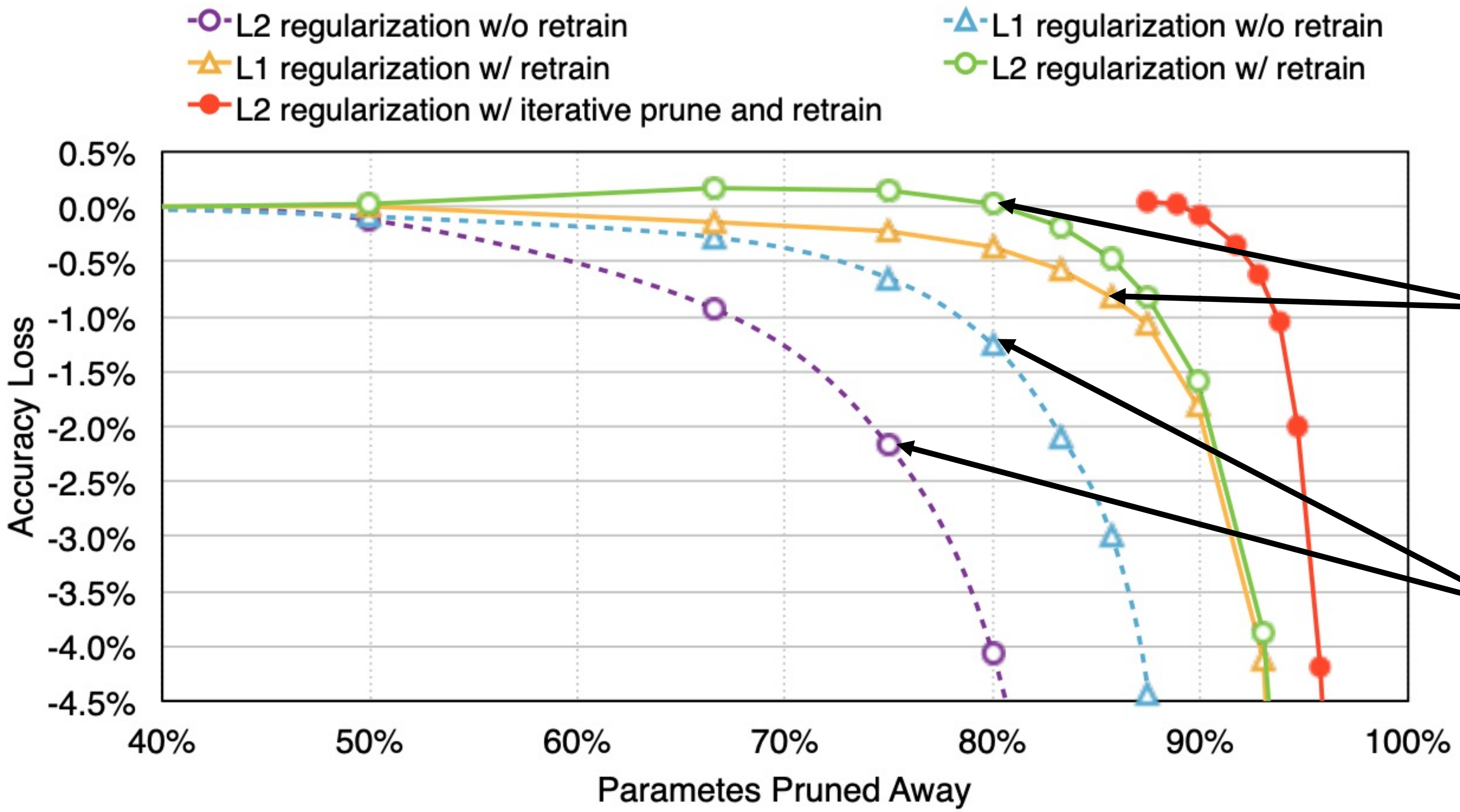


Pruning AlexNet: reduces the number of weights by 9× and computation by 3×

Han, et. al, 2015

Magnitude Based Approach

- Accuracy-efficiency trade-off

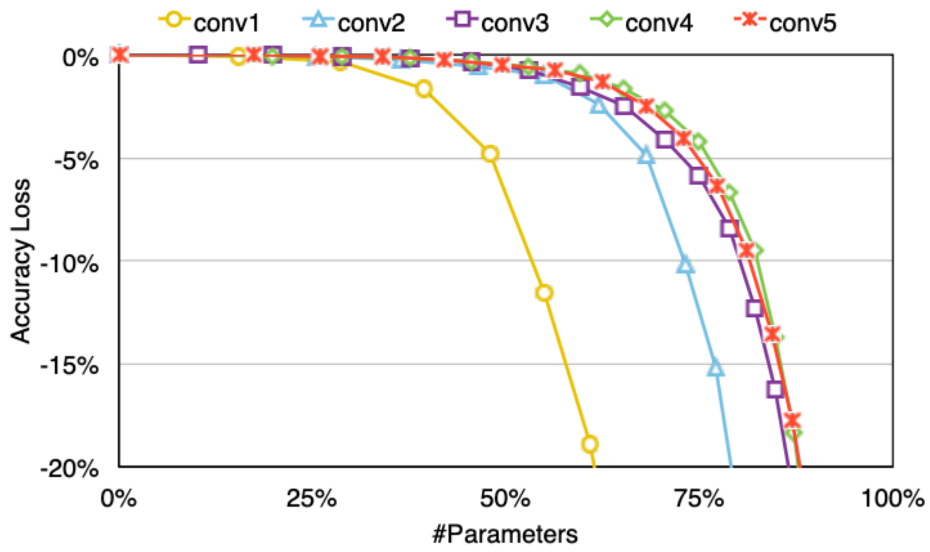


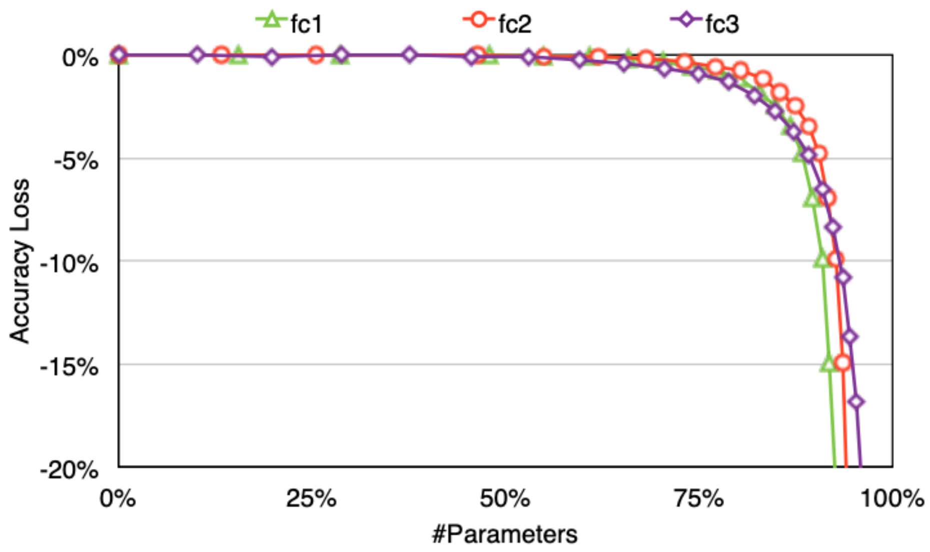
<u>with retraining:</u> weight decay better than ℓ_1

<u>without retraining: ℓ_1</u> better than weight decay

Magnitude Based Approach

- Trade-off at different layers
- Top layer is more prunable





Hessian Based: Optimal Brain Damage (OBD)

- Perturb network weights, $\boldsymbol{w} \to \boldsymbol{w}'$. Denote $\boldsymbol{\delta} \triangleq \boldsymbol{w}' - \boldsymbol{w}$
- Loss on all training data $\mathcal{L}(\boldsymbol{w}) \triangleq \frac{1}{N}\sum_{n=1}^{N} \ell(\boldsymbol{w}, \boldsymbol{x}_n)$, change of loss is:

$$\Delta_{\mathcal{L}} \triangleq \mathcal{L}(\boldsymbol{w}') - \mathcal{L}(\boldsymbol{w}) \approx \cancelto{0}{\langle \nabla\mathcal{L}(\boldsymbol{w}), \boldsymbol{\delta} \rangle} + \frac{1}{2}\boldsymbol{\delta}^T \boldsymbol{H} \boldsymbol{\delta}$$

- $\nabla\mathcal{L}(\boldsymbol{w}) = \boldsymbol{0}$ at local minimum
- $\boldsymbol{H}_{i,j} = \frac{\partial^2 \mathcal{L}}{\partial w_i \partial w_j}$ is Hessian and $\Delta_{\mathcal{L}} = \frac{1}{2}\sum_{i,j} H_{i,j}\delta_i\,\delta_j$
- Note: H is # param $\times$ # param, and requires some samples to estimate

Lecun, et. al, 1989

OBD (contd.)

- Diagonalize: set $H_{i,j} = 0$ for $i \neq j$
- So $\Delta_{\mathcal{L}} = \sum_i H_{i,i}\, \delta_i^2$
- If we sparsify w_i to 0, then $\delta_i = -w_i$, and $\Delta_{\mathcal{L}} = \sum_i H_{i,i}\, w_i^2$
- So we have an importance score for all w_i's:

$$H_{i,i} w_i^2 \equiv \frac{\partial^2 \ell}{\partial w_i^2} \cdot w_i^2$$

- Set $w_i = 0$ if $\left|H_{i,i} w_i^2\right|$ small

Discussion

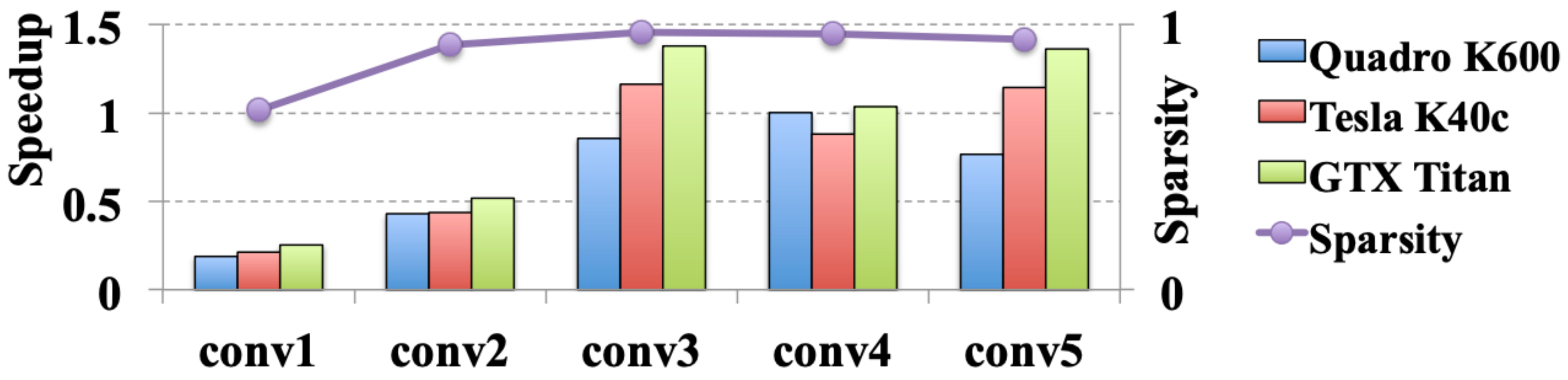
- Magnitude based vs. Hessian based, which one is better?
- Hard to answer. Performance depends on
  - Network structure
  - Pruning percentage
  - Sample data to estimate the Hessian

Agenda

- Knowledge Distillation
- Sparsity and Unstructured Pruning
- Structured Pruning
- One-shot Pruning

Unstructured Sparsity ≠ Speedup

Unstructured sparsity pattern induces irregular memory access

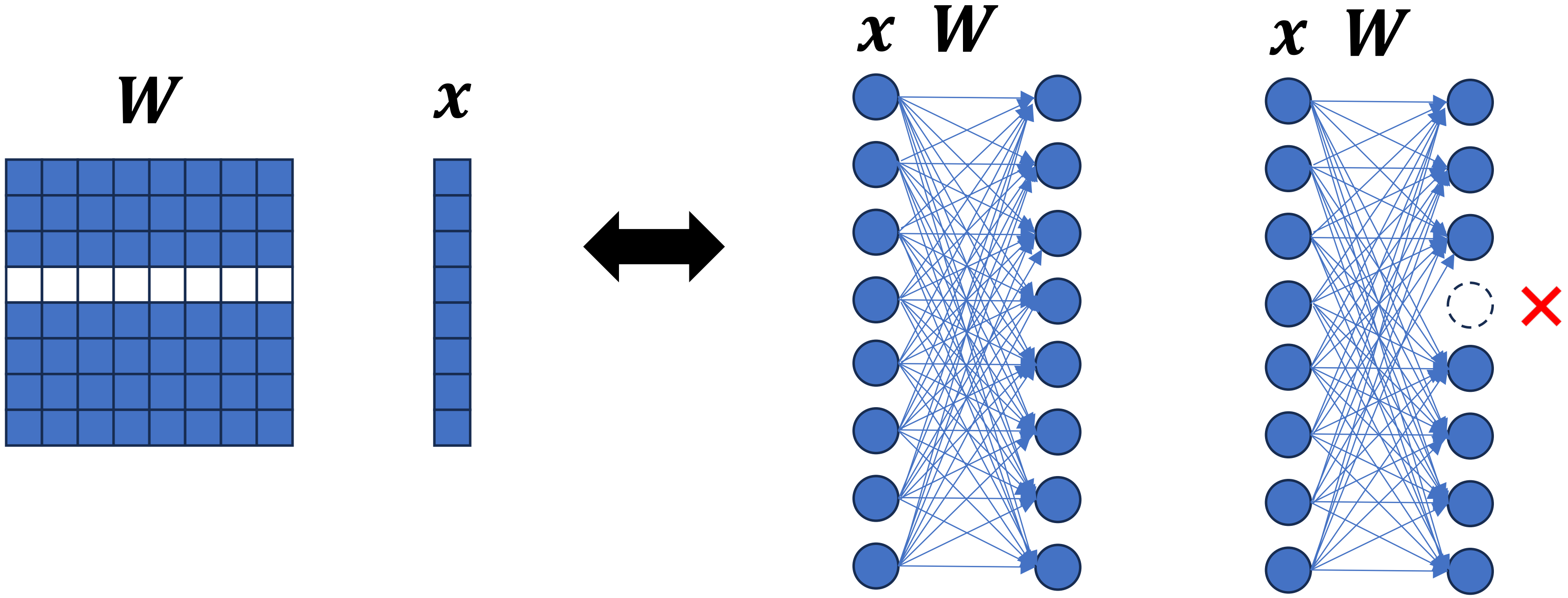


AlexNet on GPU platforms and the sparsity. 95% sparsity leads to <1.5 speedup only.

Wen, et. al, 2016

Structured Sparsity for Linear Layers

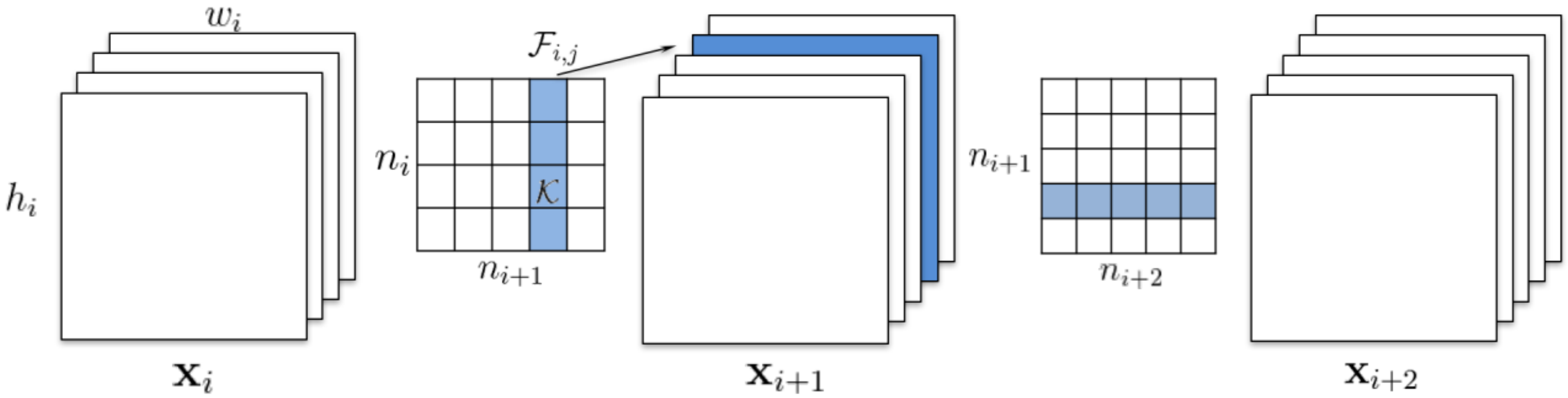
- Remove a row in matrix $\boldsymbol{W}$ amounts to remove an output neuron



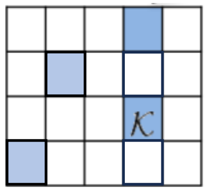
- Remove a neuron at input?
- Remove a neuron in a middle layer?

Structured Sparsity for Conv layers

- Pruning a 3D-filter amounts to remove an output channel



- Finer granularity: Pruning a 2D filter



Li, et. al, 2017, Wen, et. al, 2016

Magnitude Based Approach

Recipe

1. Determine sparsity pattern:

   Choice 1: Set groups with small magnitude ($\sum_{i \in \mathcal{G}} |w_i|$) to 0 (Li, et. al, 2017)

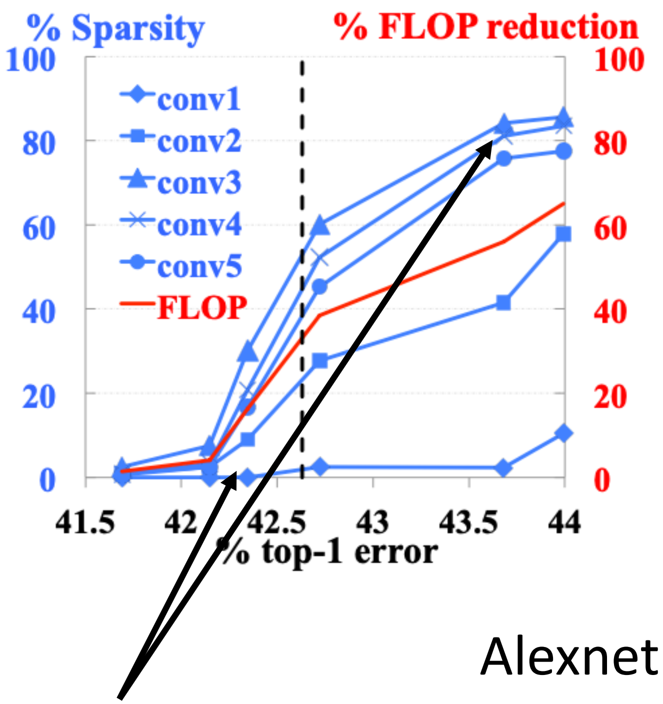
   Choice 2: Couple with training (Wen et. al, 2016):

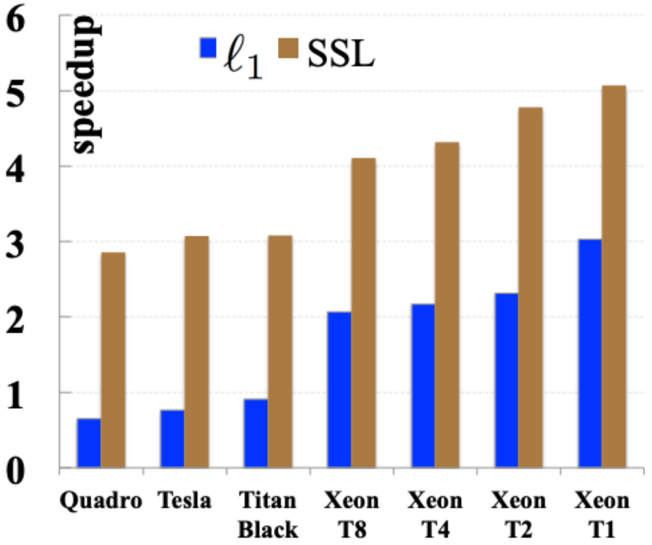
   - Train with original loss, plus a group lasso regularization: $\sum_{\mathcal{G}} \sqrt{\sum_{i \in \mathcal{G}} w_i^2}$
   - Converge to sparse groups of weights

2. Fix the sparse pattern, and retrain remaining weights
3. Optionally repeat 1-2

Compare against Unstructured Pruning

- Notable speedup



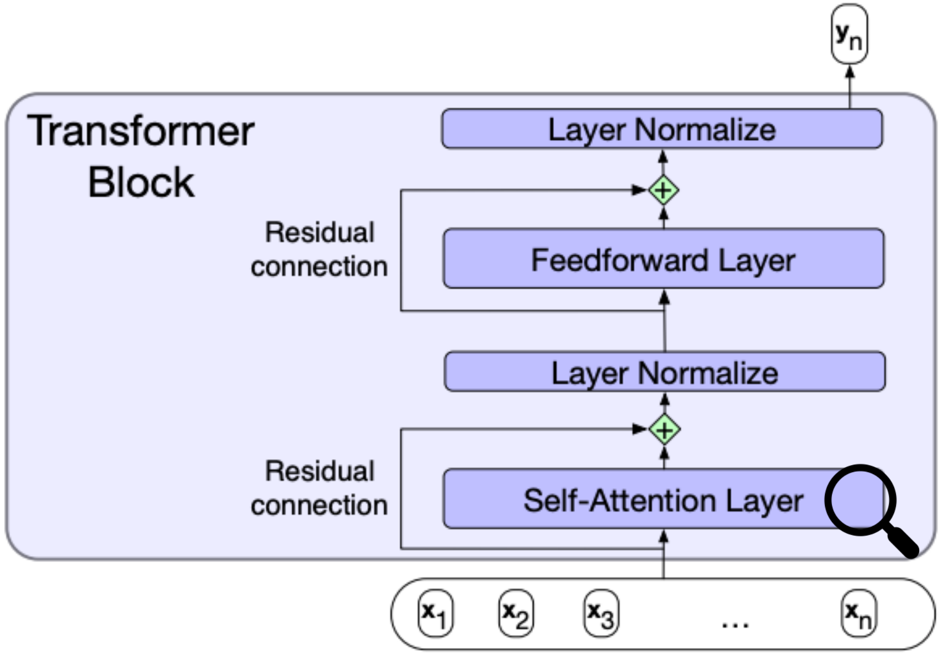


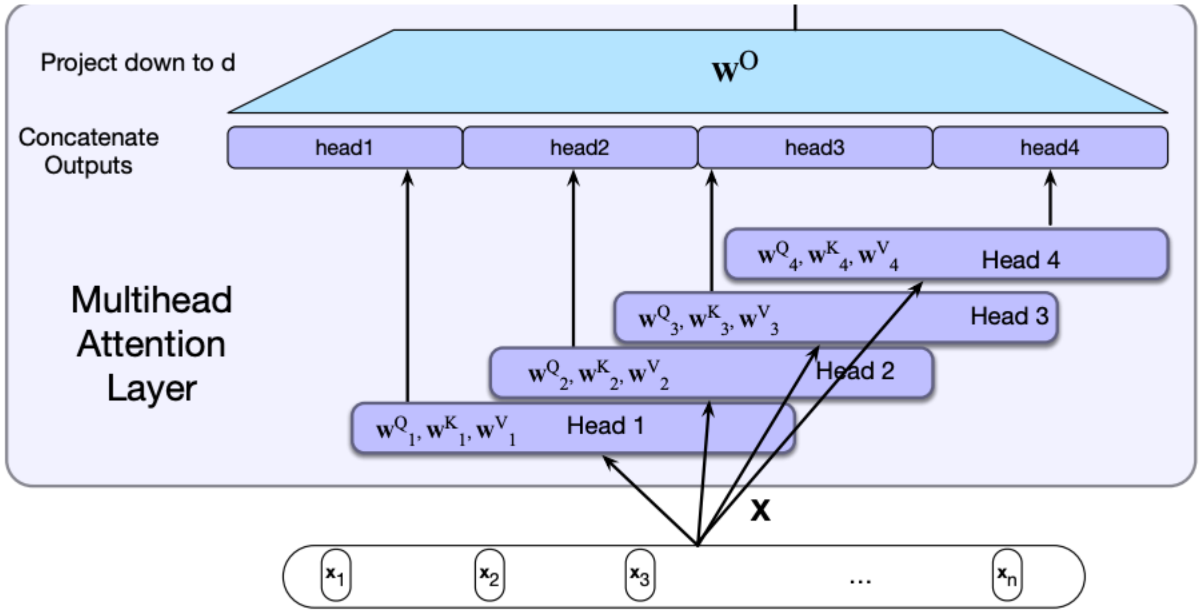
Alexnet on ImageNet

Deeper layers are more prunable

Structured Sparsity On Transformer Block

- Recap of a transformer block





Formalize Attention Heads

- Each attention head output a vector $f_i(\cdot;\boldsymbol{W}_i^Q,\boldsymbol{W}_i^K,\boldsymbol{W}_i^V)$
- Overall output is their linear combination

$$\boldsymbol{O}=\boldsymbol{W}^O\begin{bmatrix} f_1(\cdot;\boldsymbol{W}_1^Q,\boldsymbol{W}_1^K,\boldsymbol{W}_1^V) \\ \vdots \\ f_H(\cdot;\boldsymbol{W}_H^Q,\boldsymbol{W}_H^K,\boldsymbol{W}_H^V) \end{bmatrix}$$

$$=\sum_{i=1}^{H}\left\{\boldsymbol{W}_i^O f_i(\cdot;\boldsymbol{W}_i^Q,\boldsymbol{W}_i^K,\boldsymbol{W}_i^V)\equiv Att_i(\cdot;\boldsymbol{W}_i)\right\}$$

Where $\boldsymbol{W}_i\triangleq\{\boldsymbol{W}_i^O,\boldsymbol{W}_i^Q,\boldsymbol{W}_i^K,\boldsymbol{W}_i^V\}$

Removal of Attention Heads

- Magnitude based Pruning?

$$|\boldsymbol{W}_i^O| + |\boldsymbol{W}_i^Q| + |\boldsymbol{W}_i^K| + |\boldsymbol{W}_i^V|$$

- Smaller weights are not necessarily less important
- Hessian based approach? But how?
- Introduce a mask variable m_i for each head

$$\boldsymbol{O} = \sum_{h=1}^{H} m_i Att_i(\cdot, \boldsymbol{W}_i)$$

- Derive 2nd order gradient w.r.t. m_i

Kwon et. al, 2022

Agenda

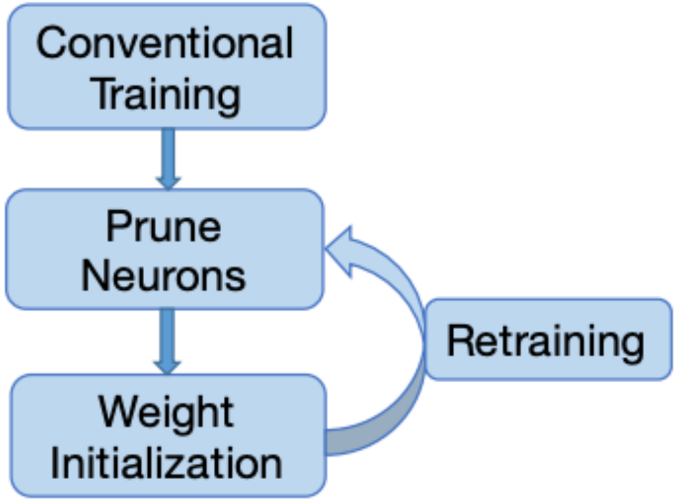
- Knowledge Distillation
- Sparsity and Unstructured Pruning
- Structured Pruning
- One-shot Pruning

Recap the General Pruning Recipe

1. Prune (set to 0 according to some importance score)
2. Adjust remaining weights by re-training
3. Iterate 1-2

Magnitude based

Hessian based



Get rid off retraining?

One-shot Prune with Hessian

- Once we set a $w_i = 0$, adjust w_j $(j \neq p)$ to:

$$\min_{\boldsymbol{\delta}} \left\{ \Delta \equiv \frac{1}{2} \boldsymbol{\delta}^T \boldsymbol{H} \boldsymbol{\delta} \right\}, \text{ s.t. } \delta_i = -w_i$$

- Introduce Lagrange multiplier λ

$$\mathbb{L}(\lambda, \boldsymbol{\delta}) = \frac{1}{2} \boldsymbol{\delta}^T \boldsymbol{H} \boldsymbol{\delta} + \lambda \left(\boldsymbol{e}_i^T \boldsymbol{\delta} + w_i \right)$$

- Set $\mathbb{L}'(\boldsymbol{\delta}) = 0 \Longrightarrow \ \boldsymbol{\delta}^* = -\lambda \boldsymbol{H}^{-1} \boldsymbol{e}_i$
- Plug $\boldsymbol{\delta}^*$ back into $\mathbb{L}$, $\mathbb{L}(\lambda) = -\frac{\lambda^2}{2} \boldsymbol{e}_i^T \boldsymbol{H}^{-1} \boldsymbol{e}_i + \lambda w_i$
- Solve $\mathbb{L}'(\lambda) = 0 \Longrightarrow \lambda^* = w_i / [\boldsymbol{H}^{-1}]_{i,i}$

Singh et. al, 2020

One-shot Prune with Hessian

- $\boldsymbol{\delta}^* = -\lambda \boldsymbol{H}^{-1}\boldsymbol{e}_i$ and $\lambda^* = w_i / [\boldsymbol{H}^{-1}]_{i,i}$
- So $\boldsymbol{\delta}^* = -\dfrac{w_i [\boldsymbol{H}^{-1}]_{\cdot,i}}{[\boldsymbol{H}^{-1}]_{i,i}}$ → i-th column of $\boldsymbol{H}^{-1}$
- Correspondingly, $\Delta^* = \dfrac{w_i^2}{2[\boldsymbol{H}^{-1}]_{i,i}}$ (Note $\boldsymbol{H}[\boldsymbol{H}^{-1}]_{\cdot,i} = \boldsymbol{e}_i$)
- Importance score for weight w_i:

$$\frac{w_i^2}{2[\boldsymbol{H}^{-1}]_{i,i}}$$

Q: how to recover the OBD method by assuming special form of $\boldsymbol{H}$?

Singh et. al, 2020

An Example: Layer-wise One-shot Pruning

- Prune weight of a linear layer, $\boldsymbol{W}$ to sparse $\widehat{\boldsymbol{W}}$, so that

$$\min_{\widehat{\boldsymbol{W}}} \left\| \boldsymbol{W}\boldsymbol{X} - \widehat{\boldsymbol{W}}\boldsymbol{X} \right\|^2$$

- Rows are independent. So just consider $\min \mathcal{L}_i \equiv \left(\widehat{\boldsymbol{W}}_{i,:}\boldsymbol{X} - \boldsymbol{W}_{i,:}\boldsymbol{X}\right)^2$
- We can apply results before: as
  - $\widehat{\boldsymbol{W}}_{i,j} = \boldsymbol{W}_{i,j} + \delta_j$ where $\delta_j = -\boldsymbol{W}_{i,j}$ for some $\boldsymbol{W}_{i,j}$ to be zero-ed
- Hessian w.r.t. $\boldsymbol{W}_{i,:}$ is $\boldsymbol{C} = \boldsymbol{X}\boldsymbol{X}^T$ (input covariance)
- Importance score for $W_{i,j}$: $\dfrac{W_{i,j}^2}{[\boldsymbol{C}^{-1}]_{j,j}}$

Frantar et. al, 2023 and sparseGPT

More Simplified

Wanda (Pruning by **W**eights **and** activations)

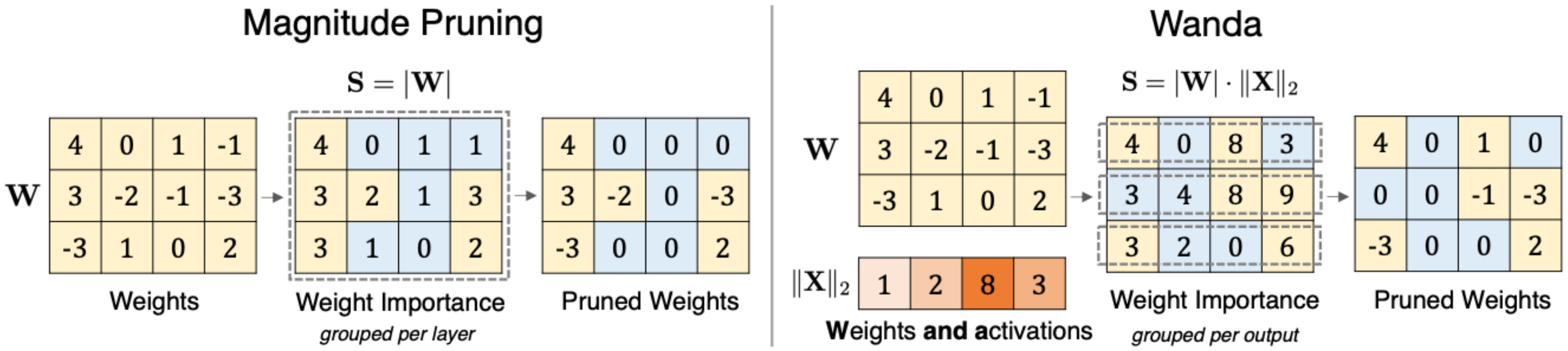
- What if we assume C diagonal?

$$[\boldsymbol{C}^{-1}]_{j,j} = \boldsymbol{C}^{-1}{}_{j,j} = \left\|\boldsymbol{X}_{j,:}\right\|^{-2}$$

- So importance score is

$$\frac{W_{i,j}^2}{[\boldsymbol{C}^{-1}]_{j,j}} = W_{i,j}^2 \left\|\boldsymbol{X}_{j,:}\right\|^2$$

Sun, et. al, 2023



Wanda (contd.)

Method	Weight Update	Sparsity	LLaMA 7B	LLaMA 13B	LLaMA 30B	LLaMA 65B	LLaMA-2 7B	LLaMA-2 13B	LLaMA-2 70B
Dense	-	0%	59.99	62.59	65.38	66.97	59.71	63.03	67.08
Magnitude	✗	50%	46.94	47.61	53.83	62.74	51.14	52.85	60.93
SparseGPT	✓	50%	**54.94**	58.61	63.09	66.30	**56.24**	60.72	**67.28**
Wanda	✗	50%	54.21	**59.33**	**63.60**	**66.67**	**56.24**	**60.83**	67.03
Magnitude	✗	4:8	46.03	50.53	53.53	62.17	50.64	52.81	60.28
SparseGPT	✓	4:8	**52.80**	55.99	60.79	64.87	**53.80**	**59.15**	65.84
Wanda	✗	4:8	52.76	**56.09**	**61.00**	**64.97**	52.49	58.75	**66.06**
Magnitude	✗	2:4	44.73	48.00	53.16	61.28	45.58	49.89	59.95
SparseGPT	✓	2:4	**50.60**	**53.22**	58.91	62.57	**50.94**	54.86	63.89
Wanda	✗	2:4	48.53	52.30	**59.21**	**62.84**	48.75	**55.03**	**64.14**

Table 2: Mean zero-shot accuracies (%) of pruned LLaMA and LLaMA-2 models. Wanda performs competitively against prior best method SparseGPT, without introducing any weight update.

Sun, et. al, 2023

Approximation of Hessian

- Assume network learns the true $p(y|\boldsymbol{x};\boldsymbol{w})$
- The loss function is therefore $\ell(\boldsymbol{w};(\boldsymbol{x},y)) = -\log p(y|\boldsymbol{x};\boldsymbol{w})$
- A known result is

$$\mathbb{E}_{(\boldsymbol{x},y)}\left[\frac{\partial^2}{\partial\boldsymbol{w}\partial\boldsymbol{w}^T}\{-\log p(y|\boldsymbol{x};\boldsymbol{w})\}\right]$$
$$= \mathbb{E}_{(\boldsymbol{x},y)}\left[\left(\frac{\partial}{\partial\boldsymbol{w}}\{\log p(y|\boldsymbol{x};\boldsymbol{w})\}\right)\left(\frac{\partial}{\partial\boldsymbol{w}}\{\log p(y|\boldsymbol{x};\boldsymbol{w})\}\right)^T\right]$$

- Namely, Fisher Information matrix

Singh et. al, 2020

Approximation of Hessian

- Discrete format

$$\widehat{\boldsymbol{H}} = \frac{1}{N}\sum_{n=1}^{N} \nabla\ell(\boldsymbol{w}; (\boldsymbol{x}_n, y_n))\nabla\ell(\boldsymbol{w}; (\boldsymbol{x}_n, y_n))^T$$

- Note: for a network trained to reach local minima,

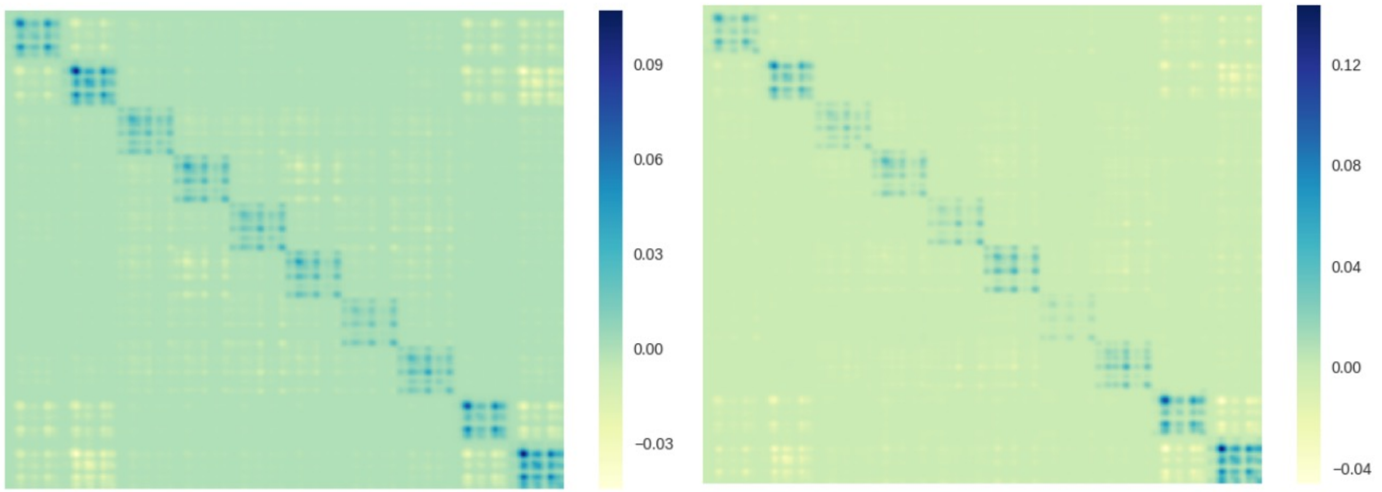
$$\frac{1}{N}\sum_{n=1}^{N} \nabla\ell(\boldsymbol{w}; (\boldsymbol{x}_n, y_n)) = 0$$

But the averaged outer product of gradient is **NOT 0**

Singh et. al, 2020

Approximation of Hessian

- But $\boldsymbol{H}$ is still too big
- Assume block-diagonal, blocks defined by network layers



Left: True Hessian; Right: the $\widehat{\boldsymbol{H}}$

Singh et. al, 2020

Approximation of Inverse Hessian

- Denote $\nabla\ell(\boldsymbol{w};(\boldsymbol{x}_n, y_n)) \triangleq \boldsymbol{g}_n$, add introduce diagonal loading

$$\widehat{\boldsymbol{H}} = \frac{1}{N}\sum_{n=1}^{N} \boldsymbol{g}_n \boldsymbol{g}_n^T + \lambda \boldsymbol{I}$$

- Define recursion $\widehat{\boldsymbol{H}}_n = \widehat{\boldsymbol{H}}_{n-1} + \frac{1}{N}\boldsymbol{g}_n \boldsymbol{g}_n^T$, where $\widehat{\boldsymbol{H}}_n = \lambda \boldsymbol{I}$. So $\widehat{\boldsymbol{H}} = \widehat{\boldsymbol{H}}_N$
- Woodbury matrix identity (Sherman–Morrison formula)

Define $\boldsymbol{v}_n = \widehat{\boldsymbol{H}}_{n-1}^{-1}\boldsymbol{g}_n$

$$\widehat{\boldsymbol{H}}_n^{-1} = \widehat{\boldsymbol{H}}_{n-1}^{-1} - \frac{\boldsymbol{v}_n \boldsymbol{v}_n^T}{N + \boldsymbol{g}_n^T \boldsymbol{v}_n}, \text{ where } \widehat{\boldsymbol{H}}_0^{-1} = \lambda^{-1}\boldsymbol{I}$$

- Apply the above for each block along diagonal

Singh et. al, 2020

Full Recipe: WoodFisher

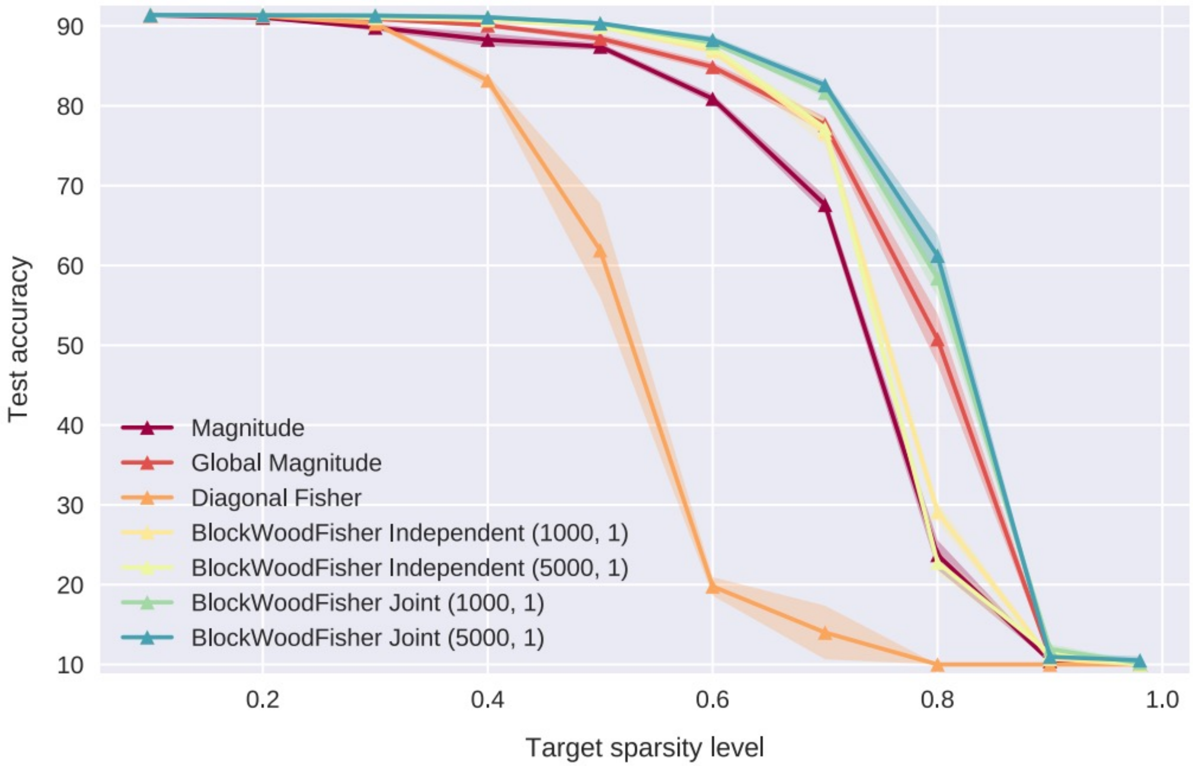
- Take N samples to estimate the block-diagonal $\widehat{\boldsymbol{H}}$
- Apply Sherman–Morrison formula to invert each diagonal block of $\widehat{\boldsymbol{H}}$
- Calculate weight importance score: $\frac{w_i^2}{2[\boldsymbol{H}^{-1}]_{i,i}}$
- Set least significant w_i's to 0. Denote the collection of these i's as $\mathcal{P}$
- Adjust remaining weights by

$$\sum_{i \in \mathcal{P}} -\frac{w_i [\boldsymbol{H}^{-1}]_{\cdot,i}}{[\boldsymbol{H}^{-1}]_{i,i}}$$

Singh et. al, 2020

Comparison

- One-shot, compare against
  - Magnitude based
  - OBD (diagonal Fisher)
- Variants of WoodFisher
  - Independent: Rank scores in each layer
  - Joint: Rank scores for all weights



One-shot prune of resnet20 trained on cifar10

Singh et. al, 2020