Hierarchical Adaptive Lasso: Learning Sparse Neural Networks with Shrinkage via Single Stage Training

Skyler Seto, Martin T. Wells and Wenyu Zhang
Department of Statistics and Data Science
Cornell University
Ithaca, NY
{ss3349, mtw1, wz258}@cornell.edu

Abstract

Deep neural networks achieve state-of-the-art performance in a variety of tasks, however this performance is closely tied to model size. Sparsity is one approach to limiting model size. Modern techniques for inducing sparsity in neural networks are (1) network pruning, a procedure involving iteratively training a model initialized with a previous run’s weights and hard thresholding, (2) training in one-stage with a sparsity inducing penalty (usually based on the Lasso), and (3) training a binary mask jointly with the weights of the network.

In this work, we study different sparsity inducing penalties from the perspective of Bayesian hierarchical models with the goal of designing penalties which perform well without retraining subnetworks in isolation. With this motivation, we present a novel penalty called Hierarchical Adaptive Lasso (HALO) which learns to adaptively sparsify weights of a given network via trainable parameters without learning a mask. When used to train over-parametrized networks, our penalty yields small subnetworks with high accuracy (winning tickets) even when the subnetworks are not trained in isolation. Empirically, on the CIFAR-100 dataset, we find that HALO is able to learn highly sparse network (only 5% of the parameters) with approximately a 2% and 4% gain in performance over state-of-the-art magnitude pruning methods at the same level of sparsity.

1 Introduction

Neural networks have made major advances towards solving problems in image recognition [34], speech recognition [29], natural language understanding [15], and healthcare [16]. Although these networks are powerful, they often contain millions of parameters necessitating considerable storage and memory costs. As a result, these modern architectures are from a financial, environmental, and computational standpoint [1] costly to train and evaluate, from a theoretical standpoint, highly over-parameterized [14] resulting in worse generalization bounds [3], and from a neurological standpoint, dense networks do not accurately represent the sparse activity of neuron activations in the neocortex [27].

Neural network pruning [13] [25] [28] [31] [38] is one technique for reducing the costs associated with deep neural networks. The typical procedure has two training stages (1) training an over-parameterized network and (2) identifying a subnetwork via a pruning criterion (such as weight magnitudes) and fine-tuning the subnetworks using the weights found in the previous iteration. Recent works [21] [40] [78] investigate the second step of the network pruning procedure and identified that the same or higher accuracy can be attained even if the subnetwork is re-initialized to the same weights or to random weights, and that the benefit comes from the subnetwork’s architecture. However,
one issue with network pruning is that it requires two or more stages of training for obtaining the subnetwork and re-training the subnetwork in isolation.

Regularization is another active area of research for achieving sparse neural networks. First studied as a method to prevent overfitting, one of the earliest forms of regularization in neural networks literature is weight decay ($L_2$ regularization). Subsequently, sparsity-inducing regularizers based on the Lasso [62] have been applied to deep neural network training [12, 51, 66]. These approaches typically involve one stage of training with an added penalty, where the sparsity inducing penalties shrink model parameters towards zero. Bayesian masking [33] is an alternative to sparsity inducing penalties have also been studied for pruning deep neural networks [35, 42, 50], but there is no guarantee that a specified sparsity level can be obtained without repeated testing with different combinations of hyperparameter values.

Inspired by these two active areas of research in neural network compression, we highlight connections between sparsity inducing regularizers and network pruning methods. From this comparison, we consider nonconvex penalties and introduce a novel penalty called Hierarchical Adaptive Lasso (HALO), which adaptively learns to sparsify over-parameterized networks through adaptive shrinkage. HALO regularizes model parameters in a hierarchical fashion and aims to shrink each model parameter based on its importance in the model. With the resulting order in the magnitude of the model parameters, further pruning by simple thresholding can be applied to obtain the desired level of sparsity with less drop in accuracy than competing methods. We demonstrate on multiple image recognition tasks and neural network architectures that HALO is able to simultaneously learn highly sparse (95% sparsity) networks with little to no accuracy drop and find that small subnetworks can be learned during training without needing to be trained in isolation or learn a mask.

2 Related Work

2.1 Pruning Methods

Individual weight pruning is a technique for compressing neural networks which first used a criteria based on the Hessian of the loss function [26, 56] to remove weights. Recent work also explored pruning individual weights based on their overall contribution to the loss [37, 68]. Other recent advances such as the work by [24, 25, 50] achieved significant network compression by pruning networks using the magnitude based criteria $|w_j|_1$ where $w_j$ is a weight parameter of the network. The majority of sparse networks learned via pruning performed well, however they are initialized based on weights from the first iteration requiring training until convergence of the full model. Recent work referred to as the lottery ticket hypothesis (LTH) [21, 78] explored the magnitude criteria for compressing network architectures and demonstrated that subnetworks within larger networks are trainable from scratch. Their approach was to first train a randomly initialized over-parameterized network, threshold small weights to zero, then re-initialize the non-zero weights to the same initialization as the over-parameterized network and train only the non-zero weights. This procedure determined that the individual weights are important and not the final weight values from the first stage of training.

Other recent works expanded on the LTH by identifying subnetworks in a randomly initialized network which perform well on a given task without training the weights [55]. Specifically, Ramanujan et al. [55] devised an algorithm for finding randomly initialized subnetworks in larger over-parameterized networks that performed better than trained networks, and Malch et al. [43] proved that for any network of depth $\ell$, a subnetwork could be found in any depth $2\ell$ network which achieves equivalent performance to the depth $\ell$ network. Motivated by the results of LTH and the performance of random subnetworks, we aim to identify small subnetworks during training which achieve competitive accuracy with the over-parameterized model using a modified version of the penalty equivalent of magnitude pruning.

2.2 Regularization Methods

Regularization, like network pruning induces sparsity in deep neural networks using a pre-specified criteria (usually referred to as a penalty). In the regularization setting, we consider the modified loss

$$Q(W|X, y) = L(W|X, y) + \xi \Omega(W),$$

(1)
where \( \Omega(W) \) is a penalty function over the parameters \( W = [w_1, w_2, \ldots, w_p] \) according to some pre-specified criteria, \( \mathcal{L} \) is the standard loss function e.g. cross entropy for classification, and \( \xi \) is a non-negative hyperparameter controlling the trade-off between the loss and penalty. Unlike network pruning, the approach simultaneously learns a sparse set of parameters while training the large network. As such, we denote these approaches as one-stage training procedures.

**Sparsity Inducing Regularization:** Early works in neural network compression with regularization augment the loss function with a sparsity inducing penalty typically based on the Lasso \([62]\) to attain sparse neural networks and prevent overfitting \([9, 11, 32, 59]\). More recently, Collins et al. \([12]\) applied \( L_q \) norm penalties (also called bridge penalties) to achieve sparse networks with 4X memory compression over the original network and a minimal decrease in accuracy on ImageNet \([34]\).

The standard \( L_q \) norm penalty is written as \( \Omega_q(W) = \left( \sum_{j=1}^{p} |w_j|^q \right)^{1/q} \) for which the popularly used Lasso \( q = 1 \) and ridge (weight decay) \( q = 2 \) penalties are special cases. Other extensions of the Lasso aimed at correcting the bias in the Lasso estimator including the trimmed Lasso \([72]\) and other nonconvex penalties like minimax concave penalty (MCP) \([74]\) and smoothly clipped absolute deviations (SCAD) \([17]\) have also been applied to deep neural networks \([66]\). However these studies were limited to smaller image classification datasets such as MNIST and Fashion-MNIST. They found marginal improvements over the \( L_1 \) penalty in some cases, and did not aim for highly sparse networks. Recently \([52, 61]\) examined theoretical properties of Lasso-type and non-convex regularization for neural networks. Other works also induce sparsity through Bayesian hierarchical models \([46, 49, 54, 64]\); similarly in Section 4, we discuss sparsity and posterior convergence of the Bayesian hierarchical model corresponding to HALO.

**Algorithms for Optimizing Regularizers:** Contemporary literature has explored algorithms optimizing regularizers for better generalization performance \([41, 48, 59]\); the latter of which optimizes the weight decay parameter, which is similar to our approach, however weight decay does not directly induce sparsity as heavily as \( L_1 \) norm penalties. Further, although we optimize our regularization coefficients jointly with the training data, they could be optimized over a validation set for improving generalization and test set performance.

### 2.3 Learning Masks and Weights

Another one-stage procedure for learning sparse neural networks is through learning a binary mask over the network’s parameters. This pruning problem is formulated as

\[
Q(W | X, y) = \mathcal{L}(h(M) \odot W | X, y) + \xi \Omega(W) + \xi_2 \Omega_2(M),
\]

where \( M \) are additional trainable parameters, \( h(M) \) is a pre-specified mask function \( h : \mathbb{R} \to \{0, 1\} \) and in some cases additional penalties might be applied to \( W \) and \( M \) to further inducing sparsity.

Numerous works suggest different approaches for selecting the mask function \( h \) \([4, 35, 59, 56, 68, 71]\). Other works explore a Bayesian model for the mask formulation \([33, 42]\) which we explore in Section 3 where we discuss point mass priors for inducing sparsity in Bayesian hierarchical models. While this class of models has optimal frequentist properties, the posterior is computationally intractable in many cases and difficult to optimize, and further does not perform any feature selection of non-zero weights. Although \((2)\) appears to be very similar to our objective, we note that the class of estimators our approach is based on and this approach are quite different in how they prune models. We elaborate on this difference in Section 3.3.

### 3 Penalization and Bayesian Hierarchical Models for Inducing Sparsity

In this section, we provide background on sparsity inducing regularization and Bayesian shrinkage estimation, and establish connections between our approach and magnitude pruning strategies through the study of sparsity inducing priors, and contrast this with the approach taken by methods which learn a binary mask over the network parameters. We start by recalling the \( L_0 \) penalty for learning sparse models

\[
\Omega_0(W) = \sum_{j=1}^{p} \mathbb{1}[w_j \neq 0]
\]

which induces sparsity by penalizing the number of non-zero entries in \( W \) without any further bias on the weights \( W \) of the model. However, the penalty is computationally intractable as it is
non-differentiable and the learning problem is NP-hard. An alternative to $L_0$ regularization is $L_1$ regularization obtained by adding the $\Omega_1(W)$ penalty, its tightest convex relaxation. The associated estimator is called the Lasso estimator [62].

An added benefit of the Lasso penalty is that it controls the shrinkage of coefficients of $W$ to 0. In (1), as $\xi$ increases, the coefficients move towards 0, and for sufficiently large $\xi$ the coefficients may be exactly 0. Among the penalized least squares methods, the Lasso is the most widely used and extensively studied. It is relatively easy to compute as it is a convex minimization problem and is typically learned via a one-stage training procedure. Additionally it has the added benefit of performing feature selection by reducing the magnitude of non-zero weights, a bias which can improve prediction performance without lowering variable selection performance.

Although the Lasso has strong oracle properties under certain conditions, it is a biased estimator [17]. The Lasso requires a neighborhood stability/strong irrepresentable condition on the design matrix $X$ for the selection consistency [63, 67, 77]. Recently [61] examined Lasso-type regularization of neural networks and showed that the prediction error increases at most sub-linearly in the number of layers and at most logarithmically in the total number of parameters. In the next sections, we discuss different attempts to correct the bias of the Lasso and further induce sparsity, and further connect these strategies to commonly used strategies for inducing sparsity in neural networks.

### 3.1 The Weighted Lasso and Magnitude Pruning

Given that the Lasso is a biased estimator determined by the regularization coefficient $\xi$, one approach to reducing the bias is to select a different regularization coefficients for each parameter resulting in the weighted Lasso penalty:

$$
\Omega_{\text{weighted}}(W) = \sum_{j=1}^{p} \lambda_j |w_j|, \quad (4)
$$

which requires choosing suitable regularization coefficients. Zou [79] proposed the adaptive Lasso for controlling the bias in the Lasso estimator and improving penalization of large weights. The adaptive Lasso is an extension of the weighted Lasso which sets $\lambda_j = \frac{1}{\hat{w}_j}$ where $\hat{w}_j$ is an initial estimate from another run of OLS or Lasso. The procedure is also shown to have oracle properties. Special cases of the weighted Lasso which are relatively simple to implement are the relaxed Lasso [44] or the magnitude pruning approach [21, 25] which set

$$
\lambda_j = \begin{cases} 
0 & \hat{w}_j \neq 0 \\
\infty & \hat{w}_j = 0 
\end{cases} \quad \text{or} \quad \lambda_j = \begin{cases} 
0 & |\hat{w}_j| \leq |\hat{w}_1^{(\alpha)}| \\
\infty & |\hat{w}_j| > |\hat{w}_1^{(\alpha)}|
\end{cases}
$$

where $|\hat{w}_1^{(\alpha)}|$ is the pre-specified $\alpha$ percentile (here the value corresponding to the 95th percentile). This approach is often known as a two-stage approach since two estimates for the parameters are made.

A more general form for the adaptive Lasso which extends to other sparsity-inducing penalties and any general loss function $L(\cdot)$ is known as the local linear approximation algorithm (LLA) which iteratively solves the objective in $k$ iterations:

$$
W^{(k+1)} = \arg \max_W \left[ L(W) - \sum_{j=1}^{p} \Omega(w_j^{(k)})|w_j|_1 \right] \quad (5)
$$

where $w_j^{(k)}$ are the weights for the previous iteration of LLA [80].

### 3.2 Nonconvex Penalties

An alternative approach to the two-stage approach is a single-stage approach which uses a penalty that diminishes in value for large parameter values. These types of penalties are non-convex but have been shown to yield both empirical and theoretical results [66, 74, 80]. Fan and Li [17] proposed a non-convex penalty, smoothly clipped absolute deviation (SCAD) penalty, to remove the bias of the Lasso and proved an oracle property for one of the local minimizers of the resulting penalized loss. Zhang [74] proposed another non-convex penalization approach, minimax concave penalty (MCP):

$$
\Omega_{MCP}(W; \gamma, \lambda) = \begin{cases} 
\lambda |w_j| - \frac{w_j^2}{\gamma} & |w_j| \leq \gamma \lambda \\
\frac{\lambda^2 \gamma^2}{2} & else
\end{cases}
$$

(6)
and proved selection consistency. It has been shown that for some nonconvex penalty functions such as the SCAD penalty, or MCP that the LLA yields an optimal solution when \( k = 1 \), and as such nonconvex penalties are a more efficient class of penalties [58, 80]. Further, this class of nonconvex penalties is preferred to other penalties as they have been shown to be the optimal class of penalties for achieving sparsity and unbiasedness of the regression parameter estimates [80].

### 3.3 Bayesian Hierarchical Priors

From a Bayesian point of view we can consider (1) as a log posterior density, and with this interpretation the penalty \( \xi(\Omega(W)) \) can then be identified with a log prior distribution of \( W \). Constructing estimates via optimization of (1) then gives a maximum a posteriori (MAP) estimation procedure. Bayesian hierarchical models have turned out to be useful modeling and estimation approaches since their model structure allows “borrowing strength” in estimation. This means that the second stage prior affects the posterior distribution by shrinking the estimates towards a central value. The importance of such an inferential procedure has been documented and studied in detail in the frequentist, Bayes, and empirical Bayes literature [20].

For the Lasso estimator, the log prior distribution is a Laplace(\( \lambda \)) prior on \( W \). Strawderman et al. [58] study the estimator given by (6) from a hierarchical Bayes perspective. The intuition is that \( \exp\{\Omega_{MCP}(W,\lambda;\gamma)\} \) is a hierarchical prior with the first level being a Laplace (\( \lambda \)) prior on \( W \) (as with the Bayesian Lasso) and the second level is a half normal prior on the hyperparameter \( \lambda \).

\[
p(w|\psi) = N_p(0, \psi I_p), \quad p(\psi|\gamma) = Gamma\left(\frac{p + 1}{2}, \frac{\gamma^2}{2}\right), \quad p(\gamma|\alpha, \lambda) = TN\left(\lambda, \frac{1}{2\alpha}\right)
\]

They further studied the priors of the corresponding hierarchical Bayes procedure as part of a class of scale mixture priors similar to those used in dropout [49] and demonstrated that the MAP estimate for this procedure is equivalent to optimization with the MCP penalty for the linear model [58, Remark 4.3]. In our experiments we denote the MAP estimate as SWS.

An alternative to the normal scale mixture priors are a type of mixture priors

\[
p(w) = \omega g(w) + (1 - \omega)\delta_0, \quad p(\omega) = Bernoulli(\pi)
\]

often referred to as a spike and slab prior [45] which is defined as a mixture of a point mass at zero and a continuous distribution \( g \). The spike and slab prior is quite reasonable since it reflects a model for sparsity that accounts for the \( \omega \) proportion of non-zero weights and assigns probability \( 1 - \omega \) to the zero weights. Bayes estimates under this class of priors give exact zero estimates and no additional thresholding is needed. [75] and [22] show that under certain regularity conditions spike and slab priors have a number of optimal frequentist properties. Variants of this formulation have been studied recently for pruning deep neural networks where a MAP estimate is approximated by learning a continuous function representing a mask over the weights of the network. [4, 35, 39, 42, 56, 68, 71].

The primary drawback of spike and slab priors is that posterior computation is much more demanding than for single component continuous shrinkage priors [53] since sampling of the point mass part of the posterior distribution can entail searching over an enormous set of of binary indicators and is not feasible in even moderately large parameter spaces. Additionally this class of priors may not effectively penalize the non-zero parameters in \( W \) leading to worse predictive performance over normal scale mixture priors.

### 4 The Hierarchical Adaptive Lasso (HALO)

Although the MCP has desirable properties among shrinkage estimators, a primary drawback is that all weights \( w \geq \gamma \lambda \) of the model are penalized equally. For the standard MCP term these are both hyperparameters of the model which must be provided a priori and directly influence the sparsity of the model, and in the hierarchical model, they are derived from the hierarchical Gamma and truncated normal priors [58]. We extend this model by considering an additional level of hierarchy and by making the penalty adaptive such that each \( w_j \) has its own \( \lambda_j \) in the scale mixture of normals.
representation. We consider a Laplace(\(\lambda\)) prior and further place a mixing distribution on the positive scale parameter of the exponential mixing distribution in the mixture of normals representation of the Laplace distribution. A gamma mixing distribution on the natural parameter of the exponential leads to a subclass of the gamma-gamma prior distribution developed in \([5][23]\). This additional level in the hierarchy is similar to the Horseshoe+ prior which consists of two positive Cauchy distributions \([6]\), and in contrast to the single level mixture of normals prior used for dropout in Table 1 of \([49]\). The additional level of the hierarchy in (9) allows for additional shrinkage and sparsity, a role that falls on the simpler penalization such as in \([7]\).

However, estimation and computation of the posterior distribution for this model can be difficult especially for neural networks with millions of parameters. Instead, we consider the MAP estimate for this model for which we write a generalized version of the penalty:

\[
\Omega_{HALO}(W, \lambda; \psi, \xi) = \xi \sum_{j=1}^{p} h(\lambda_j)|w_j| + \psi \sum_{j=1}^{p} |\lambda_j|, \tag{9}
\]

where \(h(\cdot)\) is a positive function. We call this penalty the Hierarchical Adaptive Lasso (HALO) since the hierarchical and adaptive penalty places an additional \(L_1\) norm on the regularization coefficients \(\lambda_j\). For the MAP estimate both \(\lambda\) and \(W\) are trainable parameters in the optimization allowing for learning of the appropriate amount of shrinkage and the weights of the model.

In our experiments, we set \(h(\lambda) = 1/\lambda_2\) so that \(h(x) \to \infty\) as \(x \to 0\); this combined with the \(L_1\) penalty on \(\lambda\) encourages selective shrinkage of the weights where important weights remain unregularized, and makes HALO a monotonic penalty \([7][19]\). This is more flexible than adaptive Lasso methods which fix regularization coefficients each iteration. We modify the SWS penalty to have the same functional penalty as well, and in Section 8.1.1 we explore and suggest alternatives for \(h(\cdot)\). Additionally the theorem below gives conditions under which using HALO in (1) is convex. A proof of Theorem 1 is provided in Section 8.2.

**Theorem 1.** Consider the objective for the penalized linear model

\[
L(W, \lambda) = L(W) + \Omega(W, \lambda; \psi, \xi) = \|y - \sum_{j=1}^{p} x_j w_j\|^2 + \xi \sum_{j=1}^{p} \frac{1}{\lambda_j^2} |w_j| + \psi \sum_{j=1}^{p} |\lambda_j|,
\]

and let \(X\) be a full rank \(n \times p\) matrix with smallest singular value \(\nu\). Define

\[
\mathbb{R}_{\lambda,W} = \left\{ W, \lambda : 24\nu \frac{\lambda^{10}}{|w_p|} \leq \xi^3 \leq 24\nu \frac{\lambda_{10}}{|w_1|} \right\}
\]

with \(\frac{\lambda_{10}}{|w_1|} \geq \frac{\lambda_{10}}{|w_2|} \geq \cdots \geq \frac{\lambda_{10}}{|w_p|}\).

Then \(\nabla^2 L(W, \lambda) > 0\) and \(L(W, \lambda)\) is elementwise convex over \(\mathbb{R}_{\lambda,W}\).

### 4.1 Posterior Concentration

We will next give a theoretical development for the penalty in (9). An assessment of the goodness of an estimator is some measure of center of the posterior distribution, such as the posterior mean or mode. The natural object to use for assessing feature recovery is a credible set that is sufficiently small enough to be informative, yet not so small that it does not cover the true parameter. The goal is to have a posterior distribution that contracts to its center at the same rate at which the estimator approaches the true parameter value. More formally, the prior gives rise to posterior contraction if the posterior mass of the set \(\{w : \|w - w_0\|^2 \geq M p_n \log(n/p_n)\}\) converges to zero, where \(w_0\) is the true parameter, \(p_n = o(n)\), and \(M\) is a constant. \([22]\) show that the posterior concentration at a particular rate implies the existence of a frequentist estimator that converges at the same rate. Consequently, if the posterior contraction rate is the same as the optimal frequentist estimator, the Bayes procedure also enjoys optimal properties.

For example, consider the Laplace prior, as used in a Bayesian approach to the Lasso, it is well-known that the Laplace distribution with parameter \(\lambda\) can be represented as a scale mixture of normals where the mixing density is exponential with parameter \(\lambda^2\), however Theorem 7 in \([10]\) shows that if the true vector is zero, the posterior concentration rate shown in the full posterior does not shrink.
at the minimax rate. Van der Pas et al. [65] studies a broad class of scale mixtures of normal distributions and provide general conditions on the prior on the scale parameters such that posterior concentration at the minimax estimation rate is guaranteed. Van der Pas et al. [65] comment that the normal-gamma-gamma mixture representation (see [23]) for the HALO induced prior falls within their framework and this prior satisfies their sufficient conditions for posterior concentration at the minimax estimation rate.

5 Numerical Results

We present results to motivate and justify the use of HALO as a penalty for sparsifying deep neural networks. To do so, we perform numerical experiments on image classification datasets with different network architectures which aim to answer the questions

1. Does simultaneously learning model parameters and sparsity improve model performance under highly sparse scenarios?
   Section 5.2-5.3 Yes. In all of our experiments on image classification datasets, the HALO penalty leads to models with comparable or higher accuracy than competing methods.

2. Does the HALO penalty induce a particular type of sparsity?
   Section 5.4 Yes. The HALO penalty is a monotonic penalty which learns both layer-wise sparsity and low-dimensional feature representations (like PCA or another low-rank factorization of the weight parameters). This means the penalty may also lead to direct computational benefits without directly imposing structured penalization.

3. Does the HALO penalty also prevent overfitting in neural networks?
   Section 5.5 Yes. On datasets with label noise, HALO learns to ignore irrelevant samples reducing the generalization gap by over 40% and improving performance by over 10% over standard training with weight decay.

5.1 Experimental Setup

In our experiments the following methods are evaluated for maintaining accuracy while inducing sparsity, and at high sparsity levels (95%): baseline (weight decay), GraSP [68], random initialization pruning [40], lottery ticket hypothesis [21], Lasso ($\ell_1$), MCP [74], SWS [7], and the MAP estimate for HALO [9].

We define sparsity to be the ratio of zero weights in the network to the total number of weights. For all classification results, unless otherwise stated the results represent an average over five runs and the error bars represent one standard deviation. Reported sparsity values are estimated from a single run of the model. For our training procedure, we use weight decay for all methods, and train using the SGD optimizer with momentum and decay the learning rate during training. Additional training and regularization hyperparameter details, as well as run-time for all approaches are discussed in Section B.3.2 and B.3.4. We use standard benchmark networks and datasets for evaluating pruning methods from [21, 40].

5.2 Feedforward Networks on MNIST

| Experiment          | LeNet-300-100 | LeNet-5-Caffe |
|---------------------|---------------|--------------|
|                     | Accuracy      | Sparsity     | Sparsity at Baseline | Accuracy      | Sparsity     | Sparsity at Baseline |
| Baseline            | 98.57 (±0.04) | 0.65        | 0.65                | 99.24 (±0.08) | 0.96        | 0.96               |
| Random Init Magnitude Pruning | 98.23 (±0.01) | 0.95        | 0.95                | 98.91 (±0.15) | 0.95        | 0.95               |
| Lottery Ticket      | 98.44 (±0.13) | 0.95        | 0.95                | 99.00 (±0.03) | 0.95        | 0.95               |
| $\ell_1$            | 98.29 (±0.02) | 0.95        | 0.95                | 98.96 (±0.17) | 0.95        | 0.95               |
| SWS                 | 98.17 (±0.11) | 0.95        | 0.95                | 98.96 (±0.15) | 0.95        | 0.95               |
| HALO                | 98.40 (±0.10) | 0.95        | 0.95                | 99.12 (±0.21) | 0.95        | 0.95               |

Table 1: Results of sparsity-inducing regularization and one-shot magnitude pruning based methods for LeNet-300-100 and LeNet-5-Caffe on MNIST. To obtain the reported sparsity results for the baseline, we threshold values from one run at 0.01.

We choose 95% sparsity for comparison with prior work, particularly [40], and since this still yields competitive performance with the baseline model in most experiments.
We evaluate on the LeNet-300-100 network on the MNIST dataset which consists of three fully connected layers outputting a size 300 feature vector, size 100 feature vector, and the output probabilities respectively. The results are summarized in Table 1. Overall, all methods perform similarly with the baseline model, and HALO and the lottery ticket hypothesis pruning method perform slightly better. Both regularization approaches and network pruning procedures perform similarly and nearly identical to the performance of the original model. It’s important to note that although results are similar, the regularization procedures use only one-stage of training and thus use half the number of epochs for training as the pruning methods. Of the regularization procedures, HALO is the only procedure with nearly equivalent performance to the model pruned by the lottery ticket hypothesis.

We additionally evaluate with the LeNet-5-Caffe network which consists of two convolutional layers followed by three fully connected layers, as summarized in Table 1. Like the results with LeNet-300-100, we find that both HALO and the lottery ticket pruning achieve above 99% accuracy achieving a decrease in performance under 0.25%. The HALO regularization procedures perform better than the network pruning counterparts and regularization approaches attaining a decrease of only roughly 0.1%, although all methods overall perform similarly.

### 5.3 Convolutional Networks on CIFAR-10 and CIFAR-100

We evaluate on a VGG-like network, which has the same convolutional layers as VGG-16 with batch norm and a single fully-connected layer, and the 50 layer ResNet architecture from [40]. Because both network architectures are fully-convolutional except for the last fully connected layer, we prune weights only in the convolutional layers to match the experimental setup of [40].

| Experiment         | VGG-Like CIFAR-10 | ResNet-50 CIFAR-10 |
|--------------------|-------------------|-------------------|
|                     | Accuracy | Sparsity | Sparsity at Baseline | Accuracy | Sparsity | Sparsity at Baseline |
| Baseline           | 93.17 (±0.20)    | 0.4316       |                        | 93.48 (±0.12) | 0.1626   |                      |
| Grasp              | 92.52 (±0.10)    | 0.95         | 0.21%                  | 88.90 (±0.10) | 0.95     | 0.25                |
| Random Int Magnitude Pruning | 93.05 (±0.21) | 0.95 | 0.9                     | 89.59 (±0.09) | 0.95     | 0.25                |
| Lottery Ticket     | 93.18 (±0.12)    | 0.95         | 0.9                    | 88.75 (±0.18) | 0.95     | 0.4                 |
| L1                 | 93.51 (±0.11)    | 0.95         | 0.9                    | 89.10 (±0.46) | 0.95     | 0.6                 |
| MCP                | 92.28 (±0.07)    | 0.95         | 0.85                   | 89.54 (±0.24) | 0.95     | 0.4                 |
| SWS                | 93.50 (±0.15)    | 0.95         | 0.9                    | 88.67 (±0.29) | 0.95     | 0.5                 |
| HALO               | 93.61 (±0.16)    | 0.95         | 0.9                    | 90.71 (±0.17) | 0.95     | 0.55                |

Table 2: Results of sparsity-inducing regularization and one-shot magnitude pruning based methods for the VGG-like and ResNet-50 on CIFAR-10 and CIFAR-100. To obtain the desired sparsity for the base model, we threshold values at 0.01 for ResNet-50 and 0.001 for VGG-like. Results from baseline and pruning methods at 95% sparsity are taken from [40]. For sparsity at baseline results, we compute the highest sparsity ratio that achieves performance within 0.15 of the baseline.

Results for the VGG and ResNet architectures on the CIFAR-10 and CIFAR-100 image classification dataset are summarized in Table 2. For the VGG-like architecture on CIFAR-10, we find that regularization approaches outperform the network pruning counterparts. We also find that the HALO penalty achieves the best performance. All approaches achieve similar accuracy to the baseline model (without sparsity). For ResNet-50 on CIFAR-10, we note that the SWS penalty performs similarly to the pruning based methods, however the HALO approach outperforms both approaches by 2% achieving an accuracy of 90.71, and L1 regularization improves by 0.5%. Finally, we note that while the VGG-like architecture has little drop in performance compared with the baseline model, the sparse ResNet-50 models have a much larger difference. This is likely due to the absolute sizes of the models where the VGG-like architecture has over 10^5 model parameters, while the ResNet architecture has under 10^5 parameters. So, if the same ratio of parameters is kept, the pruned ResNet architecture has many fewer parameters.

2The reported sparsity is not the highest sparsity ratio attainable by the baseline models, but rather a reference indicating that although weight decay does not directly induce sparsity it is still possible to threshold weights to zero. However if weights are set to attain 95% sparsity, accuracy drops to near random for the pruned baseline.
For VGG on the CIFAR-100 dataset, most sparse models drop performance over the baseline model by almost 3%. The best performing model is the one learned from the HALO penalty which achieves an accuracy drop of only 1% on CIFAR-100 and increases accuracy by almost 2% over other methods. On the ResNet-50 architecture the difference is even more evident where the HALO approach achieves a model with 4% higher accuracy over other pruned models and 0.5% higher than MCP which performs more competitively. Still, no pruned model performs comparably at 0.95 sparsity with the baseline model. In some scenarios high accuracy may be more desirable at lower levels of sparsity. Results for one run of each method is provided in Section 8.3.3 where we compute the highest sparsity ratio a model can have to retain accuracy within 0.15% of the baseline accuracy. For GraSP, we used 0.3% of the baseline accuracy since the performance of GraSP fluctuates more. In this regime, we find that models trained by HALO are able to retain the same accuracy at sparsity ratios 0.85 and 0.7 on CIFAR-100, higher ratios compared with other methods and are competitive for both architectures on CIFAR-10, in contrast with other methods which typically only retain the baseline accuracy at much lower levels of sparsity.

5.4 Learning Different Types of Sparsity

We have shown that our approach can prune different network architectures in order to achieve small models with little drop in performance. In this section, we investigate the type of sparsity learned through the HALO penalty. In particular, we highlight through exploration on the VGG-like architecture that the HALO penalty performs monotonic penalization, learns structured sparsity, and learns low-rank feature representations.

Monotonic Penalization: We visualize model parameters and regularization coefficients from the VGG-like network trained with HALO on CIFAR-100 in Figure 1a. Except for weights which are nearly 0, the there is an increasing (potentially quadratic) trend in the relationship between the model parameters and regularization coefficients. This indicates that HALO correctly enforces small weights to be zero, while allowing large weights to contribute uninhibited.

Approximately Structured Penalization: To take advantage of unstructured sparsity, sparse libraries or special hardware is required to deploy such networks, and recent work has aimed at pruning layers, filters, or channels of the network [2, 43, 47, 57, 70, 76]. We note that while it is not guaranteed for HALO to learn sparse group representations, our procedure learns to nearly sparsify complete layers yielding more efficient networks without needing sparse libraries or other mechanisms as shown in Figure 1b. We find that for early layers (before layer 5) the sparsity ratio is low, for middle layers (layers 5-8) there is a sharp increase in the sparsity level, and above layer 8, layers are near fully sparse. Interestingly layer 2 exhibits a high amount of sparsity (approximately 70%) on CIFAR-100, and the sparsity seems to exhibit a pattern every few layers. The pattern appears to arise from the structure of the VGG architectures which separate convolutional layers with max-pooling layers with large jumps or discontinuities occurring after max-pooling layers.
Figure 2: Training and test errors during training for model with the HALO regularizer and baseline model with only weight decay. The mean accuracy over 5 runs and one standard deviation are shown.

**Low-Rank Penalization:** We additionally plot the normalized cumulative sum of the eigenvalues of the covariance matrix generated from the outputs of each convolutional layer, which reflects the dimensionality of the output feature space. For the CIFAR-100 training set there is a sharp trend which plateaus at 1 after only very few eigenvalues. This indicates that the covariance matrix of the outputs is low-rank, and that the HALO penalty learns a model producing a low-dimensional representation of the feature spaces, much as other low-rank factorization approaches which might apply a low-rank matrix factorization (such as PCA) to the intermediate layers would learn. Although untested, this type of representation is also learned to de-correlate and prune filters in [13] and our approach may lead to structured sparsity in the filters of the convolutional layers as well. We note that as with the structured penalization, the intermediate layers have the largest effective dimensionality. Along with the increase in sparsity during the intermediate layers, this may imply that the middle layers are the most feature-rich for image classification, and are an important subject for future work.

**5.5 Regularization for Limiting Overfitting**

In addition to inducing sparsity and pruning neural network architectures, regularization is a tool for limiting overfitting to noisy data [1, 73]. A common setting where overfitting can occur is in the presence of label noise where the label of each image in the training set is independently changed to another class label with probability \( \rho \). We implement training of the VGG-like architecture with varying label noise on CIFAR-100. Figure 2 shows the training accuracy for the baseline model approaches 100% while the test error does not when \( \rho > 0 \). For the VGG-like model trained with the HALO penalty (using suitable hyperparameters to match test accuracy at \( \rho = 0 \)), we find that at \( \rho = 0.0 \), both train and test curves follow nearly identical patterns, whereas at \( \rho = 0.4, 0.7 \) the training accuracy does not increase to 100%, but reaches similar accuracy proportional to the clean images. At \( \rho = 0.4 \), the test error is roughly the same as the training error indicating a small generalization gap unlike with standard training, while for \( \rho = 0.7 \) the test accuracy is higher indicating underfitting due to the lack of data. The difference in performance between the baseline and the model trained with the HALO penalty is around 10% and 20% for \( \rho = 0.4 \) and 0.7 respectively. We further note that the network learned by HALO is sparse while still maintaining better performance.

**6 Conclusion**

Over-parameterization is a challenging problem preventing the use of deep neural networks in settings with limited computational budgets. In this work, we present HALO, a novel penalty function which when used to train neural networks produces subnetworks which achieve state-of-the-art performance compared with magnitude criteria pruning techniques, without re-training the subnetwork. Our approach has several benefits. It is simple to implement and does not require storing model weights or re-training unlike other network pruning methods. Although we only demonstrated results for image recognition, our approach can be combined with other loss functions and is not limited to classification, and is model agnostic. We believe that this work is one step towards creating penalty functions that can be applied to sparsify any model without performance degradation. Further research
in this area can lead to impressive performance gains in settings with limited computational resources, and towards a better understanding of the rich feature representations learned by neural networks.

Acknowledgments and Disclosure of Funding

We thank Ben Baer for his helpful advice and discussions regarding the LLA algorithm and one-step estimators in Section 3 as well as general pointers for optimization with sparsity-inducing penalties and shrinkage priors. Wells’ research supported by NIH grant R01GM135926.

7 NeurIPS Broader Impact Statement

Deep neural networks have made major advances towards solving problems in image recognition [34], speech recognition [29], natural language understanding [15], and healthcare [10], however this performance comes with increased model size making them unusable in low-resource scenarios. Our research on HALO aims to sparsify models improving computational efficiency and storage capabilities allowing deep neural networks to be deployed in new settings.

Improvements from sparse models to computational efficiency and storage of machine learning models will likely increase willingness for professionals to use such systems on low-power devices. In computer vision, for example, phone CPUs and camera modules typically have less processing capability than those in personal computers or larger devices making deployment of large networks (with over 100 million parameters and 20 gigaFLOPs) for improving image quality difficult; however sparse models could make deploying such models possible with increased computational efficiency and fewer FLOP counts equating to less power consumption, and less storage requirements leading to more available space for users. In healthcare applications, smaller models could lead to deployment on smaller devices leading to less invasive apparatuses for detecting medical conditions. More generally, models which require less computation may lead to “smart” devices with a smaller carbon footprint, an important environmental benefit for any device, and more affordable “smart” devices as the cost to manufacture less computationally intensive parts becomes cheaper.

While there may be important benefits deploying neural networks in new low-resource scenarios, there are potential risks to using pruned models and deploying these models in new settings. These include (1) the use of sparse machine learning models in place of the original models risks lower performance if the sparsity ratio is too large leading to distrust and unwillingness to use a technology, (2) the added risk to user privacy from data collection and monitoring in novel scenarios such as increased surveillance from tiny (smart) cameras, or health data from low-resource non-invasive medical devices, and (3) to fully take advantage of sparse models, development of new hardware that relies on sparse data structures and models may have limited functionalities in other use cases.

Nonetheless, we believe that the potential for new low-resource technologies to have smart capabilities based on sparse neural networks makes this research important and impactful. It is our hope that researchers in machine learning, medicine, and the greater tech industry continue to investigate the use of sparse models in the range of applications described above with a focus on (1) measuring the environmental and cost benefits gained through computational and storage efficiency as these models draw less power consumption, and (2) identifying new scenarios where devices did not have the capability to use AI due to model size.
References

[1] Subutai Ahmad and Luiz Scheinkman. How can we be so dense? the benefits of using highly sparse representations. *arXiv preprint arXiv:1903.11257*, 2019.

[2] Jose M Alvarez and Mathieu Salzmann. Compression-aware training of deep networks. In *Advances in Neural Information Processing Systems*, pages 856–867, 2017.

[3] Sanjeev Arora, Rong Ge, Behnam Neyshabur, and Yi Zhang. Stronger generalization bounds for deep nets via a compression approach. *arXiv preprint arXiv:1802.05296*, 2018.

[4] Kambiz Azarian, Yash Bhalgat, Jinwon Lee, and Tijmen Blankevoort. Learned threshold pruning. *arXiv preprint arXiv:2003.00075*, 2020.

[5] José M Bernardo and Adrian FM Smith. *Bayesian theory*, volume 405. John Wiley & Sons, 2009.

[6] Anindya Bhadra, Jyotishka Datta, Nicholas G Polson, Brandon Willard, et al. The horseshoe+ estimator of ultra-sparse signals. *Bayesian Analysis*, 12(4):1105–1131, 2017.

[7] Malgorzata Bogdan, Ewout van den Berg, Weijie Su, and Emmanuel Candes. Statistical estimation and testing via the sorted l1 norm. *arXiv preprint arXiv:1310.1969*, 2013.

[8] Patrick Breheny and Jian Huang. Penalized methods for bi-level variable selection. *Statistics and its interface*, 2(3):369, 2009.

[9] Miguel A Carreira-Perpinán and Yerlan Idelbayev. “learning-compression” algorithms for neural net pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8532–8541, 2018.

[10] Ismaël Castillo, Johannes Schmidt-Hieber, Aad Van der Vaart, et al. Bayesian linear regression with sparse priors. *The Annals of Statistics*, 43(5):1986–2018, 2015.

[11] Yves Chauvin. A back-propagation algorithm with optimal use of hidden units. In *Advances in neural information processing systems*, pages 519–526, 1989.

[12] Maxwell D Collins and Pushmeet Kohli. Memory bounded deep convolutional networks. *arXiv preprint arXiv:1412.1442*, 2014.

[13] Xavier Suau Cuadros, Luca Zappella, and Nicholas Apostoloff. Filter distillation for network compression. March 2020.

[14] Yann N Dauphin and Yoshua Bengio. Big neural networks waste capacity. *arXiv preprint arXiv:1301.3583*, 2013.

[15] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[16] Andre Esteva, Alexandre Robicquet, Bharath Ramsundar, Volodymyr Kuleshov, Mark DePristo, Katherine Chou, Claire Cui, Greg Corrado, Sebastian Thrun, and Jeff Dean. A guide to deep learning in healthcare. *Nature medicine*, 25(1):24–29, 2019.

[17] Jianqing Fan and Runze Li. Variable selection via nonconcave penalized likelihood and its oracle properties. *Journal of the American statistical Association*, 96(456):1348–1360, 2001.

[18] Ky Fan. On a theorem of weyl concerning eigenvalues of linear transformations i. *Proceedings of the National Academy of Sciences of the United States of America*, 35(11):652, 1949.

[19] Long Feng, Cun-Hui Zhang, et al. Sorted concave penalized regression. *The Annals of Statistics*, 47(6):3069–3098, 2019.

[20] Dominique Fourdrinier, William E Strawderman, and Martin T Wells. *Shrinkage Estimation*. Springer, 2018.
[21] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

[22] Subhashis Ghosal, Jayanta K Ghosh, Aad W Van Der Vaart, et al. Convergence rates of posterior distributions. *Annals of Statistics*, 28(2):500–531, 2000.

[23] JE Griffin and PJ Brown. Alternative prior distributions for variable selection with very many more variables than observations. *University of Kent Technical Report*, 2005.

[24] Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient dnns. In *Advances in neural information processing systems*, pages 1379–1387, 2016.

[25] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[26] Babak Hassibi, David G Stork, and Gregory J Wolff. Optimal brain surgeon and general network pruning. In *IEEE international conference on neural networks*, pages 293–299. IEEE, 1993.

[27] Jeff Hawkins and Subutai Ahmad. Why neurons have thousands of synapses, a theory of sequence memory in neocortex. *Frontiers in neural circuits*, 10:23, 2016.

[28] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. pages 1389–1397, 2017.

[29] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine*, 29(6):82–97, 2012.

[30] Jian Huang, Patrick Breheny, and Shuangge Ma. A selective review of group selection in high-dimensional models. *Statistical science: a review journal of the Institute of Mathematical Statistics*, 27(4), 2012.

[31] Zehao Huang and Naiyan Wang. Data-driven sparse structure selection for deep neural networks. pages 304–320, 2018.

[32] Masumi Ishikawa. Structural learning with forgetting. *Neural networks*, 9(3):509–521, 1996.

[33] Yohei Kondo, Shin-ichi Maeda, and Kohei Hayashi. Bayesian masking: Sparse bayesian estimation with weaker shrinkage bias. In *Asian Conference on Machine Learning*, pages 49–64, 2016.

[34] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[35] Aditya Kusupati, Vivek Ramanujan, Raghav Somani, Mitchell Wortsman, Prateek Jain, Sham Kakade, and Ali Farhadi. Soft threshold weight reparameterization for learnable sparsity. *arXiv preprint arXiv:2002.03231*, 2020.

[36] Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605, 1990.

[37] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip HS Torr. Snip: Single-shot network pruning based on connection sensitivity. *arXiv preprint arXiv:1810.02340*, 2018.

[38] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

[39] Junjie Liu, Zhe Xu, Runbin Shi, Ray CC Cheung, and Hayden KH So. Dynamic sparse training: Find efficient sparse network from scratch with trainable masked layers. *arXiv preprint arXiv:2005.06870*, 2020.
[40] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270*, 2018.

[41] Jonathan Lorraine and David Duvenaud. Stochastic hyperparameter optimization through hypernetworks. *arXiv preprint arXiv:1802.09419*, 2018.

[42] Christos Louizos, Max Welling, and Diederik P Kingma. Learning sparse neural networks through \( l_0 \) regularization. *arXiv preprint arXiv:1712.01312*, 2017.

[43] Eran Malach, Gilad Yehudai, Shai Shalev-Shwartz, and Ohad Shamir. Proving the lottery ticket hypothesis: Pruning is all you need. *arXiv preprint arXiv:2002.00585*, 2020.

[44] Nicolai Meinshausen. Relaxed lasso. *Computational Statistics & Data Analysis*, 52(1):374–393, 2007.

[45] Toby J Mitchell and John J Beauchamp. Bayesian variable selection in linear regression. *Journal of the american statistical association*, 83(404):1023–1032, 1988.

[46] Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. Variational dropout sparsifies deep neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2498–2507. JMLR. org, 2017.

[47] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440*, 2016.

[48] Kensuke Nakamura and Byung-Woo Hong. Adaptive weight decay for deep neural networks. *IEEE Access*, 7:118857–118865, 2019.

[49] Eric Nalisnick, José Miguel Hernández-Lobato, and Padhraic Smyth. Dropout as a structured shrinkage prior. *arXiv preprint arXiv:1810.04045*, 2018.

[50] Sharan Narang, Erich Elsen, Gregory Diamos, and Shubho Sengupta. Exploring sparsity in recurrent neural networks. *arXiv preprint arXiv:1704.05119*, 2017.

[51] Dae Hoon Park, Chiu Man Ho, and Yi Chang. Achieving strong regularization for deep neural networks. 2018.

[52] Konstantin Pieper and Armenak Petrosyan. Nonconvex penalization for sparse neural networks. *arXiv preprint arXiv:2004.11515*, 2020.

[53] Juho Piironen and Aki Vehtari. Sparsity information and regularization in the horseshoe and other shrinkage priors. *Electronic Journal of Statistics*, 11, 07 2017.

[54] Nicholas G Polson and Veronika Ročková. Posterior concentration for sparse deep learning. In *Advances in Neural Information Processing Systems*, pages 930–941, 2018.

[55] Vivek Ramanujan, Mitchell Wortsman, Aniruddha Kembhavi, Ali Farhadi, and Mohammad Rastegari. What’s hidden in a randomly weighted neural network? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11893–11902, 2020.

[56] Pedro Savaresi, Hugo Silva, and Michael Maire. Winning the lottery with continuous sparsification. *arXiv preprint arXiv:1912.04427*, 2019.

[57] Simone Scardapane, Danilo Commiariello, Amir Hussain, and Aurelio Uncini. Group sparse regularization for deep neural networks. *Neurocomputing*, 241:81–89, 2017.

[58] Robert L Strawderman, Martin T Wells, Elizabeth D Schifano, et al. Hierarchical bayes, maximum a posteriori estimators, and minimax concave penalized likelihood estimation. *Electronic Journal of Statistics*, 7:973–990, 2013.

[59] Matthew Streeter. Learning optimal linear regularizers. *arXiv preprint arXiv:1902.07234*, 2019.

[60] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in nlp. *arXiv preprint arXiv:1906.02243*, 2019.
[61] Mahsa Taheri, Fang Xie, and Johannes Lederer. Statistical guarantees for regularized neural networks. *arXiv preprint arXiv:*, 2020.

[62] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

[63] Joel A Tropp. Just relax: Convex programming methods for identifying sparse signals in noise. *IEEE transactions on information theory*, 52(3):1030–1051, 2006.

[64] Karen Ullrich, Edward Meeds, and Max Welling. Soft weight-sharing for neural network compression. *arXiv preprint arXiv:1702.04008*, 2017.

[65] SL van der Pas, J-B Salomond, Johannes Schmidt-Hieber, et al. Conditions for posterior contraction in the sparse normal means problem. *Electronic Journal of statistics*, 10:976–1000, 2016.

[66] Sujit Vettam and Majnu John. Regularized deep learning with a non-convex penalty. *arXiv preprint arXiv:1909.05142*, 2019.

[67] Martin J Wainwright. Sharp thresholds for high-dimensional and noisy sparsity recovery using $\ell_1$-constrained quadratic programming (lasso). *IEEE transactions on information theory*, 55(5):2183–2202, 2009.

[68] Chaoqi Wang, Guodong Zhang, and Roger Grosse. Picking winning tickets before training by preserving gradient flow. *arXiv preprint arXiv:2002.07376*, 2020.

[69] Andreas S Weigend, David E Rumelhart, and Bernardo A Huberman. Generalization by weight-elimination with application to forecasting. In *Advances in neural information processing systems*, pages 875–882, 1991.

[70] Wei Wen, Cong Xu, Chunpeng Wu, Yandau Wang, Yiran Chen, and Hai Li. Coordinating filters for faster deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 658–666, 2017.

[71] Xia Xiao, Zigeng Wang, and Sanguthhevar Rajasekaran. AutoPrune: Automatic network pruning by regularizing auxiliary parameters. In *Advances in Neural Information Processing Systems*, pages 13681–13691, 2019.

[72] Jihun Yun, Peng Zheng, Eunho Yang, Aurelie Lozano, and Aleksandr Aravkin. Trimming the l1 regularizer: Statistical analysis, optimization, and applications to deep learning. In *International Conference on Machine Learning*, pages 7242–7251, 2019.

[73] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.

[74] Cun-Hui Zhang et al. Nearly unbiased variable selection under minimax concave penalty. *The Annals of statistics*, 38(2):894–942, 2010.

[75] Dabao Zhang and Min Zhang. Bayesian profiling of molecular signatures to predict event times. *Theoretical Biology and Medical Modelling*, 4(1):3, 2007.

[76] Dejiao Zhang, Haozhu Wang, Mario Figueiredo, and Laura Balzano. Learning to share: Simultaneous parameter tying and sparsification in deep learning. 2018.

[77] Peng Zhao and Bin Yu. On model selection consistency of lasso. *Journal of Machine learning research*, 7(Nov):2541–2563, 2006.

[78] Hattie Zhou, Janice Lan, Rosanne Liu, and Jason Yosinski. Deconstructing lottery tickets: Zeros, signs, and the supermask. pages 3592–3602, 2019.

[79] Hui Zou. The adaptive lasso and its oracle properties. *Journal of the American Statistical Association*, 101(476):1418–1429, 2006.

[80] Hui Zou and Runze Li. One-step sparse estimates in nonconcave penalized likelihood models. *Annals of statistics*, 36(4):1509, 2008.
8 Appendix

8.1 Generalizations of HALO

8.1.1 Other choices of h(x)

In Section 4, we introduce the HALO penalty and in our experiments we consider $h(x) = \frac{1}{x^2}$ to enforce $h(x) \to \infty$ as $x \to 0$. We found that this combined with the $L_1$ penalty on $\lambda_j$ encourages selective shrinkage of the weights where important weights remain unregularized. Using $h(x) = \frac{1}{x^k}$ when $k > 0$ is favorable because $h(x) \to 0$ and $\infty$ as $x \to 0$ and $\infty$ respectively. With this choice of $h(x)$ our penalty is a flexible variant of the adaptive penalties such as magnitude pruning or the relaxed Lasso because the regularization coefficients in these approaches (regularization coefficients are pre-specified as either zero or infinity) are the limit of those learned by HALO. However in some cases it may not be favorable to use a sharp penalty on the regularization coefficients since this may lead to over-sparsification of the weights.

![Figure 3: Comparing different choices of $h(x)$ for HALO.](image)

As an alternative, we investigate $h(x) = \log(|x|)^2$. Figure 3 highlights the key difference: $\log(|x|)^2$ approaches infinity at a much slower rate over $\frac{1}{x^2}$, thus inducing less shrinkage of small coefficients. Another difference is that for $x > 1$, $\log(|x|)^2$ is an increasing function. This means that increasing $\lambda$ will also penalize weights, although this will rarely happen since the $L_1$ penalty acts to shrink the $\lambda$s as much as possible. On experiments with VGG-16 on CIFAR-100 we obtain comparable accuracy of $72.36 \pm 0.22$ for VGG-like on CIFAR-100. This is similar to the performance with $h(x) = \frac{1}{x^2}$ ($72.48 \pm 0.24$).

Although we have not fully explored the range of possible functions for HALO, we believe that the choice of $h(x)$ will be data, model and application dependent. A choice of $h(x)$ such that $h(x) \to \infty$ as $x \to 0$ and $0$ as $x \to \infty$ will lead to a soft-thresholding variant of the common magnitude pruning approaches, while a choice of $h(x) \to \infty$ as $x \to 0$ and $\infty$ will impose additional restraints on weight magnitudes (favoring medium magnitude weights). Still there are many other functions we can explore in future work including piecewise functions that have different behaviors for large and small weights, or those which lead to other penalty behaviors such as sorted penalties [7, 19] which penalize larger weights more strongly.

8.1.2 Structured HALO

In Section 5.4, we demonstrate that HALO learns efficient sparse architectures in performing effective dimensionality reduction of the features and nearly sparsifying entire layers. However, HALO is not a structured penalty and it is not guaranteed to enforce sparsity at a group level. While prior work [43] shows that subnetworks pruned at the neuron-level are unable to attain performance equivalent to trained networks, whereas subnetworks pruned at the weight-level can, in some cases structured sparsity may be more desirable. We propose a natural extension of HALO to structured penalization for deep neural networks. One possible version of a structured HALO (SHALO) penalty is based on the composite penalty framework proposed in [8].
\[ \Omega_C = \Omega_O \left( \sum_{j=1}^{p_g} \Omega_I (|w_{gj}|) \right), \]

where \( \Omega_O \) is some out penalty applied to the sum of inner penalties \( \Omega_I \) and \( w_{gj} \) is the \( j \)th member of the \( g \)th group. This framework is general for group penalties and includes both the group bridge penalty and group Lasso [30]. The partial derivative with respect to the \( gj \)th weight is

\[ \frac{\partial \Omega_C}{\partial w_{gj}} = \frac{\partial \Omega_O}{\partial \Omega_I (|w_{gj}|)} \frac{\partial \Omega_I (|w_{gj}|)}{\partial w_{gj}} \]

and this approach can be applied with the HALO penalty for learning structured sparsity in deep neural networks. We suggest two approaches for extending HALO

1. Apply a Lasso penalty for the inner penalty and HALO for the outer penalty:

\[ \Omega_{SHALO} = \xi \sum_{g=1}^{G} h(\lambda_g) \sum_{j=1}^{p_g} |w_{gj}| + \psi \sum_{g=1}^{G} |\lambda_g|, \]

which learns regularization coefficients for controlling groups of weights only.

2. Apply the HALO penalty for both the inner and outer penalty:

\[ \Omega_{SHALO} = \xi \sum_{g=1}^{G} h(\lambda_g) \sum_{j=1}^{p_g} \Omega_{HALO}(w_g) + \psi \sum_{g=1}^{G} |\lambda_g|, \]

where \( w_g \) is the vector of all weights in the \( g \)th group. This penalty will learn regularization coefficients group-wise and for individual weights.

In future work, we hope to consider these penalties and other structured variants for learning more efficient sparse networks.

8.2 Theorem 1: Convexity of HALO in the Linear Model

We demonstrate that for the standard linear model, the loss function with the HALO penalty is convex. Recall that the loss function with the added penalty is written

\[ L(W, \lambda) = L(W) + \Omega(W, \lambda; \psi, \xi) = \|y - \sum_{j=1}^{p} x_j w_j\|^2 + \xi \sum_{j=1}^{p} \frac{1}{\lambda_j} |w_j| + \psi \sum_{j=1}^{p} |\lambda_j|. \quad (10) \]

**Lemma 1.** For any symmetric matrix

\[ M = \begin{bmatrix} A & B \\ B & C \end{bmatrix} \]

if \( A \) is invertible, then \( M \succ 0 \) if \( A \succ 0 \), and \( C - B^T A^{-1} B \succ 0 \).

We invoke Lemma 1 to show that (10) is convex in both \( W \) and \( \lambda = [\lambda_1, \lambda_2, \ldots, \lambda_p] \):

**Proof.** First, note that \( \nabla^2 L(W, \lambda) \) is symmetric and can be written as a block matrix since

\[ \nabla^2 L(W, \lambda) = \begin{bmatrix} A & B \\ B & C \end{bmatrix} = \begin{bmatrix} \text{diag}(\frac{6\xi w_k}{\lambda_k}) & \text{diag}(\pm \frac{2\psi}{\lambda_k}) \\ \text{diag}(\pm \frac{2\psi}{\lambda_k}) & X^T X \end{bmatrix} \]
Second, since \( A \) is a diagonal matrix with positive values along the diagonal, \( A \) is both invertible and \( A \succ 0 \).

Consider the expression
\[
\Lambda = C - B^T A^{-1} B.
\]

Note that if \( X \) is full rank, then the smallest eigenvalue of \( X^T X \) is \( \nu > 0 \). Further, the eigenvalues of \( B^T A^{-1} B \) are
\[
\rho_k = \frac{4\xi^2}{\lambda_k^6} \cdot \frac{6|w_k|}{\lambda_k^4}.
\]

Then, By Weyl’s theorem \([18]\),
\[
\nu - \rho_p \leq \mu_p \leq \nu - \rho_1
\]

where \( \mu_p \) is the smallest eigenvalue of \( \Lambda \), \( \rho_p \) is the smallest eigenvalue of \( B^T A^{-1} B \), and \( \rho_1 \) is the largest. Then, since
\[
24\nu \frac{\lambda_p^{10}}{|w_p|} \leq \xi^3 \leq 24\nu \frac{\lambda_1^{10}}{|w_1|}
\]

we have
\[
0 \leq \nu - \rho_p \leq \mu_p \leq \nu - \rho_1
\]

and \( \Lambda \succ 0 \).

\[\square\]

Remark: Note that the conditions of the theorem do not depend on \( \psi \). While we believe \( \psi \) cannot be entirely disregarded, based on the above theorem and our experiments in Section 8.3.2, HALO has only one key hyperparameter \( \xi \) rather than two.

8.3 Additional Experiments

8.3.1 Training Configuration

We use the standard train/test split for the MNIST digits dataset containing 60,000 training images and 10,000 test images, and CIFAR-10/0 datasets which contain 50,000 training images and 10,000 test images available from the torchvision dataloaders.

We train all models using SGD with a momentum of \( \gamma = 0.9 \) and weight decay. For MNIST, we use a batch size of 100 and train with an initial learning rate of 0.1 decaying by 0.1 at every 25k batches for 250 epochs, and use weight decay of 0.0005. For CIFAR-10/100 we use a batch size of 64, and train with an initial learning rate of 0.1 decaying by 0.1 at the 80th and 120th epochs for 160 epochs. We set the weight decay parameter to be 0.0001. For CIFAR-10/100 experiments, we use standard data augmentations (random horizontal flip, translation by 4 pixels). Regularization coefficients are initialized at one for all \( \lambda_j \). This also reduces our approach to Lasso for the first batch.

8.3.2 Parameter Robustness

We plot accuracy of one model run according to different values of \( \xi \) and \( \psi \). In our experiments, we set \( \xi = \psi \) in order to eliminate the advantage of our approach having an extra hyperparameter over the \( L_1 \) penalty which does not have the hierarchical term. In our main results, we report accuracy for each regularization approach based on the best performing regularization hyperparameters, although we note that for HALO, there are often a few hyperparameter choices which perform comparably or better than competing methods.

\[\text{https://pytorch.org/docs/stable/torchvision/datasets.html}\]
In summary, we find that for a majority of our experiments, HALO performs reasonably for small to medium values of the hyperparameters and drops for large values, while $L_1$, MCP, and SWS penalties have good accuracy in a range of hyperparameters and experience sharp drops for small or large hyperparameter values outside of the range.

Based on our experiments, a reasonable heuristic for selecting the hyperparameters is based on the number of parameters in the model. We see that using a hyperparameter value equivalent to the reciprocal of the number of parameters in the model yields comparable or the best performance. Intuitively, it is easy to see that this choice is reasonable, since the classification loss in our experiments is typically a single digit value, and choosing such a value for the hyperparameters sets the hierarchical penalty at the same order of magnitude and the regularization loss at a similar value for the first iteration.

![Figure 4: Accuracy of LeNet architectures at 0.95 sparsity across different values of $\xi$ and $\psi$ on MNIST. On the x-axis we plot the regularization coefficients on a $\log_{10}$ scale.](image)

(a) LeNet-300-100  (b) LeNet5-Caffe

**MNIST Hyperparameter Experiments:** We visualize the accuracy of the two LeNet architectures at different regularization hyperparameter values in Figure 4. We find that for the LeNet architecture, HALO yields reasonable performance for any coefficient below $10^{-4}$ then penalizes too strongly for lower values when the regularization hyperparameter is too large. Like HALO, $L_1$ regularization penalizes too strongly for large values of $\xi$ and $\psi$ (above $10^{-4}$), however, $L_1$ regularization does not penalize strongly enough at low values of the regularization hyperparameters. We note that this is because the amount of sparsity induced by $L_1$ is much lower than $95\%$, and the model degrades in accuracy since more of its weights are set to 0.

In contrast to $L_1$ and HALO regularization, we find that the SWS penalty achieves the best results at larger penalty coefficients, particularly at $10^{-3}$. We observe also that in LeNet-300-100, for no regularization hyperparameter is the model unable to learn unlike in $L_1$ and HALO for large coefficient values. However, for LeNet-5 we find a similar behavior as with $L_1$ and HALO decaying to random at 0.1. Overall, we believe that for the MNIST dataset, HALO is the most flexible particularly when setting a smaller value for the hyperparameter or when considering our parameter count heuristic. However, we do find unlike SWS, HALO and Lasso both produce models with poor perform comparable to un-trained models for larger values of $\xi$ and $\psi$.

**CIFAR-10 Hyperparameter Experiments:** Accuracy across different regularization hyperparameter values is shown in Figure 5 for the VGG and ResNet architectures on CIFAR-10. For the VGG-like architecture, we find that HALO yields reasonable performance for values below $10^{-5}$ but drops significantly for larger values. In contrast, $L_1$, SWS and MCP regularization additionally do not penalize strongly enough for smaller values of the hyperparameters. Overall, we note that HALO has the largest range of reasonable hyperparameter values followed by $L_1$ and SWS and finally MCP.

For ResNet-50, we find that all regularizers have near random accuracy when the hyperparameters are too small and too large. We observe that HALO has the largest range of acceptable values around $10^{-6}$ compared with both $L_1$ and SWS which have very sharp peaks with reasonable accuracy at higher values.

**CIFAR-100 Hyperparameter Experiments:** Figure 6 shows accuracy for different regularization hyperparameters for VGG-like and ResNet architectures on CIFAR-100. For VGG, we find that HALO achieves its highest accuracy at $10^{-7}$ and for smaller values dips to slightly below 80%
indicating that HALO is unable to sparsify strongly enough at smaller values. We also now low accuracy values when the regularization hyperparameter is too high, particularly after $10^{-5}$.

In contrast, with the ResNet architecture, all models have sharp peaks where the accuracy performs reasonably, and for values outside the range, all models perform poorly. Although it appears HALO has a few additional values for which it performs better than $L_1$, SWS and MCP, we note that the range of values with good parameters is still limited and should be a subject of future work.

### 8.3.3 Recovering Accuracy at Lower Sparsity Levels

In Sections 5.2-5.3 we found that that for all models sparsified by HALO, accuracy is always maintained at 0.55 sparsity. For VGG and LeNet architectures accuracy is maintained at 0.85 sparsity and for ResNet architectures, we note that sparsity ratios are much lower at 0.55 and 0.7 as expected given lower reported accuracies in Section 5.3 likely due to the number of parameters in the ResNet architecture and difficulty of the task in comparison with VGG-like and the LeNet architectures for MNIST. We do however still note that performance is better than all other sparsity-inducing methods. We additionally show accuracy at different levels of sparsity (up to the reported sparsity level for maintaining accuracy) in Figure 7. As expected, sparsity ratio and accuracy have an inverse relationship for all methods. We note that for all experiments, the model pruned by HALO typically has the fastest accuracy growth up to the baseline performance. We omit figures for LeNet-300-100, LeNet-5, and VGG-like on CIFAR-10 since nearly all methods are able to retain comparable accuracy at 0.9 or greater sparsity.

Figure 5: Visualization of accuracy of VGG-like and ResNet-50 architectures at 0.95 sparsity across different values of $\xi$ and $\psi$ on CIFAR-10. On the x-axis we plot the regularization coefficients on a log$_{10}$ scale.

Figure 6: Visualization of accuracy of VGG-like and ResNet-50 architectures at 0.95 sparsity across different values of $\xi$ and $\psi$ on CIFAR-100. On the x-axis we plot the regularization coefficients on a log$_{10}$ scale.
8.3.4 Training Time

We summarize training time reported as the number of seconds taken to train a VGG-like network on CIFAR-100 with each penalized training approach in Table 3. We find that regularization approaches are slower for a single epoch, however pruning methods require two or more stages of training and in our experiments require double the number of epochs for training.

| Method | Train Time |
|--------|------------|
| Baseline | 19.07 (±0.24) |
| $L_1$ | 23.98 (±0.54) |
| SWS | 25.90 (±0.37) |
| HALO | 31.63 (±0.23) |

Table 3: Training time for a single epoch of training using the proposed penalty averaged over 5 runs. We additionally report the standard deviation of the runs as well.

8.3.5 Learning Different Types of Sparsity

We further explore the type of sparsity induced in ResNet-50 and VGG-like. In particular, we also prune the networks at the sparsity levels from Table 2 to verify the type of sparsity induced while preserving model performance.

**ResNet-50 on CIFAR-100:** In the main paper, we showed the type of sparsity learned by HALO for the VGG-like network on CIFAR-100 at 0.95 sparsity. We additionally show the same plots for ResNet-50 in Figure 8 which provide insight on the importance of the layers. For ResNet-50, we note that HALO learns a monotonic penalty and performs structured penalization for a majority of the layers. The sharp drops in sparsity correspond to layers around 1, 20 and 38 which have a relatively low number of weights and are down-sampling layers. We note that unlike the VGG-like architecture results, the last layers of ResNet have the largest number of features. This is likely different due to the skip connections aggregating prior outputs which are not present in VGG architectures.

**Different Types of Sparsity at Lower Sparsity Ratios:** Penalizing the networks at a lower level to preserve model performance results in sparsity as shown in Figure 9 and 10. The characteristics of sparsity are similar to those at the higher 0.95 sparsity level, which means that these are properties achievable with minimal drop in performance.

8.3.6 Sparsity During Training

We plot the amount of sparsity in the model during training for both the VGG-like architecture (Figure 11) and ResNet architecture (Figure 12) by counting the number of parameters in the model which are smaller in magnitude than the 95th percentile weight from a fully-trained network trained with the same penalty.
For both architectures, we find that the $L_1$ penalty pushes coefficients to zero almost immediately to achieve 0.95 sparsity early in the training phase. It can also be noted that the 0.95 sparsity threshold is relatively high given the already high amount of sparsity at the start of training indicating that small weights in the model are not shrunk enough.

The HALO penalty also learns a high sparsity ratio for both architectures early in the training phase, however at a slower rate than the $L_1$ penalty. The threshold for the weights in both architectures is also smaller than the threshold for the $L_1$ penalty as seen in the lower sparsity at initial epoch indicating that the weights are further shrunk to 0 over the $L_1$ penalty.

In contrast to the $L_1$ and HALO penalties, on the VGG-like architecture, the SWS penalty penalizes relatively late in the training phase and attains a small threshold, several orders of magnitude smaller than the $L_1$ and HALO penalties indicating it has set a majority of the weights to nearly zero at the end of training. However as evidenced by the accuracy the SWS penalty may shrink the weights too heavily and learn to enforce sparsity too late in the training phase. On the ResNet architecture the sparsity behavior of the SWS penalty is similar to the HALO penalty.

In future work we plan to investigate whether this can lead to faster training of deep neural networks where in the first few epochs all parameters are trained and once a set proportion of the weights can be pruned, only the remaining non-zero weights are trained until convergence.

8.3.7 Sparsity Overlap

We investigate how similar the learned parameters are to one another over multiple runs of HALO and summarizes the results in Figure 13. The sparsity overlap ($SO$) is computed as the Jaccard similarity of zero weights in two models; that is, let $A$ be the set of zero weights for model $A$ and $B$ be the set of weights for model $B$. Then the sparsity overlap for model $A$ and $B$ is defined as $SO = \frac{A \cap B}{A \cup B}$.

The shape of the sparsity overlap in Figure 13 follows that of the structured penalization in Section 8.3.5 which means that the higher the level of sparsity in a layer the higher the percentage
Figure 10: Different types of penalties learned by HALO for ResNet-50 on CIFAR-100 at 0.7 sparsity.

Figure 11: Amount of sparsity for a VGG-like architecture trained on CIFAR-100 with different regularizers.

overlap of sparsity across multiple runs of HALO. This is straightforward for the highly sparse layers. For the less sparse layers, this means that there are multiple winning tickets, which can be expected due the large number of weights.

The drops in SO for the ResNet model correspond to downsampling layers that occur between ResNet blocks.
Figure 12: Amount of sparsity for a ResNet-50 architecture trained on CIFAR-100 with different regularizers.

Figure 13: Sparsity overlap (SO) averaged over four runs for different HALO models compared with one another.