EECE695D: Efficient ML Systems

Pruning

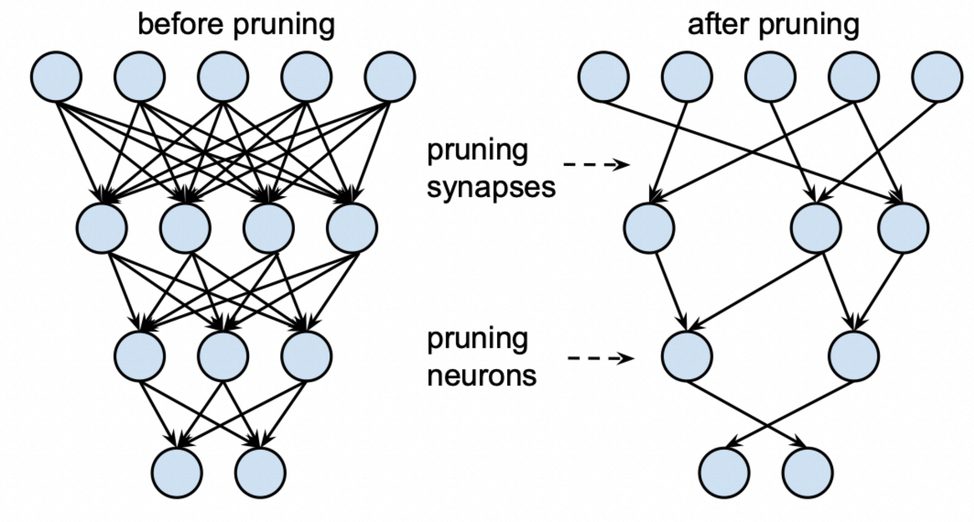
(part 1)

11/1. Virtual lecture via Zoom; will be at Google workshop

TL;DR

Pruning. A model compression technique; zero-ing out the weights of a neural network.
Surprisingly, we can add many zeros without any performance degradation.

$$\begin{bmatrix} a_1 & a_2 & a_3 & a_4 \\ a_5 & a_6 & a_7 & a_8 \\ a_9 & a_{10} & a_{11} & a_{12} \\ a_{13} & a_{14} & a_{15} & a_{16} \end{bmatrix} \longrightarrow \begin{bmatrix} 0 & 0 & \tilde{a}_3 & \tilde{a}_4 \\ \tilde{a}_5 & 0 & \tilde{a}_7 & 0 \\ \tilde{a}_9 & 0 & 0 & \tilde{a}_{12} \\ \tilde{a}_{13} & 0 & \tilde{a}_{15} & \tilde{a}_{16} \end{bmatrix}$$



Note. The surviving weights are often adapted, e.g., via retraining or by solving other optimization problems.

Note. We could select zeros to have specific patterns (e.g., remove all weights from a 3x3 convolution filter).
Such methods are called “structured pruning.”

Note. This also happens in human brain 😢

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

TL;DR

Why? The hope is that we can reduce both memory and compute associated with zero-ed out weights.

$$\begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix}$$

32bits x 4 = 128bits

$$\begin{bmatrix} a_1 & 0 \\ 0 & a_4 \end{bmatrix}$$

32bits x 2 + α = 64bits + α

$$\begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix} \begin{bmatrix} b_1 & b_2 \\ b_3 & b_4 \end{bmatrix} = \begin{bmatrix} a_1b_1 + a_2b_3 & a_1b_2 + a_2b_4 \\ a_3b_1 + a_4b_3 & a_3b_1 + a_4b_4 \end{bmatrix}$$

8 Multiplications, 4 Additions

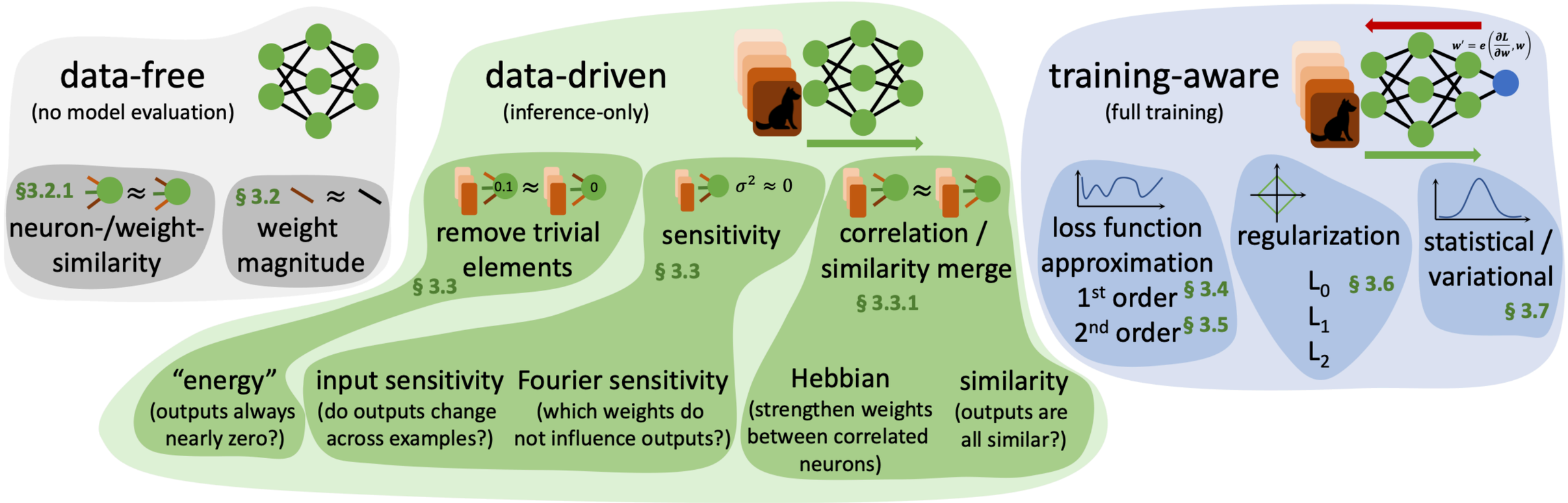
$$\begin{bmatrix} a_1 & 0 \\ 0 & a_4 \end{bmatrix} \begin{bmatrix} b_1 & b_2 \\ b_3 & b_4 \end{bmatrix} = \begin{bmatrix} a_1b_1+0 & a_1b_2+0 \\ 0+a_4b_3 & 0+a_4b_4 \end{bmatrix}$$

4 Multiplications, 0 Additions

TL;DR

Core Question #1. How should we select the weights to prune?

Popular methods are: - Magnitude-based pruning
- Loss-based pruning
- Training-based pruning

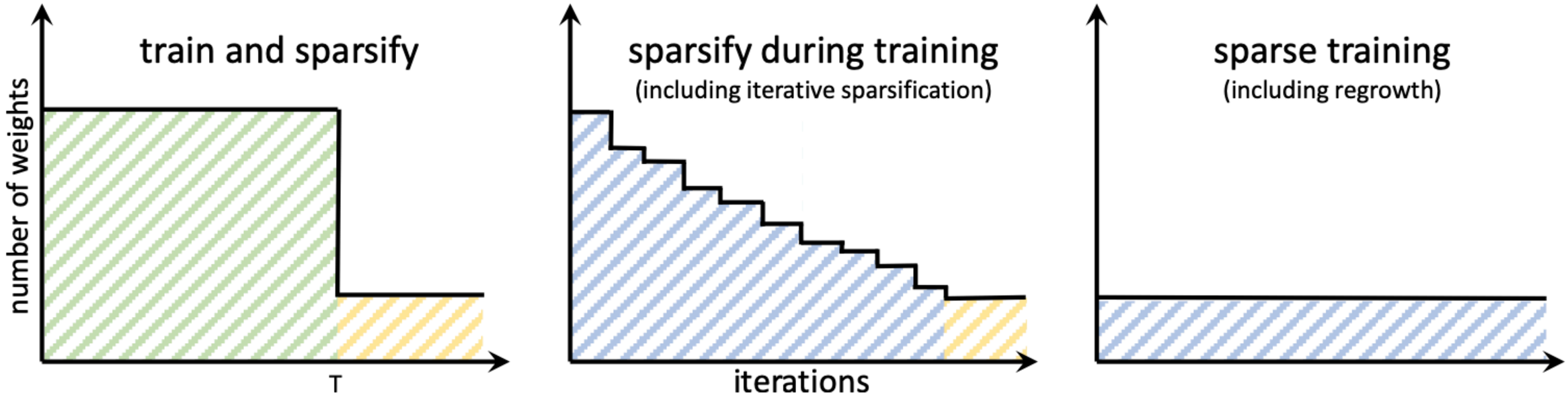


Hoefler et al., "Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks," JMLR 2021

TL;DR

Core Question #2. When should we prune the model, and by how much?
(and should we grow them again?)

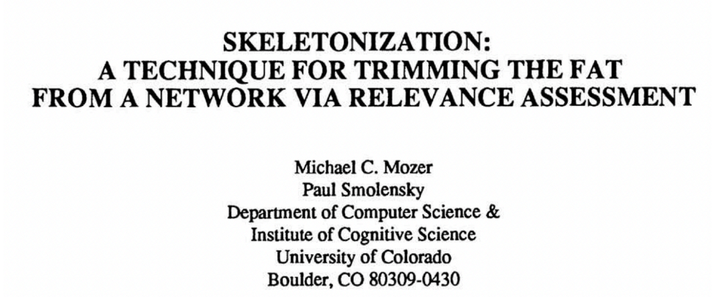
(Spoiler: These matter much for simpler pruning methods)



Hoefler et al., "Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks," JMLR 2021

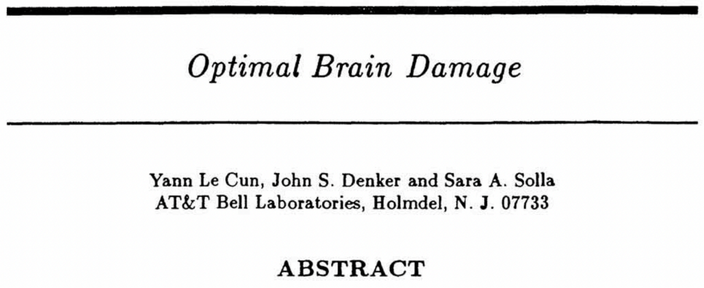
Early works — Loss-based pruning

Beginning. Dates back to late 1980s (NeurIPS 1988 & 1989)



SKELETONIZATION:
A TECHNIQUE FOR TRIMMING THE FAT
FROM A NETWORK VIA RELEVANCE ASSESSMENT

Michael C. Mozer
Paul Smolensky
Department of Computer Science &
Institute of Cognitive Science
University of Colorado
Boulder, CO 80309-0430



Optimal Brain Damage

Yann Le Cun, John S. Denker and Sara A. Solla
AT&T Bell Laboratories, Holmdel, N. J. 07733

ABSTRACT

Motivation. Back then, people believed that pruning will give you three benefits:

(1) *Generalization:* Small number of parameters -> Better generalization!

(2) *Speed-up:* Less compute -> Faster speed!

(3) *Interpretability:* Less decision rules -> Better understanding!

Nowadays, not many people seriously believe in (1) & (3).

Question. What is the best way to select the weights to prune?
Here, the focus is on the tradeoff: #params vs. test accuracy

Approach. Prune weights s.t. the training risk is minimized right after removing them (ignoring retraining).

In other words, if we use $\tilde{\theta}$ to denote the pruned version of θ, we are solving

$$\min_{\tilde{\theta} \in \pi_k(\theta)} L(\tilde{\theta})$$

where $L(\cdot)$ denotes the training loss and $\pi_k(\theta)$ denotes the set

$$\pi_k(\theta) = \left\{ \tilde{\theta} \mid \tilde{\theta} = M \odot \theta, \quad M \in \{0,1\}^d, \quad \|M\|_0 \leq k \right\}.$$

Problem. Measuring $L(\tilde{\theta})$ for every possible $\tilde{\theta} \in \pi_k(\theta)$ is too compute-intensive.....

LeCun et al., "Optimal Brain Damage," NeurIPS 1989

Idea. Use the Taylor approximation:

$$L(\tilde{\theta}) \approx L(\theta) + (\tilde{\theta} - \theta)^\top G_\theta + \frac{1}{2}(\tilde{\theta} - \theta)^\top H_\theta (\tilde{\theta} - \theta)$$

$L(\theta)$: Does not really depend on our pruning decision (so we ignore this term).

$(\tilde{\theta} - \theta)^\top G_\theta$: Should be almost zero if we optimized θ well (gradient will be small enough).

Now, the problem becomes

$$\min_{\tilde{\theta} \in \pi_k(\theta)} (\tilde{\theta} - \theta)^\top H_\theta (\tilde{\theta} - \theta)$$

Problem. Solving the quadratic problem is compute-intensive!

- Hessian is a $d \times d$ matrix (has 10^{18} elements for a neural net with 10^9 parameters).
- The search space $\pi_k(\theta)$ is discrete.

Trick. Approximate the Hessian matrix with its diagonal $\mathbf{diag}(h_{11}, h_{22}, \ldots, h_{dd})$ to get

$$\min_{\tilde{\theta} \in \pi_k(\theta)} \sum_{i=1}^{d} h_{ii}(\tilde{\theta}_i - \theta_i)^2$$

Fortunately, the Hessian diagonal is only d-dimensional and can be computed using a backprop.

Solution. Remove all weights, except for the top-k elements in terms of $h_{ii}\theta_i^2$.

Note. Hassibi & Stork (1998) uses a bit more careful analysis to give a method called "Optimal Brain Surgeon (OBS)," which requires computing Hessian inverse—known to perform better but requires much more compute.

Later, Dong et al. (2017) proposes Layerwise-OBS (L-OBS); performing the OBS for each layer instead of whole model.

Recent works aim to approximate Hessian with a Fisher matrix of the model, which can be estimated more efficiently, e.g., Fisher Pruning (Theis et al., 2018) or WoodFisher (Singh and Alistarh, 2020).

Hassibi and Stork., "Second order derivatives for network pruning: Optimal brain surgeon," NeurIPS 1998
Dong et al., "Learning to prune deep neural networks via layerwise optimal brain surgeon," NeurIPS 2017
Theis et al., "Faster gaze prediction with dense networks and Fisher pruning," arXiv 2018
Singh and Alistarh, "WoodFisher: Efficient second-order approximation for neural network compression," NeurIPS 2020.

Training-based pruning

Some methods do not explicitly zero-out the weights.
Rather, they modify the loss function / gradient update to encourage having many zeros.

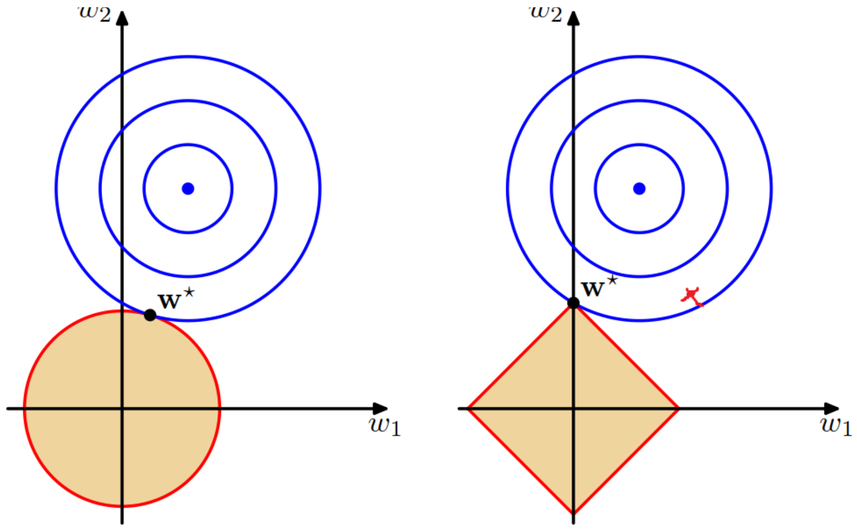
L_0 **Regularization.** Louizos et al. (2018) aims to directly solve the following the minimization objective

$$\min_{\theta} \quad L_{\text{CE}}(\theta) + \lambda \cdot \|\theta\|_0$$

i.e., regularizes training with a L_0 penalty on the parameters.

Problem. The regularization loss $\|\theta\|_0$ is difficult to minimize via SGD—not continuous!
A common trick is to relax this into L_1 constraint, a.k.a. LASSO

But this time, people started thinking differently.....



Louizos et al., "Learning sparse neural networks through L0 regularization," ICLR 2018

Idea. We relax the discrete objective using a probabilistic relaxation.
The previous objective can be rewritten as:

$$L_{\text{CE}}(M \odot \theta) + \lambda \cdot \|M\|_0 \qquad \cdots \qquad M \in \{0,1\}^d$$

Now relax the discrete of M into a random vector that is generated by:

$$M = \min\{1, \max\{0, Z\}\}, \qquad Z_i \sim q(\,\cdot \mid \phi_i).$$

In other words, M is a hard-sigmoided version of another random variable Z parameterized by some ϕ_i.

Then, we can relax the minimization objective as:

$$\mathbb{E}[L_{\text{CE}}(M \odot \theta)] + \lambda \cdot \sum_{i=1}^{d} \Pr[Z_i \leq 0]$$

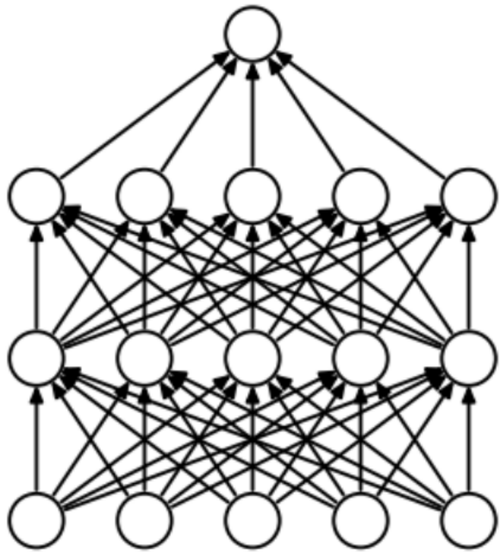
Then, we can use Monte Carlo approximation to update θ, ϕ simultaneously!
(Note: Authors use "hard concrete" distribution as $q(\,\cdot \mid \phi_i)$)
(Note: It is difficult to pre-specify the number of zeros; can only change λ)

Louizos et al., "Learning sparse neural networks through L0 regularization," ICLR 2018

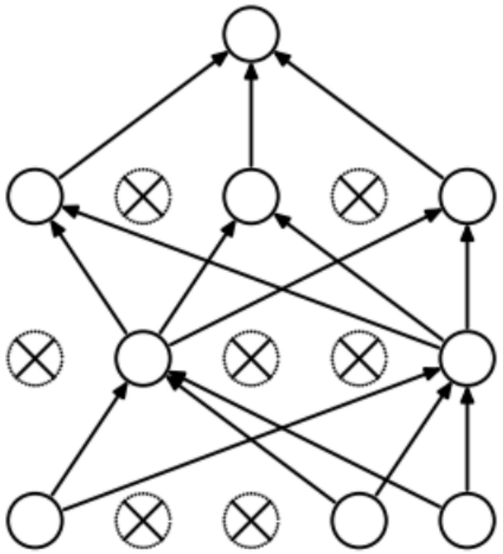
Another similar approach is using variational dropout.

Dropout. A trick to enhance the generalization of a neural network, by intermittently zero-ing out neurons.
(this can be applied to weights instead of neurons, which will be our focus)

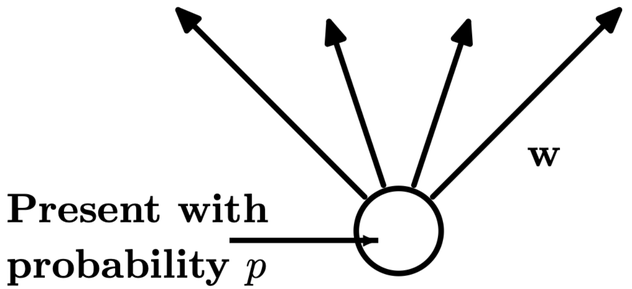
At each iteration of the training phase, we remove each neuron with probability $1 - p$.
During the test phase, we downscale each weight $\mathbf{w}$ to $p\mathbf{w}$



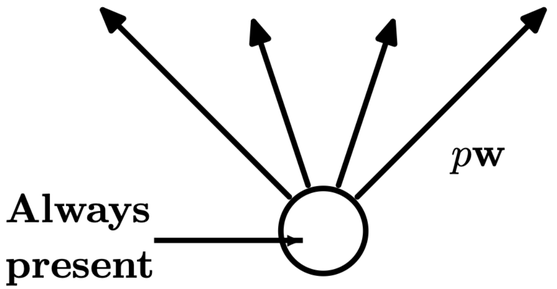
(a) Standard Neural Net



(b) After applying dropout.



(a) At training time



(b) At test time

Srivastava et al., "Dropout: A simple way to prevent neural networks from overfitting," ICLR 2018

Gaussian. Gaussian dropout slightly changes this procedure to do

$$\tilde{\theta} = M \odot \theta \qquad M_i \sim \mathcal{N}(1, \alpha)$$

This is equivalent to drawing each weight from $\tilde{\theta}_i \sim \mathcal{N}(\theta_i, \alpha\theta_i^2)$

Then, training with Gaussian dropout is similar to maximum likelihood estimation—
we fit the input-conditioned output distribution to the training data.

Variational. Make this procedure Bayesian, by performing variational inference of weights with
(1) log-uniform prior distribution of weights ($p(\theta_i) \propto 1/|\theta_i|$)
(2) Gaussian posterior

This procedure can be used to give rise to sparse networks!

Requires many elaboration; interested students are referred to Molchanov et al. (2017)

Other works. Soft threshold weight reparameterization (Kusupati et al., 2020)
Controlled sparsity (Gallego-Posada et al., 2022)

Molchanov et al., "Variational dropout sparsifies deep neural network" ICML 2017.
Kusupati et al., "Soft threshold weight reparameterization for learnable sparsity," ICML 2020.
Gallego-Posada et al., "Controlled sparsity via constrained optimization ... " NeurIPS 2022.

Magnitude-based pruning

The dumb way of pruning—remove weights with small magnitudes, i.e., $|\theta_i|$.

At the same time, easiest to use:

- no extra compute/memory
- no data needed—just model parameters

$\Rightarrow$ go-to method for large-scale!

Intuition#1. Equivalent to Hessian-based pruning, when Hessian is equal to the identity matrix.

$$\min_{\tilde{\theta}} \quad (\tilde{\theta} - \theta)^\top I_d \, (\tilde{\theta} - \theta)$$

Intuition#2. In other words, minimizes the ℓ_2 difference $\|\theta - \tilde{\theta}\|_2$

(This is a useful property; bounding ℓ_2 difference of parameters can be used to bound the overall model output difference)

Standard form. Proposed by Han et al. (2015), refined by Zhu & Gupta (2017).

Step 1. Train the model (optional: with weight decay).

Step 2. Remove some fraction of weights from the model.
(optional: also reconnect the dead weights, if their weight before being pruned has been bigger than the smallest weight to survive)

Step 3. Retrain the model

Step 4. Go back to step 2.

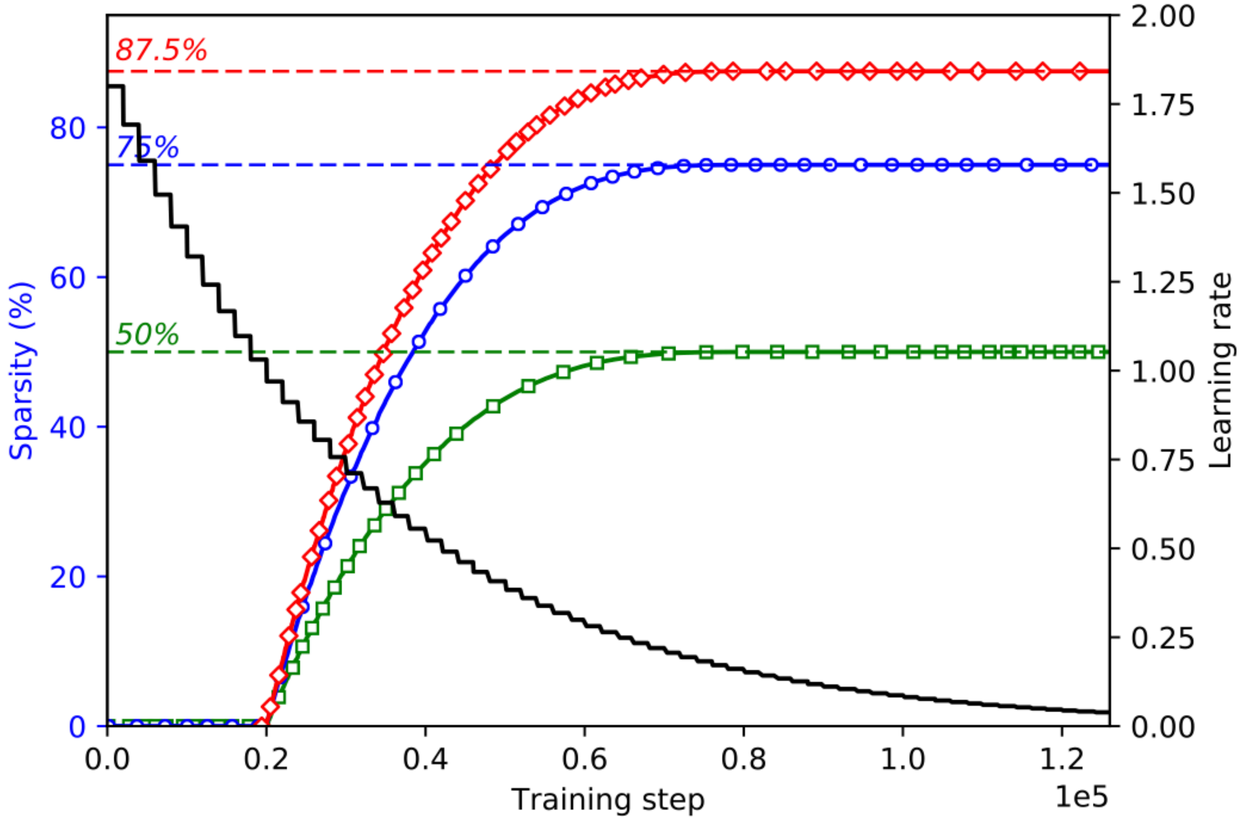
Hyperparams. This procedure has some moving parts:

- Layerwise sparsity (global threshold gives you decent performance)
- Pruning schedule—e.g., #training/retraining steps, fraction to remove at each step (best to have a gradual scheduling)

Han et al., "Learning both weights and connections for efficient neural networks" NeurIPS 2015
Zhu and Gupta, "To prune or not to prune: exploring the efficacy of pruning for model compression," arXiv 2017.

Tuning the hyperparameters of pruning is quite ugly—heuristic layerwise sparsity, “cubic” scheduling

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

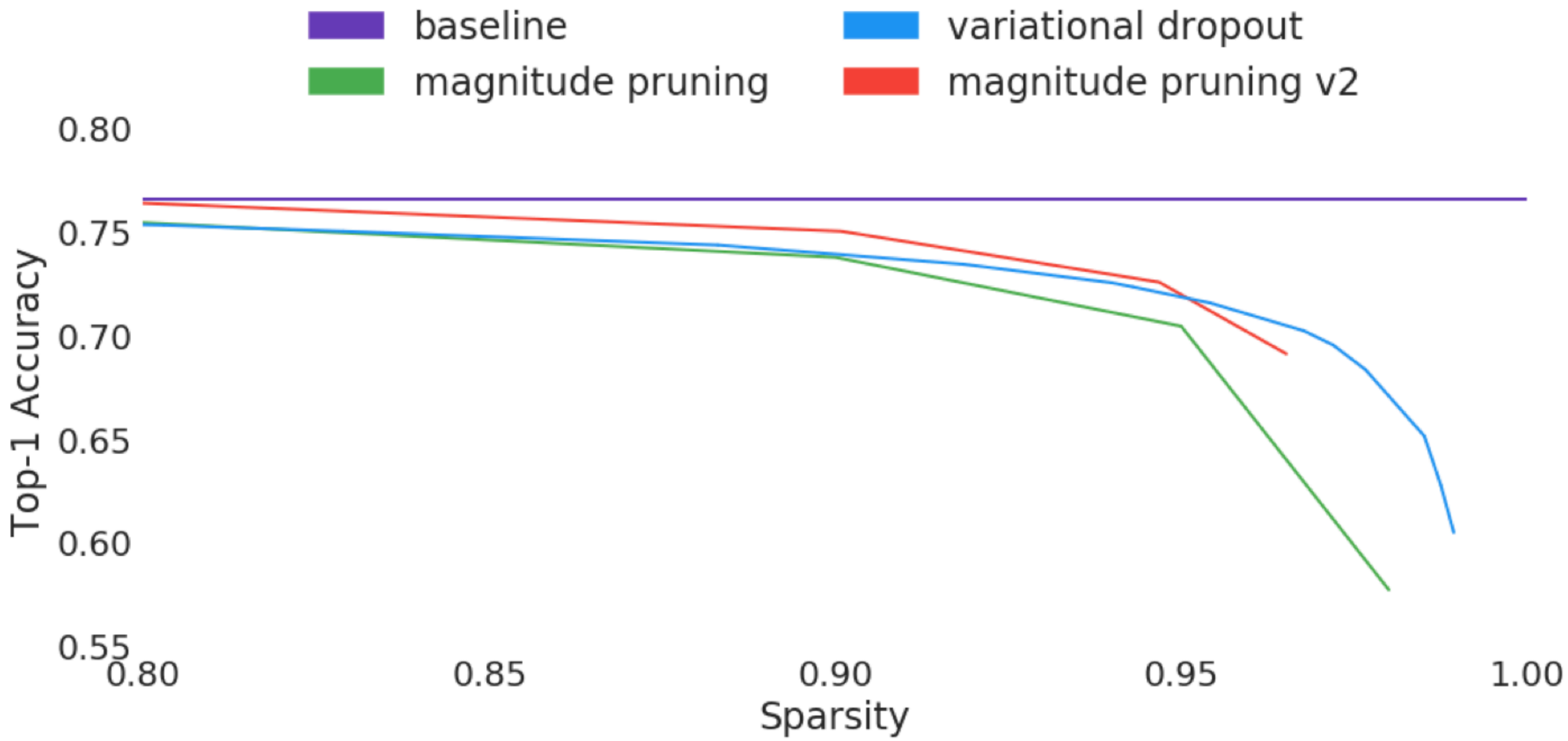


$$s_t = s_f + (s_i - s_f)\left(1 - \frac{t - t_0}{n\Delta t}\right)^3 \quad \text{for} \quad t \in \{t_0,\ t_0 + \Delta t,\ \ldots,\ t_0 + n\Delta t\}$$

Zhu and Gupta, “To prune or not to prune: exploring the efficacy of pruning for model compression,” arXiv 2017.
Gale et al., “The state of sparsity in deep neural networks,” arXiv 2019.

But when tuned right, Gale et al. (2019) shows that magnitude-based pruning achieves SOTA

(Note: Only in terms of #param-accuracy tradeoff!)



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

Comparison of pruning methods

Magnitude-based. Easiest and cheapest to use.
Requires gradual pruning and retraining.
Requires well-tuned hyperparameters (highest total search cost?)

Loss-based. More compute/memory-intensive.
Works relatively well with one-shot pruning (i.e., prune only once).
Some methods are difficult to be used under the presence of normalization layers.

Training-based. Very compute/memory-intensive.
Known to be unstable for large-scale tasks
Works very well, but sparsity level often not preset-able.

Next up. Considerations for real boosts
Sparse storage formats
Structured pruning
Pruning at initialization /
Sparse Training