RE-WEIGHTED LEARNING FOR SPARSIFYING DEEP NEURAL NETWORKS

Igor Fedorov *, Bhaskar D. Rao

Department of Electrical and Computer Engineering
University of California, San-Diego

ABSTRACT
This paper addresses the topic of sparsifying deep neural networks (DNN’s). While DNN’s are powerful models that achieve state-of-the-art performance on a large number of tasks, the large number of model parameters poses serious storage and computational challenges. To combat these difficulties, a growing line of work focuses on pruning network weights without sacrificing performance. We propose a general affine scaling transformation (AST) algorithm to sparsify DNN’s. Our approach follows in the footsteps of popular sparse recovery techniques, which have yet to be explored in the context of DNN’s. We describe a principled framework for transforming densely connected DNN’s into sparsely connected ones without sacrificing network performance. Unlike existing methods, our approach is able to learn sparse connections at each layer simultaneously, and achieves comparable pruning results on the architecture tested.

Index Terms— Sparsity, deep learning, affine scaling

1. INTRODUCTION

Deep neural networks (DNN’s) have become popular in a large number of fields due to their flexibility, simple learning procedure, and performance [1]. At a high level, DNN’s learn a mapping from a set of inputs to a set of desired outputs. More formally, let $D = \{x_i, y_i\}_{i=1}^N$ be a dataset consisting of input and target output pairs. The DNN learning problem can be stated as

$$\arg \min_{\theta} f(\theta, D)$$

(1)

where $\theta = \{W_k, b_k\}_{k=1}^K$ is the set of weights and biases, respectively, which parametrize each of the $K$ network layers and $f(\cdot, \cdot)$ is an application dependent objective function. In the following, we will omit the dependence of $f(\cdot, \cdot)$ on $D$ for brevity. Due to space limitations, we omit further background and details on DNN’s and refer the reader to [2].

As the number of parameters in the network grows, the complexity of the learned mapping grows with it. In fact, it has been shown that a DNN with a single hidden layer and finite number of neurons can approximate any measurable function arbitrarily well [3]. From a practical point of view, performing inference with a large DNN presents various challenges, including excessive power consumption and memory requirements [4]. As such, a growing trend in the DNN research community has been to try to prune trained models, i.e. throw away some network parameters without harming performance. Work on DNN pruning goes back at least several decades, with early papers focusing on identifying network weights which have small influence on the objective function as measured by the Hessian of $f(\cdot)$ (or its approximation) [5, 6]. A recent Hessian-based technique extends [5] by ensuring that the difference in network output at each layer of the original and pruned models is bounded [7].

Other works have shown that the magnitude of a network weight can be a viable measure of its importance. The general paradigm is to undertake an iterative search where, given an estimate of the network parameters at iteration $t - 1$, $\theta^{t-1}$, to alternate between

$$\theta^{(t-0.5)} = \arg \min_{\theta: f(\theta)=f(\theta^{(t-1)})} ||\theta||_0$$

(Pruning)

and

$$\theta^t = \arg \min_{\theta: ||\theta||_0=||\theta^{(t-0.5)}||_0} f(\theta)$$

(Retraining)

where $||\theta||_0$ denotes the number of non-zero elements in $\{W_k\}_{k=1}^K$. For instance, the Learning both Weights and Connections (LWC) algorithm performs (Pruning) by setting small weights to 0 [4]. The issue with LWC is that if a parameter is mistakenly pruned in the (Pruning) step, that weight will never be spliced in future iterations [8]. To remedy this shortcoming, the Dynamic Network Surgery (DNS) algorithm proposes to replace $\theta$ in (Pruning)-(Retraining) with $q \odot \omega^{t-1}$, where $\odot$ denotes element-wise multiplication, the elements of $q$ represent the value of the corresponding elements of $\theta$, and the elements of $\omega^{t-1}$ denote whether the corresponding element of $\theta$ should be pruned at iteration $t$ or

---

* Igor Fedorov was partially supported by the San Diego Chapter of the ARCS Foundation, Inc.

1 For the result to hold, the non-linearity must be a squashing function [3].

2 Splicing refers to re-introducing a pruned parameter [8].
Consider the regularized DNN learning problem

\[ \arg \min_{\theta} f(\theta) + \lambda \sum_{j=1}^{J} g(\theta_j) \]  

where \( g(\cdot) \) is a sparsity promoting regularizer and \( J \) denotes the number of elements of \( \theta \). While [4] provides the benefit of learning sparse \( \theta \), the trade-off is that the solution of [4] may not necessarily be a solution to [1]. Suppose, further, that \( g(\cdot) \) is a concave function. It can be shown that the objective in [4] is non-increasing under the update rule

\[ \arg \min_{\theta} f(\theta) + \lambda \sum_{j=1}^{J} \psi_j^{-1} \]  

where \( \psi_j^{-1} = \nabla g(\theta_j^{-1}) \). Methods like [5] are collectively known as majorization-minimization (MM) algorithms [9]. Let \( q = \theta \odot \psi_j^{-1} \) and \( \lambda = 0 \), then [5] becomes

\[ \arg \min_{q} f(q \odot \omega_j^{-1}) \]  

where \( \omega_j = (\psi_j)^{-1} \). Let \( q' \) by the solution of [6]. The proposed approach proceeds in an iterative fashion, where each iteration consists of finding \( q' \) and computing

\[ \theta' = q' \odot \omega_j^{-1}. \]  

Unlike LWC and DNS, our learning procedure is global, i.e. all of the network weights are updated at each iteration.

### 1.1. Contribution

While the subject of sparsity has only recently gained traction in the DNN community, a considerable amount of literature dedicated to sparse solutions of linear systems already exists in the signal processing community. The purpose of this paper is to begin to bridge the gap between the two fields and show that a popular class of sparse signal recovery (SSR) techniques can be transferred to the task of DNN pruning. In the following, we propose a general affine scaling transformation (AST) algorithm for sparsifying DNN’s. Unlike LWC and DNS, which perform pruning layer by layer, our approach learns sparse connections at all layers simultaneously. In some sense, this makes the proposed approach less greedy and allows it greater flexibility in exploring the search space. We will show that this framework is general, gives rise to many effective approaches, and is related to the state-of-the-art DNS algorithm.

### 2. PROPOSED FRAMEWORK

Consider the regularized DNN learning problem

\[ \arg \min_{\theta} f(\theta) + \lambda \sum_{j=1}^{J} g(\theta_j) \]  

where \( g(\cdot) \) is a sparsity promoting regularizer and \( J \) denotes the number of elements of \( \theta \). While [4] provides the benefit of learning sparse \( \theta \), the trade-off is that the solution of [4] may not necessarily be a solution to [1]. Suppose, further, that \( g(\cdot) \) is a concave function. It can be shown that the objective in [4]

3In this work, as in [8, 11], we are interested in pruning the network weights only, i.e. not the biases.

4In [6], each layer \( k \) has its own \((a_k, b_k)\), but we have omitted this detail in [2] for brevity.

5By re-centering, we mean positioning the unknowns in the middle of the search space, such as \( q_j = 1 \psi_j \).
2.2. Special Cases

To illustrate how broad the proposed framework is, we proceed by showing the many forms which \( \sigma \) can take for various choices of \( g(\theta_j) \) used in the SSR literature. To the best of our knowledge, none of the following AST approaches have been used in the context of sparsifying DNN’s.

Let \( g(\theta_j) = \log (|\theta_j| + \tau) \), where \( \tau > 0 \). Then, \( \ell_1 \) reduces to what is referred to as a re-weighted \( \ell_1 \) algorithm:

\[
\arg \min_q f \left( q \odot (|\theta_t^{-1}| + \tau) \right)
\]

where \( \cdot \) refers to taking the absolute value of the input.

Suppose, instead, that \( g(\theta_j) = h(\theta_j^2) = \log (\theta_j^2 + \tau) \) and consider repeating the MM procedure described in Section 2 for \( h(\cdot) \). Then, \( \ell_2 \) becomes what is referred to as a re-weighted \( \ell_2 \) algorithm:

\[
\arg \min_q f \left( q \odot \sqrt{\theta_t^{-1} + \tau} \right).
\]

Another variant of \( \ell_2 \) comes from the FOCUSS algorithm, which uses \( g(\theta_j) = |\theta_j|^p, 0 \leq p \leq 2 \). In other words, FOCUSS considers \( \ell_p \) norm regularization, which includes the \( \ell_0 \) pseudo-norm. Repeating the MM procedure for \( g(\theta_j) = h(\theta_j^2) = (|\theta_j|^2)^{p/2} \), \( \ell_2 \) becomes

\[
\arg \min_q f \left( q \odot (|\theta_t^{-1}|^{2-p} + \tau) \right)
\]

where \( \tau > 0 \) is added for stability purposes.

2.3. Implementation Details

In practice, several considerations must be taken into account in the implementation of the proposed approach in \( \ell_2 \). Ideally, one would use \( \ell_2 \) to find successively sparser estimates of \( \theta \) while retaining the same network performance. We employ the stochastic gradient descent (SGD) algorithm and Theano software to find a stationary point of \( \ell_2 \) at each re-weighting iteration. In order to prevent instabilities in the propagation of gradients through the network, it is important that each re-weighting iteration \( t \) is initialized such that the network is not taken too far from its state at \( t - 1 \). For instance, one could initialize \( q \) in \( \ell_2 \) to \( \theta_t^{-1} \odot \psi_t^{-1} \), but this would not allow the learning procedure to move to a new, sparser solution because the initializer would already be a stationary point of \( f(\cdot) \) by definition. We propose two alternatives. The first option is to initialize \( q \) using

\[
(\theta_t^{-1} \odot \psi_t^{-1}) + v, v \sim N(0, \sigma^2)
\]

(11)

where \( \sigma \) controls how far \( \ell_2 \) is from the previous state of the network. Setting \( \sigma \) too large can result in instabilities, whereas setting \( \sigma \) too small can result in \( \ell_2 \) converging to \( \theta_t^{-1} \). The second approach, which we refer to as the greedy method, initializes \( q \) at re-weighting iteration \( t \) to \( q_t^{-1} \).

The complete algorithm pseudo-code is summarized in Algorithm 1. To speed up convergence, it is possible to update \( \psi_t^{-1} \) for a single network layer at each re-weighting iteration. In this regime, the learning procedure remains global since all of the network weights are still updated at each iteration.

As will be shown in Section 6, executing Algorithm 1 leads to a network whose weights are heavily concentrated around 0, but not necessarily strictly equal to 0. The task then becomes to select which weights to prune. We prune the weights at each layer by thresholding, re-train the entire network, and repeat the procedure for the rest of the layers (i.e. the LWC algorithm applied to the output of Algorithm 1). In this case, pruning based on magnitude is justified because the regularizer in \( \psi \) pushes weights which do contribute to the minimization of \( f(\cdot) \) toward 0. Moreover, splicing operations like the ones employed by DNS are unnecessary.

### Algorithm 1 Proposed algorithm

**Require:** \( \theta^0 \)

1. \( t \leftarrow 1 \)
2. **while** not converged **do**
   3. Compute \( \psi_t^{-1} \)
   4. Solve \( \ell_2 \) to obtain \( q_t \)
   5. Update \( \theta_t \) using \( \ell_2 \)
   6. \( t \leftarrow t + 1 \)
3. **end while**

**Return:** \( \theta^{t+1} \)

2.4. Relation to Dynamic Network Surgery [8]

Although the authors of [8] did not frame DNS as an AST approach, DNS can be seen as a special case of the proposed framework. Let \( a = b \) in (2) and

\[
g(\theta_j) = u(\theta_j - a) \theta_j
\]

(12)

where \( u(\cdot) \) denotes the unit-step function. Then, it can be shown that (12) reduces to the DNS algorithm, with the exception that DNS computes \( \omega^t \) using the scaled variable \( q \) whereas the proposed framework uses \( \theta \). Since \( \omega \) is a binary variable, the only difference between the two approaches is when \( \omega_j^{t-1} = 0 \). In this case, \( \omega_j^t \) must be 0 for the proposed framework, implying that pruned connections stay pruned for the choice of \( g(\theta_j) \) in (12). Notice that this discrepancy is a result of the difference of \( g(\cdot) \) in (12). For the choices of \( g(\cdot) \) in Section 2.2, \( \omega_j^t \) is guaranteed to be strictly greater than 0.

3. RESULTS

This section presents experimental results for the proposed algorithms. We focus on classifying the MNIST dataset using LeNet-5, a convolutional neural network architecture consisting of two convolution layers and two fully connected layers,
Table 1: Pruning performance on LeNet-5 in terms of the % non-zeros and the test set error.

| Reference | Re-weighted $\ell_1$ | Re-weighted $\ell_2$ | FOCUSS ($p = 0.5$) | DNS | LWC |
|-----------|----------------------|----------------------|---------------------|-----|-----|
| Conv1     | –                    | 27.8                 | 50.4                | 67.6| 14.2| 12  |
| Conv2     | –                    | 6                    | 4.9                 | 8.1 | 3.1 | 8   |
| FC1       | –                    | 0.7                  | 0.9                 | 1   | 0.7 | 8   |
| FC2       | –                    | 18.6                 | 4.7                 | 15  | 4.3 | 19  |
| Total     | –                    | 1.28                 | 1.26                | 1.7 | 0.9 | 8   |
| Test set error (%) | 0.86 | 1.16 | 1.41 | 1.13 | 0.91 | 0.77 |

Fig. 1: Visualization of proposed learning procedure using the AST in [8]. (a) shows the evolution of validation set error and kurtosis as a function of $t$ for the $Q$ initializer in (11). (b) shows the evolution of validation set error and kurtosis as a function of $t$ for the greedy $Q$ initializer.

We update $\omega$ twice for a given layer during the learning process.

We describe a general AST approach for sparsifying DNN’s. Our approach is founded in principles from the SSR literature and provides an effective method of increasing the sparsity of a given DNN without sacrificing performance. Our approach is competitive with state-of-the-art pruning approaches and has the distinct characteristic of learning sparse weights for the entire network simultaneously.
5. REFERENCES

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton, “Imagenet classification with deep convolutional neural networks,” in Advances in neural information processing systems, 2012, pp. 1097–1105.

[2] Ian Goodfellow, Yoshua Bengio, and Aaron Courville, Deep learning, MIT press, 2016.

[3] Kurt Hornik, Maxwell Stinchcombe, and Halbert White, “Multilayer feedforward networks are universal approximators,” Neural networks, vol. 2, no. 5, pp. 359–366, 1989.

[4] 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, 2015, pp. 1135–1143.

[5] Yann LeCun, John S. Denker, and Sara A. Solla, “Optimal brain damage,” in Advances in Neural Information Processing Systems, 1989, pp. 598–605.

[6] Babak Hassibi and David G Stork, “Second order derivatives for network pruning: Optimal brain surgeon,” in Advances in neural information processing systems, 1993, pp. 164–171.

[7] Xin Dong, Shangyu Chen, and Sinno Jialin Pan, “Learning to prune deep neural networks via layer-wise optimal brain surgeon,” arXiv preprint arXiv:1705.07565, 2017.

[8] Yiwen Guo, Anbang Yao, and Yurong Chen, “Dynamic network surgery for efficient dnns,” in Advances In Neural Information Processing Systems, 2016, pp. 1379–1387.

[9] David R Hunter and Kenneth Lange, “A tutorial on mm algorithms,” The American Statistician, vol. 58, no. 1, pp. 30–37, 2004.

[10] Irina F Gorodnitsky and Bhaskar D Rao, “Sparse signal reconstruction from limited data using focuss: A reweighted minimum norm algorithm,” IEEE Transactions on signal processing, vol. 45, no. 3, pp. 600–616, 1997.

[11] Bhaskar D Rao and Kenneth Kreutz-Delgado, “An affine scaling methodology for best basis selection,” IEEE Transactions on signal processing, vol. 47, no. 1, pp. 187–200, 1999.

[12] Kenneth Kreutz-Delgado and Bhaskar D Rao, “A general approach to sparse basis selection: Majorization, concavity, and affine scaling,” University of California, San Diego, Tech. Rep. UCSD-CIE-97-7-1, 1997.

[13] Bhaskar D Rao and Irina F Gorodnitsky, “Affine scaling transformation based methods for computing low complexity sparse solutions,” in Acoustics, Speech, and Signal Processing, 1996. ICASSP-96. Conference Proceedings., 1996 IEEE International Conference on. IEEE, 1996, vol. 3, pp. 1783–1786.

[14] Stephen G Nash and Ariela Sofer, “Linear and nonlinear programming,” 1996.

[15] Theodore B Trafalis and Nicolas P Couellan, “Neural network training via an affine scaling quadratic optimization algorithm,” Neural Networks, vol. 9, no. 3, pp. 475–481, 1996.

[16] Emmanuel J Candès, Michael B Wakin, and Stephen P Boyd, “Enhancing sparsity by reweighted 11 minimization,” Journal of Fourier analysis and applications, vol. 14, no. 5, pp. 877–905, 2008.

[17] Rick Chartrand and Wotao Yin, “Iteratively reweighted algorithms for compressive sensing,” in Acoustics, speech and signal processing, 2008. ICASSP 2008. IEEE international conference on. IEEE, 2008, pp. 3869–3872.

[18] Theano Development Team, “Theano: A Python framework for fast computation of mathematical expressions,” arXiv e-prints, vol. abs/1605.02688, May 2016.

[19] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner, “Gradient-based learning applied to document recognition,” Proceedings of the IEEE, vol. 86, no. 11, pp. 2278–2324, 1998.