Review of Aghasi et al. (2020):

Fast Convex Pruning of Deep Neural Networks

Apostolos Psaros

1 Motivation

- Increasing size/flexibility of NNs leads to increasing complexity and in turn to overfitting
- Regularization is used to reduce overfitting. Typical approaches include:
  1. hard/soft constraints on the parameters (e.g., weight decay)
  2. ensembling
  3. data augmentation
  4. pruning
- Goodfellow et al. (2016): "We almost always find that the best fitting model (in the sense of generalization error) is a large model that has been regularized appropriately"
- An additional problem of increasing the size of NNs is the related increase in storage requirements and the number of floating point operations (FLOPs)
- It becomes hard to apply the models on mobile platforms with limited memory and processing units

2 Pruning background

- Motivation: Significantly redundant parametrizations: sparser weight matrices may result in similar or better performance
- Commonly (e.g., Han et al., 2015), pruning consists of 3 stages:
  1. standard training (we learn what neuron connections are important)

2. deleting of unimportant connections (magnitude below some threshold): set corresponding weights to zero
3. retraining the network with the remaining connections (often called fine-tuning)

- Net-Trim stages:

1. standard training
2. for each layer replacing weight matrix by a sparser one that produces approximately the same layer outputs (internal features)
3. fine-tuning (optional)

3 Introduction to Net-Trim

- Consider an FFN with L layers

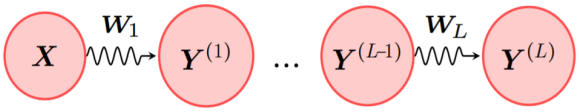


Fig. 1. Fig 2 from Aghasi et al. (2017). In Aghasi et al. (2020) X is used in the layers instead of Y.

- Each weight matrix W_l is of size $N_{l-1} \times N_l$
- layer outputs are given as $X^{(l)} = ReLU(W_l^T X^{(l-1)})$

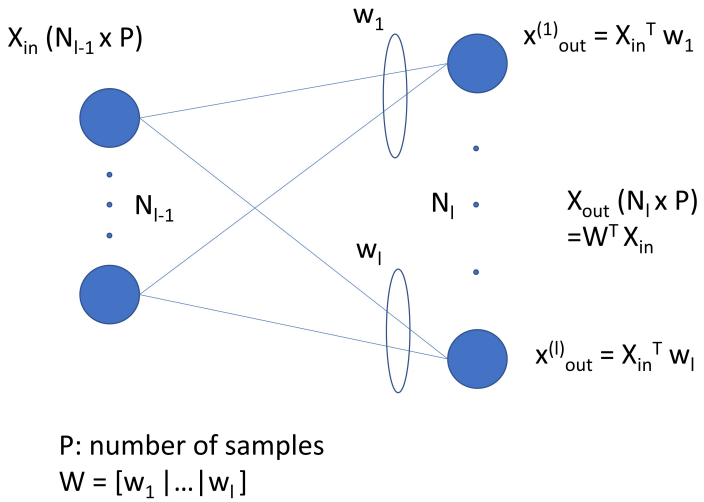


Fig. 2. Layer outputs without activation function for simplicity.

- For each layer search for a sparser weight matrix:
  $\min \|W\|_1 \quad \text{subject to} \quad \left\|ReLU(W^T X_{in} - X_{out})\right\|_F \leq \epsilon$
- Similar output (controllable discrepancy ϵ) with smaller $\|W\|_1$
- Convexify constraint: replace $X^{(l)} = ReLU(W_l^T X^{(l-1)})$ by

$$\left\|(W^T X_{in} - X_{out})_\Omega\right\|_F \leq \epsilon$$
$$(W^T X_{in})_{\Omega^c} \leq 0$$

$$\Omega = \text{supp } \boldsymbol{X}^{out} = \left\{(m,p) : \left[\boldsymbol{X}^{out}\right]_{m,p} > 0\right\}$$

  - if an element of X_{out} is greater than zero then it will be unaffected by ReLU. Thus, we require that $W^T X_{in}$ is close to X_{out}. This both gives the same result and also avoids ReLU
  - if an element of X_{out} is less than zero then it will be zeroed out by ReLU. Thus, we require that $W^T X_{in}$ is also less than zero which both gives the same result and also avoids ReLU

- Two frameworks: Parallel and Cascade

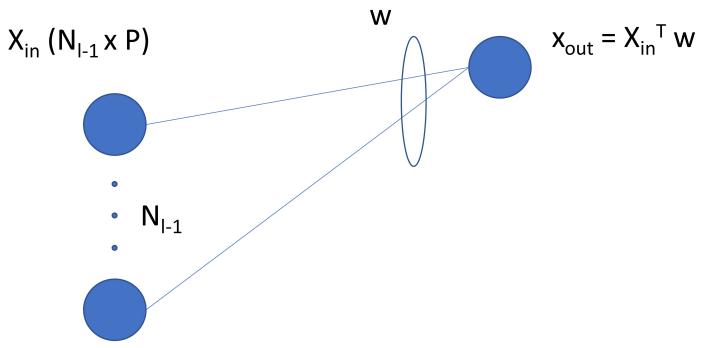
  1. Parallel: Each layer is retrained independently based on the inputs-outputs from the original training
     - We can prove that the output discrepancy for each layer is bounded

$$\left\|\hat{\boldsymbol{X}}^{(\ell)} - \boldsymbol{X}^{(\ell)}\right\|_F \leq \sum_{j=1}^{\ell} \epsilon_j, \quad \ell = 1, \ldots, L.$$

     - Recall that after retraining $\hat{X}$ (not X) will be fed to the next layer so we do need the above bound
     - Retraining of layers can be done in parallel
  2. Cascade: Incorporates the outcome of the previously pruned layers to retrain the next layers
     - Similar bound can be proved
     - Sparser networks at the expense of increased computational cost

4 Connections with sparse representations/compressed sensing

- Consider an unknown vector $w_{n\times 1}$
- We observe linear measurements $y_{m\times 1} = X_{m\times n} w_{n\times 1} + \epsilon$
- How can we approximate w?
- X is usually called sampling matrix
  - Often the columns of X correspond to basis vectors sampled at random locations (rows)
  - In the over-determined case ($m > n$) we use the LSE given as $\hat{w} = (X^T X)^{-1} X^T y$
  - Gives the exact solution if there exists one or an approximation if no exact solution exists
  - In the under-determined case ($m < n$) X does not have full column rank and thus $X^T X$ does not exist. This is the compressed sensing problem
  - How can we obtain a unique solution?
  - If w is sparse enough then we can obtain it by solving $\min \|w\|_1 \quad \text{subject to} \quad \|Xw - y\|_2 \leq \epsilon$
  - We are interested in the true w and sparsity is used as a means for getting it
  - Many theoretical results in the 2000s
- Back to Net-Trim: Let's isolate an output neuron of a NN layer



- The equation $x_{out} = X_{in}^T w$ is of the form of $y = Xw$

- The sampling matrix is X_{in}^T: its columns correspond to different input neurons and its rows to different samples
- Can be over/under-determined system
- In order to sparsify w we need to **solve the same minimization problem** as the one solved in compressed sensing
- Sparsifying w means reducing the number of connections from the input neurons to a given output neuron
- We first reduced the original convex problem with non-convex constraints to a convex one with convex constraints (got rid of ReLU) and then to a series of convex problems (one for each output neuron)
- Easier to do theory: we can use the rich compressed sensing theory

5 Brief review of compressed sensing (underdetermined case)

- Underdetermined system: $y_{m\times 1} = X_{m\times n} w_{n\times 1}$
- If w is s-sparse then we can obtain it exactly by solving
  $\min \|w\|_0 \quad \text{subject to} \quad Xw = y$
- How do we know that the solution is unique and exact?
- w needs to be structured (e.g., sparsity $s = \|w\|_0 << n$)
- X needs to be "nice" (e.g., small coherence)
- Thus, most related theorems follow these lines: Suppose that w is an unknown s-sparse vector and we have observations $y_{m\times 1} = X_{m\times n} w_{n\times 1}$. If s is smaller than a number that has to do with some property of X, then by solving the ℓ_0 minimization problem you can get the exact w
- For example, if $s \leq krank(X)/2$ then the ℓ_0 minimization problem results in the exact solution w (krank(X) is the largest number r such that any subset of r columns of X is linearly independent)
- Because X is a sampling matrix and corresponds to random observations, these theorems often have a randomness component. For example, if X is a matrix filled with Gaussian i.i.d. samples then $krank(X) = m$ almost surely

- Thus, most related theorems take this form: Suppose that w is an unknown s-sparse vector and we have observations $y_{m\times 1} = X_{m\times n} w_{n\times 1}$ where X is a matrix filled with Gaussian i.i.d. samples. If the number of observations m is larger than a number that is a function of the sparsity s, then by solving the ℓ_0 minimization problem you can get the exact w with high probability

- Since solving the ℓ_0 minimization problem is NP-hard we instead solve the convex alternative which is the ℓ_1 minimization problem. We can prove similar theorems for this problem

- The main theorem of the paper has exactly this form but is more general because it considers also the overdetermined case. Thus without even looking at the proof we can still understand the basic ingredients of the theorem

6 Main theorem of the paper

Theorem 4.4 (main sample complexity result). *For the model (2.2), consider a trained neuron obeying* $\boldsymbol{x}^{out} = \texttt{ReLU}(\boldsymbol{X}^{in\top}\boldsymbol{w}_0)$, *where* $\boldsymbol{X}^{in} = [\boldsymbol{x}_1, \dots, \boldsymbol{x}_P] \in \mathbb{R}^{N\times P}$ *and* $\boldsymbol{x}, \boldsymbol{x}_1, \dots, \boldsymbol{x}_P$ *are independent samples of a subgaussian distribution. Assume that an* s*-sparse vector* $\boldsymbol{w}^* \in \mathbb{R}^N$ *is feasible for the convex program (4.7) and* $\hat{\boldsymbol{w}}$ *is any solution to (4.7). Fix* $\beta \geq 1/2$ *and* $t \geq 0$, *and define the virtual input* $\boldsymbol{v} = \boldsymbol{x}1_{\boldsymbol{w}_0^\top \boldsymbol{x}>0}$. *Then with probability exceeding* $1 - e^{-ct}$,

$$\|\hat{\boldsymbol{w}} - \boldsymbol{w}^*\| \leq \frac{2\epsilon'}{\|\boldsymbol{v} - \mathbb{E}\boldsymbol{v}\|_{\psi_2}\left(\sqrt{\frac{P}{C_{\beta,v}}} - C\sqrt{s\log\left(\frac{N}{s}\right) + s + 1 + t}\right)^+}. \tag{4.8}$$

The absolute constants c *and* C *are universal and the coefficient* $C_{\beta,v}$ *depends on the statistics of the virtual input via*

$$C_{\beta,v} = (1+\beta)^2\left(\frac{\|\boldsymbol{v} - \mathbb{E}\boldsymbol{v}\|_{\psi_2}^2}{\lambda_{\min}(\mathrm{cov}(\boldsymbol{v}))}\right)^{3+\frac{1}{\beta}}. \tag{4.9}$$

In an underdetermined setting, where the number of samples used to apply the Net-Trim is less than the neuron's input size (i.e., $P < N$), one may use (4.6) to retrain the layer. In this case, upon the existence of a sparse solution, a corollary of Theorem 4.4 reveals that the required number of samples may be much less than the layer's input dimension.

Corollary 4.5. *Consider a similar neuron and setting as in Theorem 4.4. Assume that an* s*-sparse vector* $\boldsymbol{w}^* \in \mathbb{R}^N$ *is capable of generating an identical response to* $\boldsymbol{X}^{in}$ *as* $\boldsymbol{x}^{out}$. *If*

$$P \gtrsim C_{\beta,v}\left(s\log\left(\frac{N}{s}\right) + s + 1 + t\right), \tag{4.10}$$

retraining the neuron via (4.6) accurately identifies $\boldsymbol{w}^*$ *with probability exceeding* $1 - e^{-ct}$.

- What to remember: If the weight matrix of a layer is indeed redundant and there exists a sparser matrix producing the same outputs, then you can solve the ℓ_1 minimization problem and approximately obtain this sparser matrix. This is true even if the number of samples you use is less than the number of output neuron connections
- The original minimization problem **for each layer** is
  $\min \|W\|_1 \quad \text{subject to} \quad \left\|(W^T X_{in} - X_{out})_\Omega\right\|_F \leq \epsilon \quad \text{and} \quad (W^T X_{in})_{\Omega^c} \leq 0$
  but in the theoretical part the problem is divided into a number of smaller minimization problems so that it is easier to provide a theorem. In the paper they call it a neuronwise application of the Net-Trim algorithm
- It is exact for $\epsilon = 0$ and a reasonable approximation for $\epsilon > 0$.
- In the implementation the original problem is tackled numerically

7 Net-Trim implementation

- Basic idea: The problem is of the form $\min f_1(x) \quad \text{subject to} \quad g(x) \in C$
- It is recast as $\min f_1(x) + f_2(g(x))$, where $f_2(g(x)) = 0$ if $g(x) \in C$ else ∞
- And then recast as $\min f_1(x) + f_2(z)$ subject to $z = g(y) \quad \text{and} \quad x = y$
- For such constrained minimization problems we solve for a saddle point of the Lagrangian, i.e.,

$$\begin{aligned} \text{minimize}_{\boldsymbol{x},\boldsymbol{z}} \quad & f_1(\boldsymbol{x}) + f_2(\boldsymbol{z}) \\ \text{subject to} \quad & \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{z} = \boldsymbol{b} \end{aligned}$$

$$\Updownarrow$$

$$\text{maximize}_{\boldsymbol{\lambda}} \quad \min_{\boldsymbol{x},\boldsymbol{z}} \underbrace{f_1(\boldsymbol{x}) + f_2(\boldsymbol{z}) + \langle \boldsymbol{\lambda}, \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{z} - \boldsymbol{b} \rangle}_{=:\mathcal{L}(\boldsymbol{x},\boldsymbol{z},\boldsymbol{\lambda}) \text{ (Lagrangian)}}$$

- The alternating direction method of multipliers (ADMM) solves the above problem by alternating minimizations for x and z and updating the dual variable λ, i.e.,

$$\begin{aligned} \boldsymbol{x}^{t+1} &= \arg\min_{\boldsymbol{x}} \left\{ f_1(\boldsymbol{x}) + \frac{\rho}{2} \left\| \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{z}^t - \boldsymbol{b} + \frac{1}{\rho}\boldsymbol{\lambda}^t \right\|_2^2 \right\} \\ \boldsymbol{z}^{t+1} &= \arg\min_{\boldsymbol{z}} \left\{ f_2(\boldsymbol{z}) + \frac{\rho}{2} \left\| \boldsymbol{A}\boldsymbol{x}^{t+1} + \boldsymbol{B}\boldsymbol{z} - \boldsymbol{b} + \frac{1}{\rho}\boldsymbol{\lambda}^t \right\|_2^2 \right\} \\ \boldsymbol{\lambda}^{t+1} &= \boldsymbol{\lambda}^t + \rho\big(\boldsymbol{A}\boldsymbol{x}^{t+1} + \boldsymbol{B}\boldsymbol{z}^{t+1} - \boldsymbol{b}\big) \end{aligned}$$

- Final algorithm

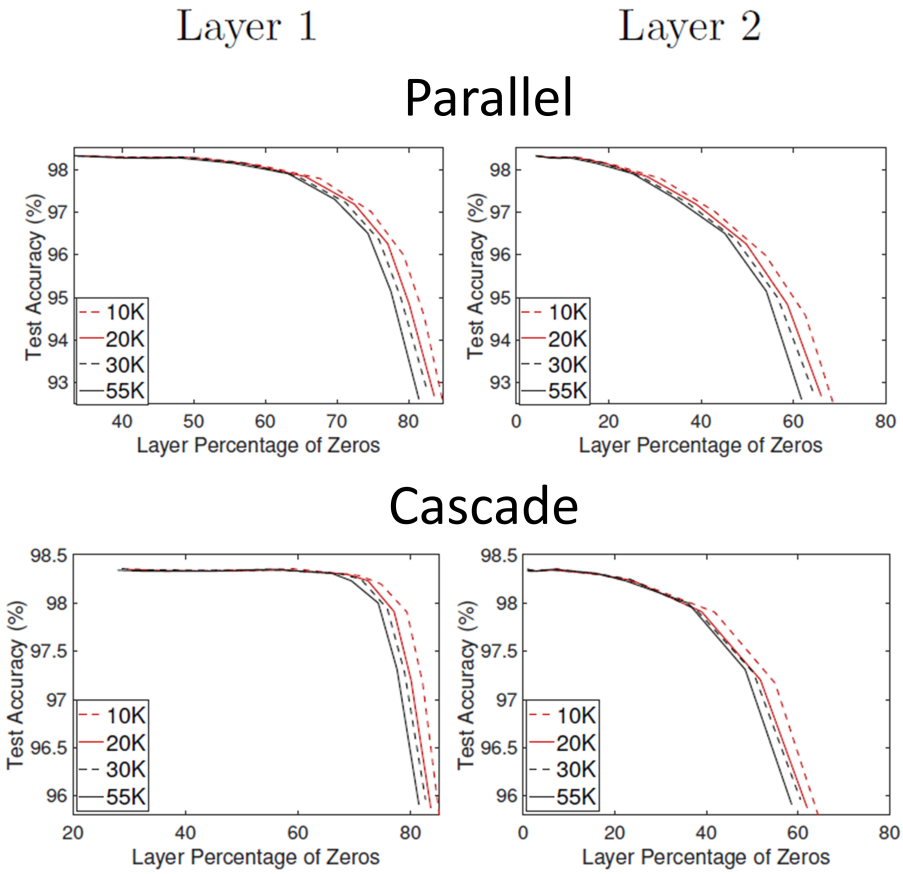
Algorithm 5.1. Implementation of the Net-Trim central program.

input: $\boldsymbol{X}^{in} \in \mathbb{R}^{N \times P}$, $\boldsymbol{X}^{out} \in \mathbb{R}^{M \times P}$, Ω, $\boldsymbol{V}_\Omega$, ϵ, ρ
initialize: $\boldsymbol{U}^{(1)}, \boldsymbol{U}^{(2)}$ and $\boldsymbol{W}^{(3)}$ `% all initializations can be with 0`
$\boldsymbol{C} \leftarrow \boldsymbol{X}^{in}\boldsymbol{X}^{in\top} + \boldsymbol{I}$
while not converged **do**
  $\boldsymbol{Y} \leftarrow \boldsymbol{W}^{(3)\top}\boldsymbol{X}^{in} - \boldsymbol{U}^{(1)}$
  if $\left\|\boldsymbol{Y}_\Omega - \boldsymbol{X}^{out}_\Omega\right\|_F \le \epsilon$ **then** — *Primal updates*
    $\boldsymbol{W}^{(1)}_\Omega \leftarrow \boldsymbol{Y}_\Omega$
  else
    $\boldsymbol{W}^{(1)}_\Omega \leftarrow \boldsymbol{X}^{out}_\Omega + \epsilon \left\|\boldsymbol{Y}_\Omega - \boldsymbol{X}^{out}_\Omega\right\|_F^{-1} \left(\boldsymbol{Y}_\Omega - \boldsymbol{X}^{out}_\Omega\right)$
  end if
  $\boldsymbol{W}^{(1)}_{\Omega^c} \leftarrow \boldsymbol{Y}_{\Omega^c} - (\boldsymbol{Y}_{\Omega^c} - \boldsymbol{V}_{\Omega^c})^+$
  $\boldsymbol{W}^{(2)} \leftarrow S_{1/\rho}(\boldsymbol{W}^{(3)} - \boldsymbol{U}^{(2)})$ `% $S_{1/\rho}$ applies to each element of the matrix`
  $\boldsymbol{W}^{(3)} \leftarrow \boldsymbol{C}^{-1}(\boldsymbol{X}^{in}(\boldsymbol{W}^{(1)} + \boldsymbol{U}^{(1)})^\top + \boldsymbol{W}^{(2)} + \boldsymbol{U}^{(2)})$
  $\boldsymbol{U}^{(1)} \leftarrow \boldsymbol{U}^{(1)} + \boldsymbol{W}^{(1)} - \boldsymbol{W}^{(3)\top}\boldsymbol{X}^{in}$ — *Dual updates*
  $\boldsymbol{U}^{(2)} \leftarrow \boldsymbol{U}^{(2)} + \boldsymbol{W}^{(2)} - \boldsymbol{W}^{(3)}$
end while
return $\boldsymbol{W}^{(3)}$

8 Experiments

1. Comparison between parallel and cascade frameworks

   - FFN $784 \times 300 \times 1000 \times 100 \times 10$ for MNIST
   - In practice original training data can be used for the retraining phase
   - Working with less amount of data is computationally desirable



   - Test accuracy drops faster for increasing sparsity for parallel framework
   - Recall that sparsity is controlled through the allowed discrepancy ϵ
   - However, parallel is distributable and thus can be faster

2. Comparison of Net-Trim with Dropout and ℓ_1 penalty

- LeNet convolutional network

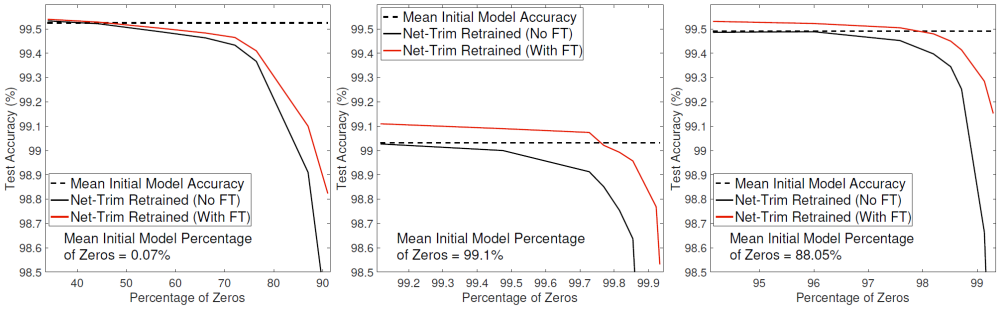


Figure 4. *Mean test accuracy versus mean model sparsity after the application of Net-Trim to the LeNet network initially regularized via ℓ_1 penalty, Dropout, or both (the regularization parameter and Dropout probability are picked from a range of values and the mean accuracy and sparsity are reported);* (a) *a model trained with Dropout only:* $0.3 \leq p \leq 0.8$*;* (b) *a model trained with ℓ_1 penalty only:* $10^{-5} \leq \lambda \leq 5 \times 10^{-3}$*;* (c) *a model trained with Dropout and ℓ_1:* $10^{-5} \leq \lambda \leq 2 \times 10^{-4}$, $0.5 \leq p \leq 0.75$.

- For an initially trained model with Dropout + l_1 penalty we can reduce percentage of zeros from 88% to 98% without loss of accuracy (panel c)
- It seems that with fine-tuning we can even increase accuracy while also pruning the network

3. Comparison of Net-Trim with HPTD (Han et al., 2015) which truncates the small weights across a trained network and performs another round of training on the active weights (fine-tuning)

    - FC (top) and LeNet (bottom) networks

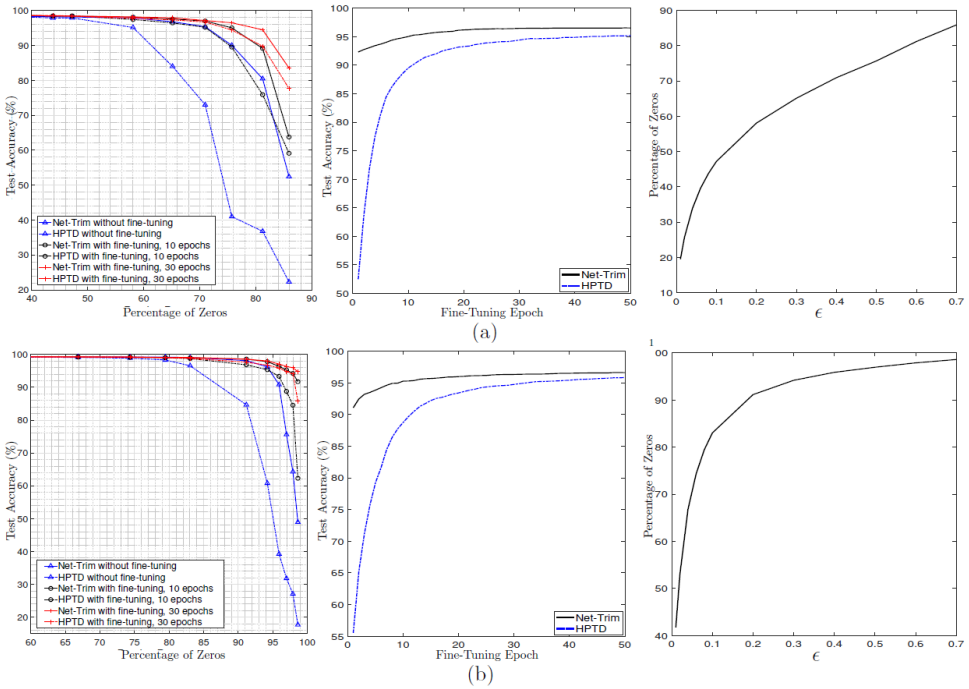


Figure 6. *Comparison of Net-Trim and HPTD in different settings for* (a) *FC model,* (b) *LeNet model; the left panels compare Net-Trim and HPTD test accuracy versus percentage of zeros, without fine-tuning, and with fine-tuning using* 10 *and* 30 *epochs; middle panels show the number of fine-tuning epochs and the acquired accuracy using Net-Trim and HPTD; the right panels indicate the percentage of zeros as a function* ϵ *for Net-Trim.*

- For the Net-Trim we use different values of ϵ to prune the trained networks. To compare the method with the HPTD, after each application of the Net-Trim and counting the number of zeros, the same number of elements are truncated from the initial network to be used for the HPTD implementation. HPTD is followed by a fine-tuning step after the truncation, which is also an optional task for Net-Trim
- Net-Trim outperforms HPTD in these scenarios. The reason is that pruning with HPTD is based on the magnitude of the weights, which in many cases may discard connections to the important features and variables in the network

- Applying the Net-Trim to each layer of the MNIST networks is on the order of 20 seconds, and the fine-tuning step is on the order of 30 seconds. For the CIFAR-10 network, the Net-Trim application to each layer is about 1 minute, and the fine-tuning step takes about 20–30 minutes, depending on the sparsity of the network
- Discarding weights via HPTD does not incur any computational cost while fine-tuning may take more time compared to Net-Trim (see middle panels in figure above)
- Note that fine-tuning may degrade the accuracy especially in low pruning regimes (small percentage of zeros) due to overfitting

References

Aghasi, A., Abdi, A., Nguyen, N., and Romberg, J. (2017). "Net-Trim: Convex Pruning of Deep Neural Networks with Performance Guarantee". *Advances in Neural Information Processing Systems*, pp. 3177–3186.

Aghasi, A., Abdi, A., and Romberg, J. (2020). "Fast Convex Pruning of Deep Neural Networks". *SIAM Journal on Mathematics of Data Science* 2, pp. 158–188.

Goodfellow, I., Bengio, Y., Courville, A., and Bengio, Y. (2016). *Deep Learning*. Vol. 1. MIT press Cambridge.

Han, S., Pool, J., Tran, J., and Dally, W. (2015). "Learning Both Weights and Connections for Efficient Neural Network". *Advances in neural information processing systems* 28, pp. 1135–1143.