Sparsity – 1

EECE695D: Efficient ML Systems

Spring 2025

Agenda

- **Last Class**
  - Matmuls
  - Computation vs. Memory
- **W2 & W3**
  - Reducing computation & memory at the matmul level
- **Today**
  - Sparsity & Pruning

Basic idea

Goal

- We want to reduce the computational cost of matrix multiplication
  - Well-trained linear model with $d_{\text{in}} = d_{\text{out}} = 3$ and the dataset size $N = 3$

$$\mathbf{WX} = \begin{bmatrix} w_1 & w_2 & w_3 \\ w_4 & w_5 & w_6 \\ w_7 & w_8 & w_9 \end{bmatrix} \begin{bmatrix} x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 \\ x_7 & x_8 & x_9 \end{bmatrix}$$

  - **Compute.** $2 d_{\text{in}} d_{\text{out}} N$ = 54 FLOPs
  - **Memory I/O.** (loading weights) 3×3 FP32 weights = 36 Bytes

Sparsity

- Remove less important entries of the weight, thus skipping associated ops
  - Suppose that we "prune out" 4 entries, to get a 5-sparse matrix

$$\mathbf{W}_{\text{pruned}}\mathbf{X} = \begin{bmatrix} w_1 & w_2 & 0 \\ 0 & w_5 & 0 \\ w_7 & 0 & w_9 \end{bmatrix} \begin{bmatrix} x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 \\ x_7 & x_8 & x_9 \end{bmatrix}$$

  - Fancily put, we take a Hadamard product with some mask matrix

$$\mathbf{W}_{\text{pruned}} = \mathbf{M} \odot \mathbf{W}, \qquad \mathbf{M} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

Quiz. The matrix $\mathbf{W}_{\text{pruned}}$ has …

(a) 44.4% Sparsity

(b) 55.5% Sparsity

$$\begin{bmatrix} w_1 & w_2 & 0 \\ 0 & w_5 & 0 \\ w_7 & 0 & w_9 \end{bmatrix}$$

Advantages

$$\begin{bmatrix} w_1 & w_2 & 0 \\ 0 & w_5 & 0 \\ w_7 & 0 & w_9 \end{bmatrix} \begin{bmatrix} x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 \\ x_7 & x_8 & x_9 \end{bmatrix}$$

- Compute & memory decreases proportionally to the sparsity
  - **Compute.** $(1 - \text{sparsity}) \times (\text{dense FLOPs}) =$ 30 FLOPs
  - **Memory I/O.** $(1 - \text{sparsity}) \times (\text{dense I/O}) =$ 20 Bytes

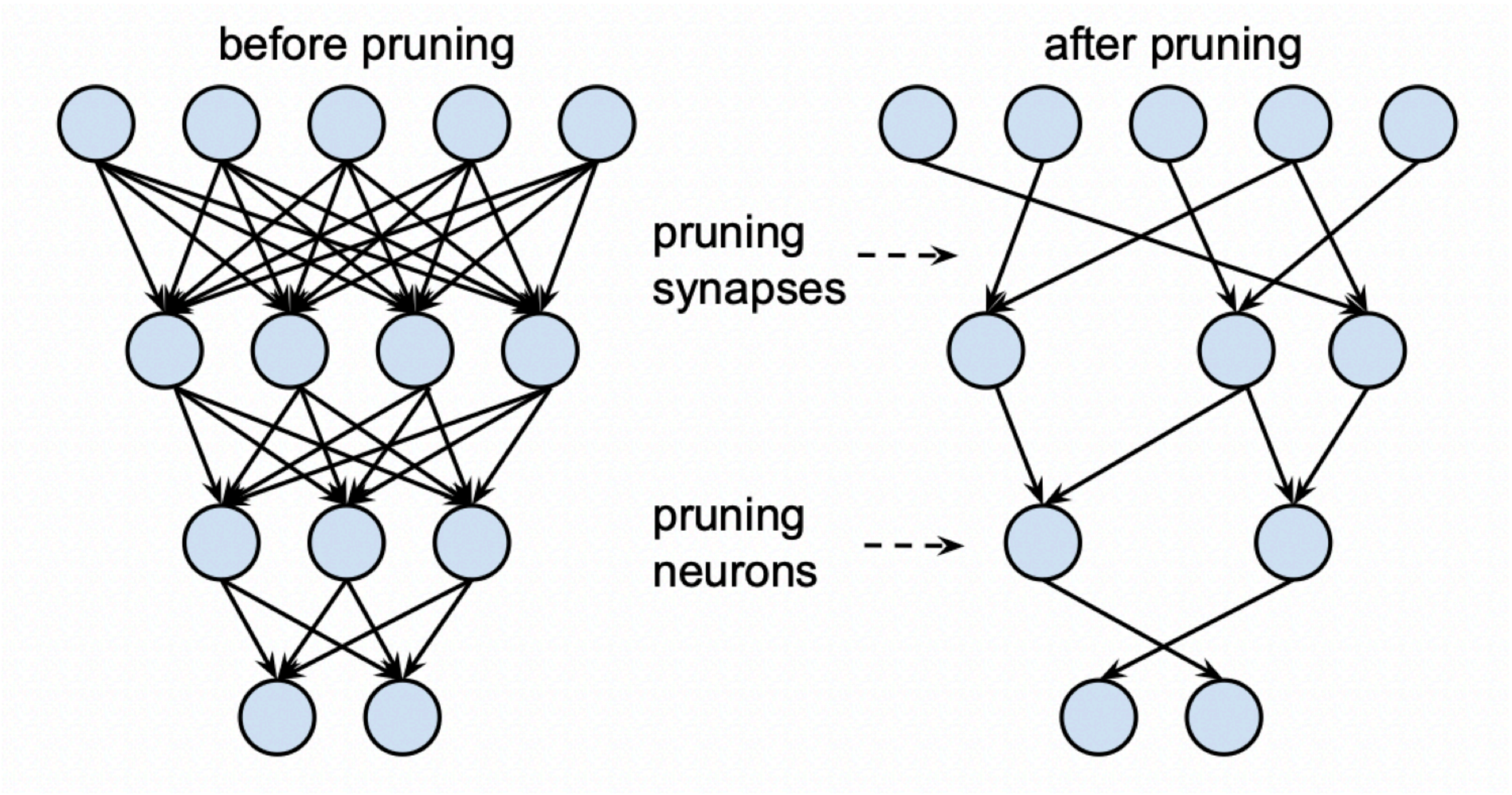
Note. There are certain overheads, as we will see in the next class

Problem formulation

Problem formulation

- We apply pruning at a model level
  - Layer 1 pruned to xx% sparsity
  - Layer 2 pruned to yy% sparsity
  - (...)

  ⇒ Model achieves zz% global sparsity



- Typically, one can achieve 20%—80% global sparsity without accuracy drop

Han et al., “Learning both weights and connections for efficient neural networks,” NeurIPS 2015

Problem formulation

- Minimize the **training risk** of the pruned model, given the **sparsity constraint**

$$\text{minimize}_{\mathbf{w}_{\text{pruned}}} \quad \hat{L}(\mathbf{w}_{\text{pruned}}) \qquad \text{subject to} \quad \|\mathbf{w}_{\text{pruned}}\|_0 \leq \tau$$

  - $\mathbf{w}$: all neural net weights, vectorized
  - $\hat{L}(\,\cdot\,)$: training risk
  - $\|\cdot\|_0$: ℓ_0 norm (i.e., the number of nonzero entries)
  - τ: sparsity constraint

Note. We are using a global sparsity constraint, just for simplicity.

Problem formulation

- Alternatively, view it as a **joint optimization** of weights and mask

$$\text{minimize}_{\mathbf{m},\mathbf{w}} \quad \hat{L}(\mathbf{m} \odot \mathbf{w})$$
$$\text{subject to} \quad \|\mathbf{m}\|_0 \leq \tau, \quad m_{ij} \in \{0,1\}$$

- By doing so, we have decomposed this into two **subproblems**
    - Optimizing $\mathbf{w}$: Unconstrained, continuous optimization
    - Optimizing $\mathbf{m}$: Constrained, discrete optimization
        - Tricky part!

Algorithm

Algorithm

- Typical pruning algorithms solve this via **alternating optimization**:

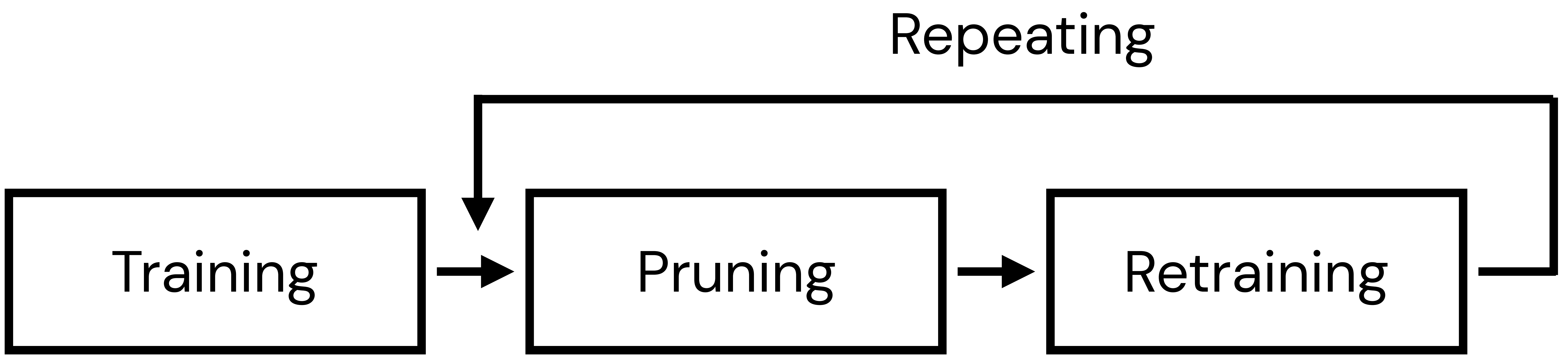
  1. Training. Train the model for some steps (optimize w)

  2. Pruning. Remove some weights, using some criterion (fix w, optimize m)

  3. Retraining. Retrain the model for some steps, to recover from damage. (fix m, optimize w)

  4. Repeating. Repeat steps 2–3 for some iterations

  (Note. there are other relaxation-based optimization algorithms as well)



Algorithm

- Two key elements:
  - **Saliency.** How can we identify less important weight?
  - **Schedule.** When do we introduce the sparsity?
    - Often reflects operational constraints, e.g., training cost

1. Training. Train the model for some steps

2. Pruning. Remove some weights, using some criterion

3. Retraining. Retrain the model for some steps, to recover from damage.

4. Repeating. Repeat steps 2–3 for some iterations

Saliency

Saliency

- At the pruning phase, we are solving the **mask optimization**:

$$\text{minimize}_{\mathbf{m}} \quad \hat{L}(\mathbf{m} \odot \mathbf{w}) \quad \text{subject to} \quad \|\mathbf{m}\|_0 \leq \tau, \quad m_i \in \{0,1\}$$

- This is **NP-hard**, and thus we typically rely on heuristics
    - Hessian
    - Gradient
    - Magnitude
- That is, we compute these “scores” and simply prune out bottom-K.

Hessian-based pruning

- **Idea.** Express the training risk using the Taylor approximation
  - Suppose that pruning changes the weight $\mathbf{w} \to \mathbf{w} + \mathbf{u}$.
    - Example. Removing i-th weight makes $\mathbf{u} = -w_i \mathbf{e}_i$
  - Then, we can write:

$$\hat{L}(\mathbf{w} + \mathbf{u}) \approx \hat{L}(\mathbf{w}) + \mathbf{g}^\top \mathbf{u} + \frac{\mathbf{u}^\top \mathbf{H} \mathbf{u}}{2}$$

- $\mathbf{g}$: First-order derivative (gradient), evaluated at $\mathbf{w}$
- $\mathbf{H}$: Second-order derivative (Hessian), evaluated at $\mathbf{w}$

Hessian-based pruning

- Now, assume that the weight $\mathbf{w}$ is well-trained.
  - Then, the gradient is near-zero, making:

$$\hat{L}(\mathbf{w}+\mathbf{u}) \approx \hat{L}(\mathbf{w}) + \frac{\mathbf{u}^\top \mathbf{H}\mathbf{u}}{2}$$

  - As the first term on RHS is independent of mask, the mask optimization can be approximated by:

$$\min_{\mathbf{u}} \frac{\mathbf{u}^\top \mathbf{H}\mathbf{u}}{2}$$

with appropriate constraints on $\mathbf{u}$.

Optimal Brain Damage

- LeCun et al. (1989) simplifies this as follows:
  - Suppose that we remove only one weight
  - Then, removing i-th layer makes $\mathbf{u} = -w_i\mathbf{e}_i$, and thus

$$\frac{\mathbf{u}^\top \mathbf{H}\mathbf{u}}{2} = \frac{|w_i|^2 \mathbf{H}_{ii}}{2}$$

    - Simply compute this score for all i, and remove bottom-k weights
      - Requires some calibration data to compute Hessian

Computational aspects

- **Problem.** Hessians have $(\#\text{weight})^2$ entries
  - 1B-scale model will have 10^{18} entries for Hessian = 4 exabytes

- Fortunately, OBD only need Hessian diagonals, with $(\#\text{weight})$ entries
  - Can be computed in a similar way to backpropagation (Homework. Derive the formula)

LeCun et al., "Optimal brain damage" NeurIPS 1989

Optimal Brain Surgeon

- Hassibi & Stork (1992) considers a slightly involved version:
  - Suppose that we remove only one weight, but can also update other weights to compensate for the removed weight.
  - Then, we are solving:

$$\min_{i} \left\{ \min_{\mathbf{u}_i} \left\{ \frac{\mathbf{u}_i^\top \mathbf{H} \mathbf{u}_i}{2} \right\} \quad \text{subject to} \quad \mathbf{e}_i^\top \mathbf{u}_i + w_i = 0 \right\}$$

  - The Lagrangian form is:

$$L_i = \frac{\mathbf{u}_i^\top \mathbf{H} \mathbf{u}_i}{2} + \lambda (e_i^\top \mathbf{u}_i - w_i)$$

Hassibi & Stork., "Second order derivatives for network pruning: Optimal brain surgeon" NeurIPS 1992

Optimal Brain Surgeon

- For fixed i, the solution and the Lagrangian is:

$$\mathbf{u}_i = -\frac{w_i}{[\mathbf{H}^{-1}]_{ii}} \mathbf{H}^{-1} \cdot \mathbf{e}_i$$

$$L_i = \frac{w_i^2}{2[\mathbf{H}^{-1}]_{ii}}$$

  - We can select one weights with the smallest Lagrangian, make corresponding updates, and repeat…

- **Problem.** This requires computing the inverse Hessian!

  - How can we do this, without requiring extremely large matrix inverse?

Hassibi & Stork., "Second order derivatives for network pruning: Optimal brain surgeon" NeurIPS 1992

Computing the inverse Hessian

- Suppose that we use the squared loss for the risk

$$\hat{L}(\mathbf{w}) = \frac{1}{2N}\sum_{i=1}^{N}(y_i - f(x_i; \mathbf{w}))^2$$

- Then, the loss gradient can be written as:

$$\mathbf{g} = \frac{1}{N}\sum_{i=1}^{N}\underbrace{(f(x_i; \mathbf{w}) - y_i)}_{\text{Samplewise Error}}\underbrace{\frac{\partial f}{\partial \mathbf{w}}(x_i; \mathbf{w})}_{\text{Samplewise Gradient}}$$

Hassibi & Stork., "Second order derivatives for network pruning: Optimal brain surgeon" NeurIPS 1992

Computing the inverse Hessian

- The Hessian is:

$$\mathbf{H} = \frac{1}{N}\sum_{i=1}^{N} \frac{\partial f}{\partial \mathbf{w}}(x_i; \mathbf{w}) \frac{\partial f}{\partial \mathbf{w}}^\top(x_i; \mathbf{w}) - \frac{1}{N}\sum_{i=1}^{N} (f(x_i; \mathbf{w}) - y_i) \frac{\partial^2 f}{\partial \mathbf{w}^2}(x_i; \mathbf{w})$$

  - If our model is good enough, we can approximate:

$$\mathbf{H} \approx \frac{1}{N}\sum_{i=1}^{N} \frac{\partial f}{\partial \mathbf{w}}(x_i; \mathbf{w}) \frac{\partial f}{\partial \mathbf{w}}^\top(x_i; \mathbf{w})$$

$$\triangleq \frac{1}{N}\sum_{i=1}^{N} \mathbf{q}_i \mathbf{q}_i^\top$$

Samplewise Gradient

Hassibi & Stork., "Second order derivatives for network pruning: Optimal brain surgeon" NeurIPS 1992

Computing the inverse Hessian

- This gives us a recursive formula for computing the Hessian

$$\mathbf{H}_m = \mathbf{H}_{m-1} + \frac{1}{N}\mathbf{q}_m\mathbf{q}_m^\top$$

- Combine this with the matrix inversion formula:

$$(\mathbf{A} + \mathbf{BCD})^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1}\mathbf{B}(\mathbf{C}^{-1} + \mathbf{DA}^{-1}\mathbf{B})^{-1}\mathbf{DA}^{-1}$$

  - Then, we get a recursive formula for computing inverse Hessian

$$\mathbf{H}_m^{-1} = \mathbf{H}_{m-1}^{-1} - \frac{\mathbf{H}_{m-1}^{-1}\mathbf{q}_m\mathbf{q}_m^\top\mathbf{H}_{m-1}^{-1}}{N + \mathbf{q}_m^\top\mathbf{H}_{m-1}^{-1}\mathbf{q}_m}$$

Hassibi & Stork., “Second order derivatives for network pruning: Optimal brain surgeon” NeurIPS 1992

Other techniques

- Still, Hessian is too large to compute & hold on RAM for large models
- **Solution.**
  - Compute Hessian layer-by-layer (Dong et al., 2017; Layerwise OBS)
    - https://arxiv.org/abs/1705.07565
  - Recent works on LLM develop more involved techniques:
    - Will be discussed in future lectures; https://arxiv.org/abs/2301.00774

Dong et al., “Learning to Prune Deep Neural Networks via Layer-wise Optimal Brain Surgeon” NeurIPS 2017

Gradient-based pruning

- In many cases, we cannot simply assume that gradient = 0
  - Pruning at initialization
    - e.g., SNIP (Lee et al., 2019)
  - Pruning pre-trained models before fine-tuning
    - e.g., Movement Pruning (Sanh et al., 2020)
  - Pruning underfitting models
    - e.g., large language models

Lee et al., "SNIP: Single-shot Network Pruning based on Connection Sensitivity," ICLR 2019
Sanh et al.., "Movement Pruning: Adaptive Sparsity by Fine-Tuning" NeurIPS 2020

Gradient-based pruning

- **Idea.** Use the first-order approximation:

$$\hat{L}(\mathbf{w}+\mathbf{u}) \approx \hat{L}(\mathbf{w}) + \mathbf{g}^\top \mathbf{u}$$

  - Choose i weights with the smallest values of the gradient score:

$$-w_i \mathbf{g}_i$$

    - In fact, taking an absolute value is a good idea (**why?**)

$$|w_i \mathbf{g}_i|$$

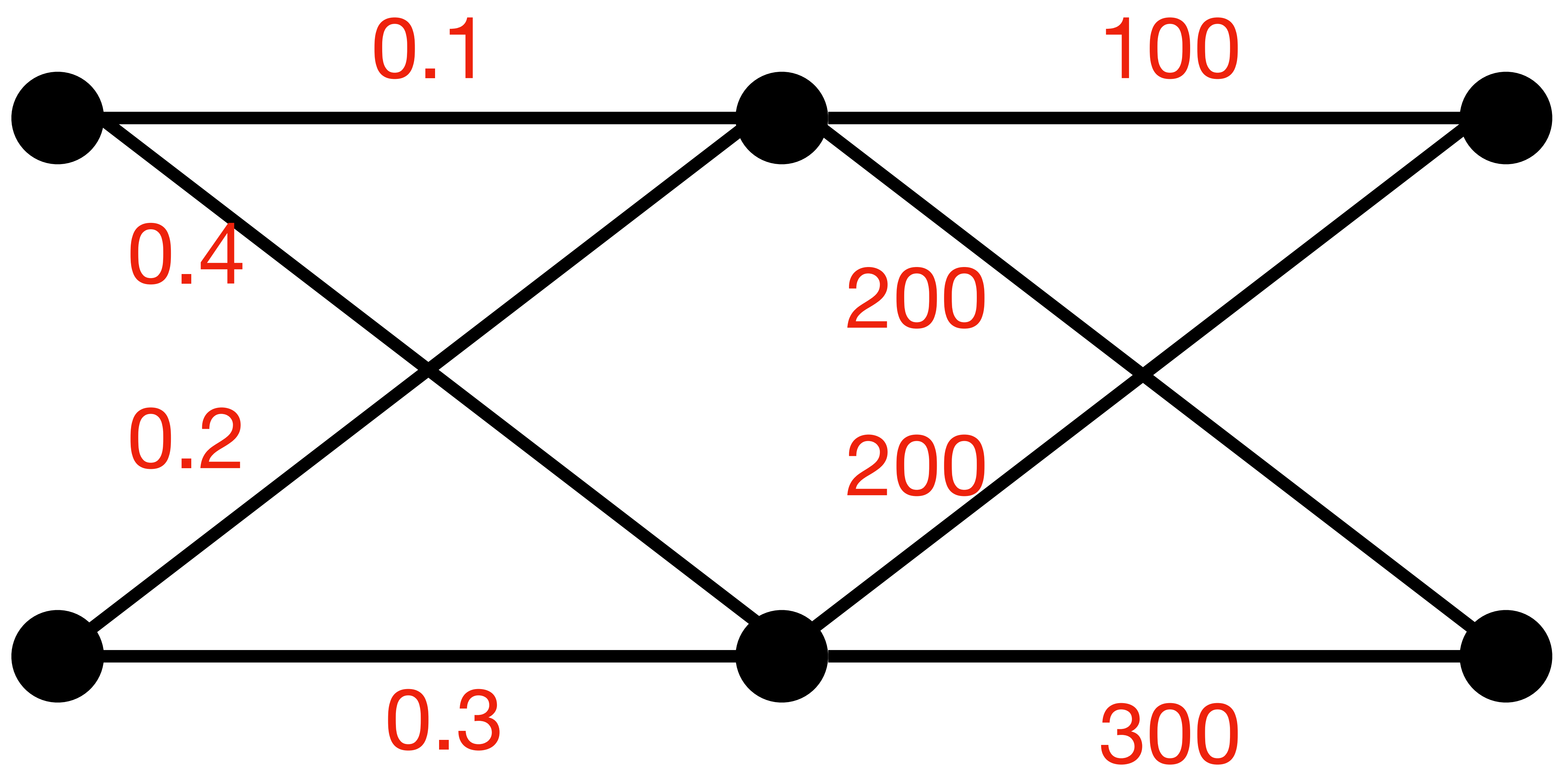
Magnitude-based pruning

- Suppose that we cannot compute Hessian
  - Too much memory & computation needed
  - No calibration data

- **Idea.** Blindly assume that the Hessian diagonal $\mathbf{H}_{ii}$ is identical for all weights.
  - i.e., use the saliency score w_i^2
  - i.e., remove weights with bottom-k weight magnitudes $|w_i|$

Magnitude-based pruning

- **Problem.** Prone to layer collapse, on global pruning
    - Suppose that we have two layer MLP, with:



Magnitude-based pruning

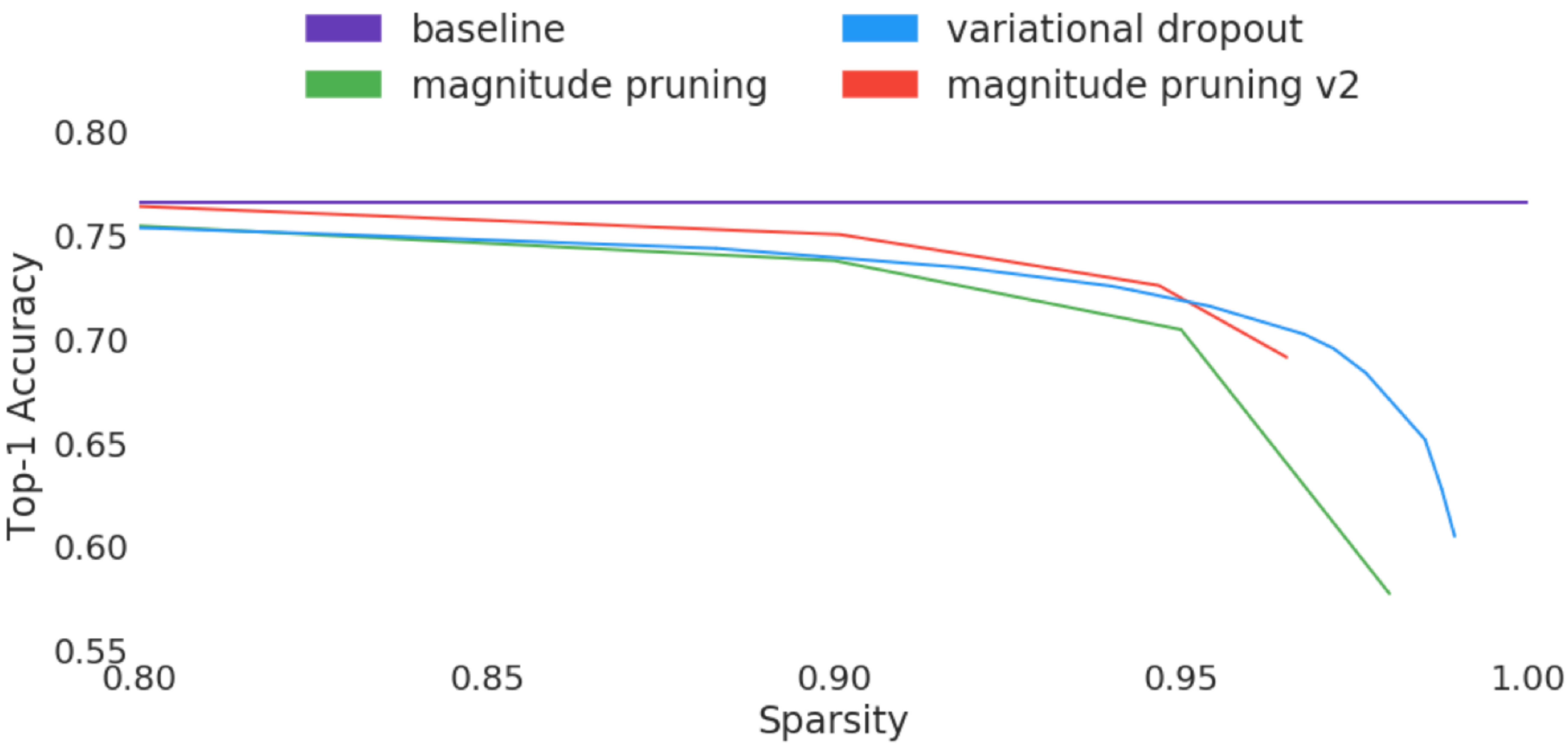
- **Solution.**
  - Layerwise heuristics (Gale et al., 2019)
  - Score scaling (Lee et al., 2021)

> To understand the limits of the magnitude pruning heuristic, we modify our ResNet-50 training setup to leave the first convolutional layer fully dense, and only prune the final fully-connected layer to 80% sparsity. This heuristic is reasonable for ResNet-50, as the first layer makes up a small fraction of the total parameters in the model and the final layer makes up only .03% of the total FLOPs. While tuning

Gale et al., "The state of sparsity in deep neural networks," arXiv 2019
Lee et al., "Layer-adaptive Sparsity for the Magnitude-based Pruning," ICLR 2021

Magnitude-based pruning

- With gradual pruning & good HP, magnitude-based pruning is good enough.
  - With limited retraining, Hessian-based methods are better

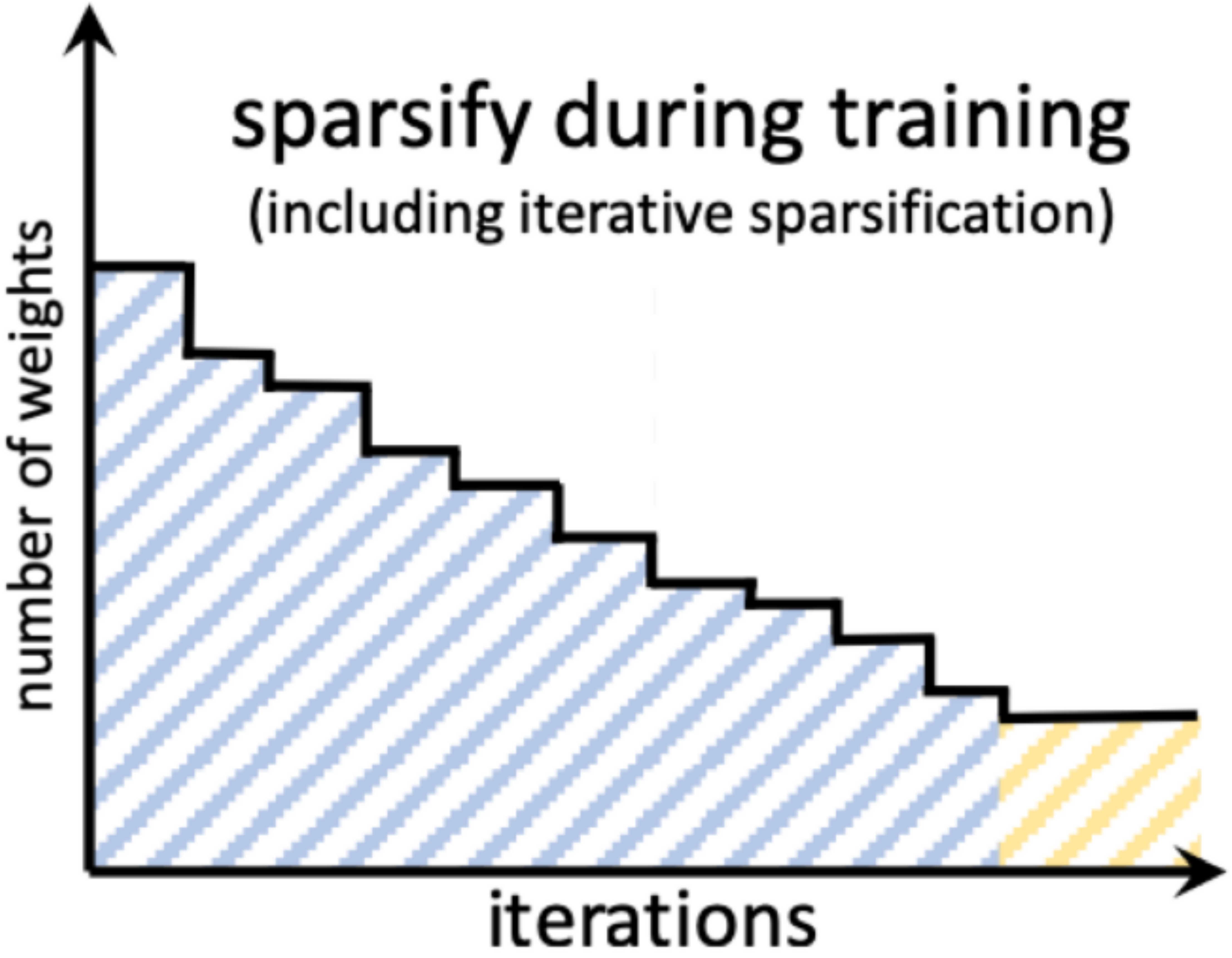


Gale et al., "The state of sparsity in deep neural networks," arXiv 2019

Schedule

Pruning schedules

- There are many different sparsity schedules, with different purposes.
- **Gradual Pruning.**
  - Best in terms of the accuracy vs. inference cost tradeoff
  - Requires lengthy training (2x — 10x of the original training)
  - Needs joint tuning of learning rate and sparsity schedules

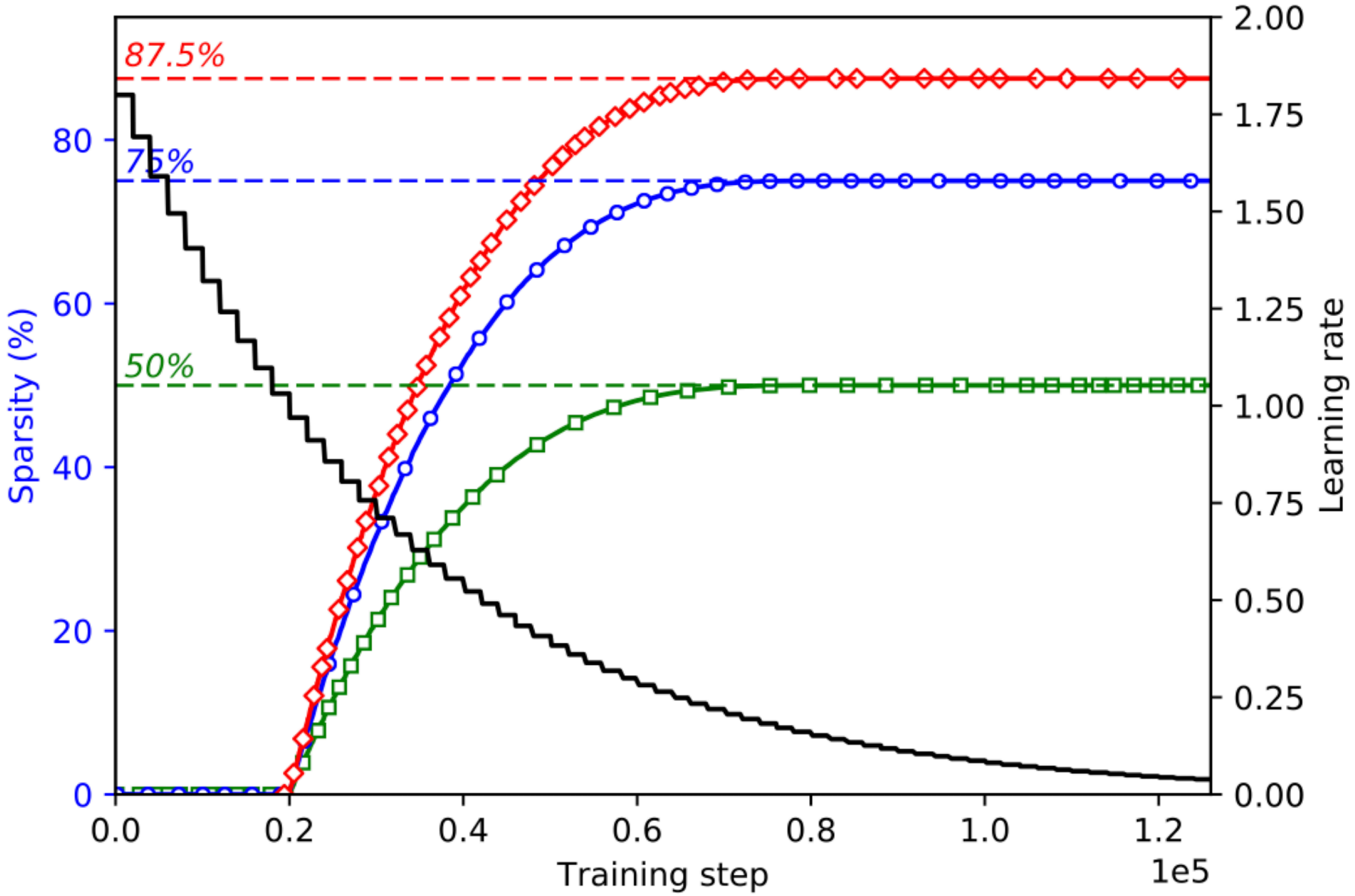


Gale et al., "The state of sparsity in deep neural networks," arXiv 2019

Pruning schedules

- e.g., cubic schedule (Zhu & Gupta, 2017; Google default)

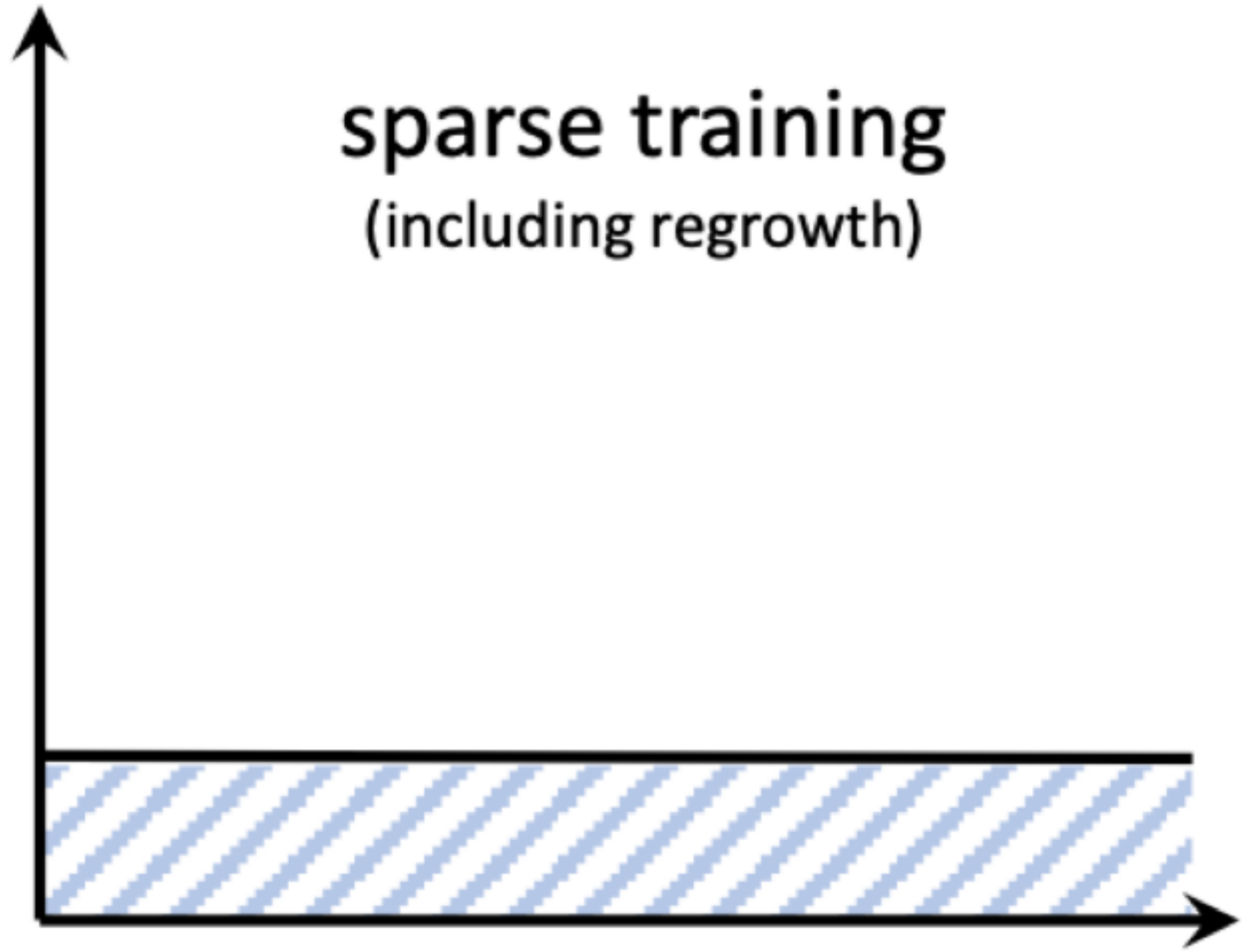
$$s_t = s_f + (s_i - s_f)\left(1 - \frac{t - t_0}{n\Delta t}\right)^3 \quad \text{for} \quad t \in \{t_0,\ t_0 + \Delta t,\ \ldots,\ t_0 + n\Delta t\}$$



Zhu & Gupta., "To prune, or not to prune: exploring the efficacy of pruning for model compression," arXiv 2017

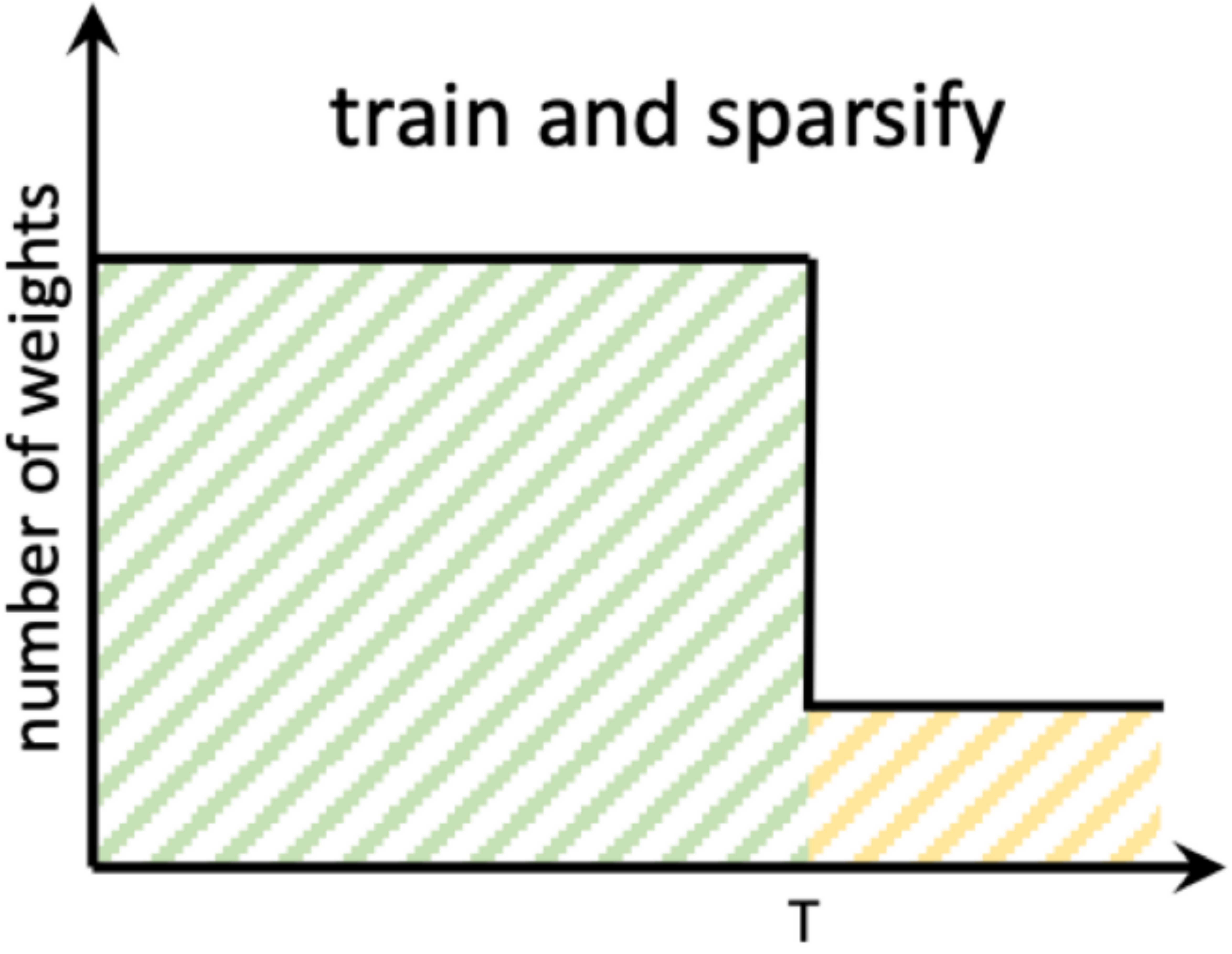
Pruning schedules

- **Sparse Training (Pruning at init.)**
  - Less GPU memory & training cost
    - Can train larger (sparse) models, theoretically
  - Requires lengthy training
  - Very unstable performance
    - Usually requires small sparsity or regrowth



Pruning schedules

- **One-shot Pruning (or Post-Training Sparsity)**
  - Little or no retraining.
    - Suitable for LLM-scale models
  - Bad performance, usually.
  - Can exploit pretrained checkpoints
    - End-user friendly



Further Readings

- Lottery ticket hypothesis
  - https://arxiv.org/abs/1803.03635
- What is the state of neural network pruning?
  - https://arxiv.org/abs/2003.03033

That’s it for today 🙌