Training Sparse Neural Networks using Compressed Sensing

Anonymous Author(s)
Affiliation
Address
email

Abstract

Pruning the weights of neural networks is an effective and widely-used technique for reducing model size and inference complexity. We develop and test a novel method based on compressed sensing which combines the pruning and training into a single step. Specifically, we utilize an adaptively weighted ℓ^1 penalty on the weights during training, which we combine with a generalization of the regularized dual averaging (RDA) algorithm in order to train sparse neural networks. The adaptive weighting we introduce corresponds to a novel regularizer based on the logarithm of the absolute value of the weights. Numerical experiments on the CIFAR-10 and CIFAR-100 datasets demonstrate that our method 1) trains sparser, more accurate networks than existing state-of-the-art methods; 2) can also be used effectively to obtain structured sparsity; 3) can be used to train sparse networks from scratch, i.e. from a random initialization, as opposed to initializing with a well-trained base model; 4) acts as an effective regularizer, improving generalization accuracy.

1 Introduction

Recently, neural networks have led to a breakthrough in a wide range of fields, especially computer vision and natural language processing [32]. However, neural network models require large amounts of computational power both for training and inference. In addition, trained models often contain an extremely large number of parameters, which require a large amount of memory to store. This makes it difficult or even impossible to employ trained networks in computationally limited environments. Approaches to overcome this problem include quantizing the weights of neural networks [10, 27, 53, 18, 17, 55], designing compressed achitectures which are computationally less expensive yet achieve comparable accuracy [24, 50, 28], and network pruning, which involves removing weights from a network as part of the training process [18, 17, 19].

We focus on the problem of network pruning, for which a very popular and effective approach is the iterative pruning and retraining method [19]. This method works by pruning small weights after training, then finetuning or retraining the resulting model and repeating this process. There are also methods which combine the pruning and training into a single step [13, 2, 1, 14]. While existing methods achieve impressive results, compressing networks by over an order of magnitude with minimal loss in accuracy, we demonstrate that by utilizing compressed sensing [15] significantly smaller and more accurate networks can be obtained.

Compressed sensing is concerned with the recovery of a sparse signal from a limited number of measurements. Importantly, the support of the signal is not known and must be inferred from the measurements. It is well-known that an ℓ^1-norm regularizer can be used to recover the signal under a variety of assumptions [51, 6, 7]. This suggests using an ℓ^1-norm regularizer on the weights of a neural network to train sparse models. Indeed, the ℓ^1-norm has been used successfully in neural

Submitted to 34th Conference on Neural Information Processing Systems (NeurIPS 2020). Do not distribute.

network training, although an ℓ^2-norm penalty combined with pruning and retraining often performs better for obtaining sparse networks[19]. In addition, when seeking structured sparsity, the closely related group lasso regularizer [56] is often used effectively [52].

In the field of compressed sensing, other regularizers have been shown to reconstruct sparser signals than the ℓ^1-norm. For instance, an adaptively weighted ℓ^1-norm [8] and the ℓ^p-norm with $0 \leq p < 1$[9]. These are also related, since the ℓ^p-norm can be implemented via an iteratively reweighted ℓ^1-norm penalty, although an iteratively reweighted ℓ^2-norm approach is computationally simpler [9]. However, such techniques have not, to the best of our knowledge, been applied to the training of deep neural networks yet. In this work, we seek to fill this gap.

In particular, we design an algorithm for training sparse neural networks which is based on compressed sensing, specifically on an adaptively weighted ℓ^1-norm penalty. There are two major problems we overcome in doing this. First, we design an appropriate adaptive ℓ^1 weighting scheme for neural network training. Second, we design an algorithm which generates sparse iterates during training and also achieves good generalization accuracy. For convex objectives, the regularized dual averaging (RDA) algorithm [54] was designed for this purpose, and we generalize it to the setting of deep learning. These contributions are summarized as follows.

- We design an adaptively weighted ℓ^1-regularization scheme which works well for training neural networks. We connect this regularization scheme with a novel logarithmic regularizer.
- In order to use this scheme to train sparse neural networks, we propose the use of a modified version of the regularized dual averaging (RDA). Importantly, this modified RDA method generates sparse iterates while also maintaining good generalization accuracy.
- To further improve the generalization accuracy, we incorporate momentum into the RDA method in a novel way, which also improves the robustness of the sparse training algorithm.
- We demonstrate experimentally, on a variety of datasets and architectures, that our method trains networks which generalize better and are significantly sparser than existing state-of-the-art methods.

2 Related Work

It is widely accepted that highly overparameterized neural networks are easier to train and achieve better generalization accuracy than smaller models [3, 23, 57]. However, large models are also computationally expensive to deploy. This motivates the idea of network pruning, where most of the weights and complexity of the network are removed during training. This idea was first introduced in [34, 20] and shown to be highly effective in [19]. Since then, a variety of approaches to training sparse neural networks have been proposed.

2.1 Unstructured pruning

Many successful approaches to network pruning follow the pruning and retraining methodology first successfully used in [19, 18]. In this approach, weights with small magnitude are removed after training and the remaining weights are finetuned. This process is often repeated multiple times to compress networks with no loss in generalization accuracy. A variety of heuristics for pruning have been introduced, for instance based on neuron outputs instead of weight magnitudes [25], redundancy [40, 48], and second derivatives [14]. The difference between fine-tuning the remaining unpruned weights, retraining them from scratch, and retraining them from their initial values is studied in [16, 37, 59], where specifically the "Lottery Ticket Hypothesis" is explored. Also, in [18], it is shown that pruning can be effectively combined with weight quantization to further improve network compression.

In addition, many methods which combine pruning and training have been proposed. In particular, it has been shown that variational dropout [41] can be used to train exceptionally sparse networks. An ℓ^1-norm penalty on the weights combined with the RDA algorithm [54] is applied to neural networks in [21]. In [13] a modified momentum method is used to train sparse networks, which is particularly effective even for deep architectures. The use of momentum to obtain sparsity is also considered in [12]. This paper builds upon these approaches by using an adaptively weighted ℓ^1-norm regularizer to combine the pruning and training steps. An important difference between our method and most

previous methods which combine training and pruning is that our algorithm trains the network from scratch, i.e. from a random initialization, as opposed to initializing with a well-trained base model.

2.2 Structured pruning

To improve the efficiency of inference and to adapt the network to specific hardware, it is often desirable to obtain structured sparsity. Two common approaches are kernel sparsity and channel sparsity, where whole kernels or channels are pruned from convolutional layers [35, 36, 37, 31, 58, 26]. A common method for obtaining structured sparsity is to use the group lasso [56, 46] as a regularization term during training. While our method is designed for unstructured sparsity, we apply an appropriate modification to the problem of obtaining kernel sparsity, showing that it can be used to obtain structured sparsity as well.

3 Applying compressed sensing to neural network training

We begin by introducing some notation. The neural networks which we consider consist of a sequence of convolutional, linear, and batch normalization [29] layers. Each of the convolutional and linear layers contain trainable weights W and bias parameters b. In addition, the batch normalization layers contain trainable shift and scale parameters γ and β. We group the weights of the network by layer and also separate weights W from biases b, and shifts β from scale parameters γ in convolutional, linear and batch normalization layers. We denote the resulting groups of parameters $G_1, ..., G_N$, where each group G_i is either W or b from a convolutional of linear layer, or is γ or β from a batch normalization layer. Here N is the total number of groups, which will be twice the number of layers. We also let $\Theta = \{G_i\}_{i=1}^N$ denote the collection of all parameters, $\mathcal{D}$ denote the training dataset, and

$$L(\Theta) = \frac{1}{|\mathcal{D}|} \sum_{(x,y)\in\mathcal{D}} l(x, y, \Theta) \tag{1}$$

denote the empirical loss function, where l is typically taken to be the cross-entropy. Finally, we denote by $\tilde{\nabla} L(\Theta)$ a stochastic gradient sample obtained by considering a mini-batch of training data.

3.1 Adaptively weighting ℓ^1-norm

The lasso [51], which involves adding an ℓ^1-norm regularization to the regression loss function, is a well-known and effective method for performing sparse regression and signal estimation in compressed sensing. In the context of neural network training, this corresponds to solving

$$\arg\min_{\Theta} L(\Theta) + \lambda \|\Theta\|_1, \tag{2}$$

where λ is a hyperparameter controlling the trade-off between sparsity and training loss.

While the lasso regularizer can be effectively applied to neural network training, it can be considerably improved by using an adaptive ℓ^1 weight. An adaptively reweighted ℓ^1-constraint was first introduced in [8], where it is shown to be considerably better at recovering sparse signals than the Danzig selector [5]. In this method, the ℓ^1-norm on a parameter $\theta \in \Theta$ is weighted by $|\tilde{\theta}|^{-1}$, where $\tilde{\theta}$ is an estimate of the true value of θ obtained in the previous iteration.

We use a similar adaptively weighted ℓ^1-norm on the weights of our neural networks. Our method differs in the observation that parameters belonging to the same group G_i should be of a similar scale, while parameters belonging to different G_i can have very different scales. This suggests using a scheme similar to that introduced in [8] within each group G_i. Specifically, we weight the ℓ^1-norm on a parameter $\theta \in G_i$ with

$$\lambda(\beta + 1)\left(\beta + \frac{|\theta|}{M_i}\right)^{-1}, \tag{3}$$

where β and λ are hyperparameters and M_i is the maximum absolute value of all parameters in G_i, i.e. $M_i = \max_{\theta\in G_i} |\theta|$. This weight is λ for the entry with largest absolute value in each group and increases to a maximum of $\lambda(1 + \beta^{-1})$ for entries which are 0. So the hyperparameter λ controls the total strength of the ℓ^1 penalty, while β controls how large the penalty can become for small weights.

The precise form of the weighting scheme 3 was determined through trial and error by running experiments on the LeNet-5 model on MNIST [33]. However, it can also be related to a novel regularizer, via an argument given in [9]. There, an ℓ^p-penalty for $0 \leq p < 1$ is implemented using an adaptively weighted ℓ^1-penalty, with appropriately weights. Following this viewpoint, the weighting scheme in 3 can be understood as corresponding to the logarithmic regularizer

$$R(\Theta) = \lambda(\beta+1)\sum_{i=1}^{N}\sum_{\theta\in G_i} M_i \log\left(\beta + \frac{|\theta|}{M_i}\right), \tag{4}$$

where $x = G_1 \cup \cdots \cup G_n$ represents all of the weights in the network. This is very similar to the case of an ℓ^0 penalty in [9].

Finally, in order to make the algorithm more robust, we calculate the weights in (3) based upon a running average of the magnitudes of the parameters. In particular, we consider a running average of the absolute values of each parameter, computed recursively according to

$$|\theta|_n^{av} = \mu|\theta|_{n-1}^{av} + (1-\mu)|\theta_n|. \tag{5}$$

Here μ is a momentum parameter which effectively controls the number of iterations over which we average, and we discuss it in more detail later. We use the averaged absolute values $|\theta|_n^{av}$ in (3) to calculate the adaptive weights in each iteration.

3.2 Extended Regularized Dual Averaging

A key difficulty in using an ℓ^1-norm regularizer for neural network training is that the naive forward-backward stochastic gradient descent algorithm applied to the regularized loss function

$$f(\Theta) = L(\Theta) + \lambda\|\Theta\|_1, \tag{6}$$

which takes the form

$$\begin{aligned} \Theta_{n+\frac{1}{2}} &= \Theta_n - s_n\tilde{\nabla}L(\Theta_n) \\ \Theta_{n+1} &\in \Theta_{n+\frac{1}{2}} - s_n\partial(\lambda\|\cdot\|_1)(\Theta_{n+1}), \end{aligned} \tag{7}$$

doesn't generate sparse iterates. Here the second line represents a backward step for the regularizer $\lambda\|\cdot\|_1$, whose solution can be given in closed form and is known as the soft-thresholding operator

$$x_{n+1} = \arg\min_y \left(\lambda\|y\|_1 + \frac{1}{2}\|x_{n+\frac{1}{2}} - y\|_2^2\right) = \text{sign}(x_{n+\frac{1}{2}})\max(0, |x_{n+\frac{1}{2}}| - s_n\lambda). \tag{8}$$

The fact that iteration (7) doesn't generate sparse iterates, which was first observed in [54], occurs because a small step size s_n is often necessary for convergence, or to obtain good generalization accuracy, and this means that the soft-thresholding parameter $s_n\lambda$ is very small. This phenomenon has also been observed in the context of training compressed neural networks, where it has been termed magnitude plateau [13].

To overcome this issue for convex machine learning problems, the regularized dual averaging algorithm (RDA), which is an extension of the dual averaging algorithm of Nesterov [43], is introduced in [54]. A special case of this algorithm, which bears resemblance to (7), is

$$\begin{aligned} \Theta_{n+\frac{1}{2}} &= \Theta_{n-\frac{1}{2}} - s_n\tilde{\nabla}L(\Theta_n) \\ \Theta_{n+1} &\in \Theta_{n+\frac{1}{2}} - S_n\partial(\lambda\|\cdot\|_1)(\Theta_{n+1}), \end{aligned} \tag{9}$$

where the backward step is given by $S_n = \sum_{k=1}^{n} s_i$ and $\Theta_{-\frac{1}{2}} = \Theta_0$. Note that this implies that the soft-thresholding parameter, λS_n, grows significantly and this leads to sparse iterates. Extensive experiments showing the effectiveness of this method on convex problems are given in [54].

The RDA algorithm has previously been applied to the training of neural networks in [21]. However, when applied to highly non-convex problems such as neural network training, the RDA algorithm is not particularly robust and may not always converge. This is due to the fact that the soft-thresholding parameter increases too rapidly and this makes the algorithm highly sensitive to the initialization. To get around this issue, we use a generalization of the RDA algorithm, first proposed in [45] and called

xRDA (eXtended RDA). This algorithm introduces an additional averaging parameter α_n and takes the form

$$\begin{aligned} \Theta_{n+\frac{1}{2}} &= (1-\alpha_n)\Theta_n + \alpha_n \Theta_{n-\frac{1}{2}} - s_n \tilde{\nabla} L(\Theta_n) \\ \Theta_{n+1} &\in \Theta_{n+\frac{1}{2}} - S_n \partial(\lambda \|\cdot\|_1)(\Theta_{n+1}), \end{aligned} \tag{10}$$

where the backward step size now satisfies $S_n = \alpha_n S_{n-1} + s_n$ and again $\Theta_{-\frac{1}{2}} = \Theta_0$. Notice that if $\alpha_n = 0$ we get SGD (7), and if $\alpha_n = 1$ we get RDA (9). Thus this method can be thought of as interpolating between the two. In [45] it is shown that this method converges for convex objectives. By slowly increasing α_n during the course of training, the soft-thresholding parameter slowly increases, improving the robustness of the algorithm and leading to better accuracy and sparsity.

3.3 Momentum for RDA

Finally, in order to improve the generalization accuracy of the trained sparse models, we introduce a method of adding momentum to RDA. Momentum, which has an intuitive physical interpretation [44], has long been used to accelerate convex optimization [42] and has been used extensively in deep learning to accelerate convergence and improve generalization accuracy [49]. In the context of deep learning, training with momentum typically means the search direction in each iteration is an exponentially weighted average of the gradient over the previous iterations, i.e.

$$\begin{aligned} v_n &= \mu v_{n-1} + \tilde{\nabla} L(\Theta_n) \\ \Theta_{n+1} &= \Theta_n - s_n v_n, \end{aligned} \tag{11}$$

where the momentum parameter μ effectively determines how many previous iterates the gradient is averaged over. We note that in the context of convex optimization, the above iteration must be modified to guaratee convergence for arbitrary convex objectives [42]. A detailed investigation of (11) and Nesterov's momentum for training deep neural networks is given in [49].

We incorporate momentum in the xRDA algorithm (10) by replacing the sampled gradient $\tilde{\nabla} L(\Theta)$ by an average over the past gradients. This results in the algorithm

$$\begin{aligned} v_n &= \mu v_{n-1} + (1-\mu)\tilde{\nabla} L(\Theta_n) \\ \Theta_{n+\frac{1}{2}} &= (1-\alpha_n)\Theta_n + \alpha_n \Theta_{n-\frac{1}{2}} - s_n \tilde{v}_n \\ \Theta_{n+1} &\in \Theta_{n+\frac{1}{2}} - S_n \partial(\lambda \|\cdot\|_1)(\Theta_{n+1}), \end{aligned} \tag{12}$$

where as before the backward step size now satisfies $S_n = \alpha_n S_{n-1} + s_n$ and again $\Theta_{-\frac{1}{2}} = \Theta_0$.

The momentum parameter μ in (12) controls the scale over which we average our gradients to determine an appropriate search direction. This parameter is often taken independent of the step size, which corresponds to averaging the gradients over a fixed number of iterations. However, we have observed that better generalization accuracy is obtained if the gradients are averaged over a fixed timescale. In particular, we take the momentum parameter $\mu = e^{-s_n/T}$, where T determines the time scale over which we average. We note that this dependence of momentum on step size corresponds to a discretization of Langevin dynamics with constant damping parameter [4].

We also use this value of μ to determine the average parameter magnitudes $|\theta|_n^{av}$. In other words, we set $m = e^{-s_n/T}$ in equation (5) so that the parameter magnitudes are averaged over a fixed time scale when calculating the adaptive ℓ^1 weights. We have found that this significantly improves the stability and robustness of the algorithm.

3.4 Structured Sparsity

In this section, we discuss how the above method can be modified to obtain structured sparsity. In particular, the example we use to demonstrate this is to train networks with kernel sparsity [35]. To achieve this, we propose modifying the adaptive ℓ^1-weights in (3) to be constant within each kernel. Specifically, let $K \in \mathbb{R}^{h \times w \times r \times s}$ denote a the weights in a convolutional layer, i.e. some of the groups G_i will take this form. Here h and w denote the height and width of the convolutional kernel, while r and s denote the number of input and output channels, respectively. The weights K consist

of rs convolutional kernels $k_j \in \mathbb{R}^{hw}$, $j = 1, ..., rs$. Now we weight the ℓ^1-norm on a parameter $\theta \in k_j$ by

$$\lambda(\beta + 1)\left(\beta + \frac{|k_j|_1}{M_K}\right)^{-1}, \tag{13}$$

where M_K is the maximum is taken over all kernels in the K, i.e. $M_K = \max_j |k_j|_1$. Note that all parameters in each kernel are given the same ℓ^1-weight. The reasoning behind this is that we wish to remove the entire kernel as a group, rather than each individual parameter. We only make this change in parameter groups G_i which represent convolutional weights. Analyzing this weighting scheme along the same lines as in [9], we see that it corrresponds to the regularizer

$$R(\Theta) = \lambda(\beta + 1)\sum_{i=1}^{N} R_{G_i}, \tag{14}$$

where $R(G_i)$ is equal to

$$\sum_{j=1}^{rs} M_K \log\left(\beta + \frac{|k_j|_1}{M_K}\right) \tag{15}$$

if G_i consists of the kernel of a convolutional layer, and is the same as in (4) otherwise.

4 Experimental Results

4.1 Unstructured sparsity results and comparison

We evaluate the effectiveness of using compressed sensing to train sparse neural networks on several common benchmark models and datasets. Specifically, we consider the datasets CIFAR-10 and CIFAR-100 [30]. In all of our tests, we run our algorithm from scratch, i.e. we do not need to initialize with a well-trained base model, unlike previous approaches which combine training and pruning into a single step [13, 41].

4.1.1 CIFAR-10

We present the results of our first set of experiments on CIFAR-10 in Table 1. We test a variety of architectures, specifically VGG-16 and VGG-19 [47], and ResNet-18 and ResNet-56 [22]. We note that for the ResNet-18 structure we use is much wider that the ResNet-56 network. In particular, the ResNet-56 architecture we use contains 3 blocks with $[16, 32, 64]$ channels, while the ResNet-18 architecture contains 4 blocks with $[64, 128, 256]$ channels. We use these two architectures to show that compressed sensing can be applied both to deep narrow and to shallow wide residual networks.

For VGG-16 and VGG-19 we set the momentum timescale $T = 9.5$ and use a cosine learning rate schedule [38] with initial learning rate 1.0. We determine the averaging parameter α_n in a similar fashion, increasing from 0 to 1 according to the same cosine dependence. Finally, we set the ℓ^1-norm parameters $\lambda = 10^{-6}$ and $\beta = 2 \cdot 10^{-3}$. For ResNet-16 and ResNet-18 we use mostly the same hyperparameters, but we set the initial learning rate to 0.25 and set $\lambda = 1.3 \cdot 10^{-6}$. We run the algorithm for 600 epochs in all cases. This follows the methodology in [13], and we have found that this large number of epochs is necessary to give the algorithm time to find advantageous sparse configurations. The baseline accuracy is obtained by training with SGD using appropriately tuned hyperparameters for the same number of epochs. For all models, we use a standard data augmentation strategy, consisting of padding to 40 by 40, followed by a random crop back to 32 by 32, and random right-left flipping.

We can see from these results that using compressed sensing (CS) results in significantly sparser and more accurate networks than existing state-of-the-art methods. For instance, we are able to reduce the number of parameters in VGG-19 by more than two orders of magnitude while increasing the accuracy. In addition, this holds across a wide range of architectures, including both deep and wide residual neural networks.

4.1.2 CIFAR-100

Next, we present results on the CIFAR-100 dataset in Table 2. Here we again consider the VGG-19 network architecture. We used the same hyperparameters as for VGG-19 on CIFAR-10 and the same data augmentation strategy, discussed previously.

We see from these results that compressed sensing (CS) is also effective on more complicated datasets. In particular, we are able to reduce the size of VGG-19 by a factor of nearly 40 while attaining a significant increase in accuracy. This suggests that compressed sensing is an effective regularizer in addition to producing sparse networks.

Table 1: Unstructured sparsity results on CIFAR-10.

Model	Algorithm	Base Top1	Sparse Top1	Dense / Sparse Parameters	Compression Ratio	Non-Zero Fraction
ResNet-18	**CS**	**95.05**	**94.73**	**11.17M / 0.17M**	**65x**	**1.52**
ResNet-18	RDA [21]	-	93.95	11.17M / 0.56M	20x	5.00
ResNet-56	**CS**	**93.52**	**94.12**	**853K / 98K**	**8.7x**	**11.46**
ResNet-56	GSM [13]	94.05	94.10	853K / 128K	6.6x	15.00
ResNet-56	GSM [13]	94.05	93.80	853K / 85K	10x	10.00
VGG-16	**CS**	**93.79**	**94.16**	**14.73M / 0.19M**	**77x**	**1.29**
VGG-16	Momentum [12]	93.41	93.31	14.73M / 0.74M	20x	5.00
VGG-16	Bayesian [39]	91.60	91.00	14.73M / 0.81M	18x	5.50
VGG-16	Var Dropout [41]	92.70	92.70	14.73M / 0.31M	48x	2.08
VGG-16	Slimming [36]	93.66	93.41	14.73M / 0.65M	22x	4.40
VGG-19	**CS**	**93.60**	**94.00**	**20.04M / 0.18M**	**108x**	**0.92**
VGG-19	Han [19]	93.50	93.34	20.04M / 1.00M	20x	5.00
VGG-19	Scratch-B [37]	93.50	93.63	20.04M / 1.00M	20x	5.00

Table 2: Unstructured sparsity results on CIFAR-100.

Model	Algorithm	Base Top1	Sparse Top1	Dense / Sparse Parameters	Compression Ratio	Non-Zero Fraction
VGG-19	**CS**	**73.83**	**75.23**	**20.09M / 0.54M**	**37x**	**2.67**
VGG-19	Han [19]	71.70	70.22	20.09M / 1.00M	20x	5.00
VGG-19	Scratch-B [37]	71.70	72.08	20.09M / 1.00M	20x	5.00

4.2 Structured Sparse Network Training

Finally, we present results demonstrating the effectiveness of compressed sensing to obtain structured sparsity in Table 3. We ran our experiments on CIFAR-10 with VGG-16 and used the same hyperparameters as for the unstructured pruning experiments. The only change was to use the structured ℓ^1 weighting scheme (13).

From these results, we see that compressed sensing can reduce the number of non-zero kernels by a factor of nearly 50, while only suffering from a small drop in validation accuracy. This is significantly better than the unstructured pruned model whose results are listed in Table 1, which only completely removed about 95% of the kernels. It is also significantly better than previous approaches to obtaining kernel sparsity, and demonstrates that our compressed sensing can be used effectively to obtain structured sparsity as well.

Table 3: Results for structured (kernel) pruning on CIFAR-10.

Model	Algorithm	Base Top1	Sparse Top1	Dense / Sparse Parameters	Compression Ratio	Non-Zero Fraction
VGG-16	**CS**	**93.79**	**93.80**	**14.73M / 0.29M**	**52x**	**1.94**
VGG-16	Filter pruning [35]	93.25	93.40	14.73M / 5.30M	3x	36.00
VGG-16	Scratch-B [37]	93.63	93.78	14.73M / 5.30M	3x	36.00

5 Conclusion and future work

We have shown that compressed sensing can be used to effectively train sparse neural networks. In particular, by developing an appropriate adaptively weighted ℓ^1-norm penalty, combined with a novel optimization algorithm based on the RDA algorithm, we are able to train significantly smaller and sparser networks than previously possible on both CIFAR-10 and CIFAR-100. A significant increase in generalization accuracy also suggests that compressed sensing has an important regularizing effect. In addition, we show that structured sparsity can also be effectively obtained in this fashion.

Further work involves more extensive testing and tuning of the algorithm, in addition to a theoretical analysis of compressed neural networks. In particular, we propose testing this technique on larger and more complex datasets such as ImageNet [11], and attempting to obtain better results with more extensive hyperparameter tuning. We also propose testing different types of structured sparsity using the method, and expect that this technique will also be effective for different types of machine learning models beyond feed-forward convolutional neural networks. Finally, we hope that a theoretical analysis can explain the precise nature of the regularizing effect of compressed sensing observed in our experiments.

6 Broader impact

The ability to train compressed neural networks promises to enable the deployment of trained models for inference in a wider variety of contexts. This includes in computationally limited environments such as mobile phones. The effects of such a wider deployment include both positives such as greater accessibility of deep learning based applications, and negatives such as the potential for misuse or mistakes to have a greater impact.

In addition, we expect that understanding the regularization effect that compressed sensing has will shed light on how and what convolutional neural networks learn. This in turn will aid the development of more robust models and machine learning methods. The resulting development of more reliable and robust models will expand the scope of potential applications to domains where the precise behavior of networks much be predictable.

References

[1] Jose M Alvarez and Mathieu Salzmann. Learning the number of neurons in deep networks. In *Advances in Neural Information Processing Systems*, pages 2270–2278, 2016.

[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] Yoshua Bengio, Nicolas L Roux, Pascal Vincent, Olivier Delalleau, and Patrice Marcotte. Convex neural networks. In *Advances in neural information processing systems*, pages 123–130, 2006.

[4] Rabi N Bhattacharya and Edward C Waymire. *Stochastic processes with applications*, volume 61. Siam, 2009.

[5] Emmanuel Candes, Terence Tao, et al. The dantzig selector: Statistical estimation when p is much larger than n. *The annals of Statistics*, 35(6):2313–2351, 2007.

[6] Emmanuel J Candès, Justin Romberg, and Terence Tao. Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information. *IEEE Transactions on information theory*, 52(2):489–509, 2006.

[7] Emmanuel J Candès, Justin K Romberg, and Terence Tao. Stable signal recovery from incomplete and inaccurate measurements. *Communications on Pure and Applied Mathematics: A Journal Issued by the Courant Institute of Mathematical Sciences*, 59(8):1207–1223, 2006.

[8] Emmanuel J Candès, Michael B Wakin, and Stephen P Boyd. Enhancing sparsity by reweighted ℓ^1 minimization. *Journal of Fourier analysis and applications*, 14(5-6):877–905, 2008.

[9] Rick Chartrand and Wotao Yin. Iteratively reweighted algorithms for compressive sensing. In *2008 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 3869–3872. IEEE, 2008.

[10] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*, pages 3123–3131, 2015.

[11] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[12] Tim Dettmers and Luke Zettlemoyer. Sparse networks from scratch: Faster training without losing performance. *arXiv preprint arXiv:1907.04840*, 2019.

[13] Xiaohan Ding, Xiangxin Zhou, Yuchen Guo, Jungong Han, Ji Liu, et al. Global sparse momentum sgd for pruning very deep neural networks. In *Advances in Neural Information Processing Systems*, pages 6379–6391, 2019.

[14] Xin Dong, Shangyu Chen, and Sinno Pan. Learning to prune deep neural networks via layer-wise optimal brain surgeon. In *Advances in Neural Information Processing Systems*, pages 4857–4867, 2017.

[15] David L Donoho. Compressed sensing. *IEEE Transactions on information theory*, 52(4):1289–1306, 2006.

[16] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

[17] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. Eie: efficient inference engine on compressed deep neural network. *ACM SIGARCH Computer Architecture News*, 44(3):243–254, 2016.

[18] 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.

[19] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143, 2015.

[20] Babak Hassibi and David G Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in neural information processing systems*, pages 164–171, 1993.

[21] Juncai He, Xiaodong Jia, Jinchao Xu, Lian Zhang, and Liang Zhao. Make ℓ_1 regularization effective in training sparse cnn. *arXiv*, pages arXiv–1807, 2018.

[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[23] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[24] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[25] Hengyuan Hu, Rui Peng, Yu-Wing Tai, and Chi-Keung Tang. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint arXiv:1607.03250*, 2016.

[26] Zehao Huang and Naiyan Wang. Data-driven sparse structure selection for deep neural networks. In *Proceedings of the European conference on computer vision (ECCV)*, pages 304–320, 2018.

[27] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research*, 18(1):6869–6898, 2017.

[28] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[29] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[30] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[31] Vadim Lebedev and Victor Lempitsky. Fast convnets using group-wise brain damage. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2554–2564, 2016.

[32] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[33] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

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

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

[36] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2736–2744, 2017.

[37] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270*, 2018.

[38] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

[39] Christos Louizos, Karen Ullrich, and Max Welling. Bayesian compression for deep learning. In *Advances in Neural Information Processing Systems*, pages 3288–3298, 2017.

[40] Zelda Mariet and Suvrit Sra. Diversity networks: Neural network compression using determinantal point processes. *arXiv preprint arXiv:1511.05077*, 2015.

[41] 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.

[42] Yurii Nesterov. A method for unconstrained convex minimization problem with the rate of convergence o (1/kˆ 2). In *Doklady an ussr*, volume 269, pages 543–547, 1983.

[43] Yurii Nesterov. Primal-dual subgradient methods for convex problems. *Mathematical programming*, 120(1):221–259, 2009.

[44] Boris Teodorovich Polyak. Gradient methods for minimizing functionals. *Zhurnal Vychislitel'noi Matematiki i Matematicheskoi Fiziki*, 3(4):643–653, 1963.

[45] Jonathan W. Siegel and Jinchao Xu. Extended regularized dual averaging. 2019.

[46] Noah Simon, Jerome Friedman, Trevor Hastie, and Robert Tibshirani. A sparse-group lasso. *Journal of computational and graphical statistics*, 22(2):231–245, 2013.

[47] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[48] Suraj Srinivas and R Venkatesh Babu. Data-free parameter pruning for deep neural networks. *arXiv preprint arXiv:1507.06149*, 2015.

[49] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147, 2013.

[50] Mingxing Tan and Quoc V Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019.

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

[52] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *Advances in neural information processing systems*, pages 2074–2082, 2016.

[53] Shuang Wu, Guoqi Li, Feng Chen, and Luping Shi. Training and inference with integers in deep neural networks. *arXiv preprint arXiv:1802.04680*, 2018.

[54] Lin Xiao. Dual averaging methods for regularized stochastic learning and online optimization. *Journal of Machine Learning Research*, 11(Oct):2543–2596, 2010.

[55] Penghang Yin, Shuai Zhang, Jiancheng Lyu, Stanley Osher, Yingyong Qi, and Jack Xin. Blended coarse gradient descent for full quantization of deep neural networks. *Research in the Mathematical Sciences*, 6(1):14, 2019.

[56] Ming Yuan and Yi Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(1):49–67, 2006.

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

[58] Hao Zhou, Jose M Alvarez, and Fatih Porikli. Less is more: Towards compact cnns. In *European Conference on Computer Vision*, pages 662–677. Springer, 2016.

[59] Hattie Zhou, Janice Lan, Rosanne Liu, and Jason Yosinski. Deconstructing lottery tickets: Zeros, signs, and the supermask. In *Advances in Neural Information Processing Systems*, pages 3592–3602, 2019.