GROUPED SPARSE PROJECTION FOR DEEP LEARNING

Riyasat Ohib
Georgia Institute of Technology
Atlanta, GA 30318, USA
riyasat.ohib@gatech.edu

Nicolas Gillis [*]
Université de Mons
B-7000 Mons, Belgium
nicolas.gillis@umons.ac.be

Sameena Shah [†]
JP Morgan AI Research
sameena.shah@jpmchase.com

Vamsi K. Potluru
JP Morgan AI Research
vamsi.k.potluru@jpmchase.com

Sergey Plis [‡]
Georgia State University
Atlanta, GA 30302, USA
s.m.plis@gmail.com

ABSTRACT

Accumulating empirical evidence shows that very large deep learning models learn faster and achieve higher accuracy than their smaller counterparts. Yet, smaller models have benefits of energy efficiency and are often easier to interpret. To simultaneously get the benefits of large and small models we often encourage sparsity in the model weights of large models. For this, different approaches have been proposed including weight-pruning and distillation. Unfortunately, most existing approaches do not have a controllable way to request a desired value of sparsity as an interpretable parameter and get it right in a single run. In this work, we design a new sparse projection method for a set of weights in order to achieve a desired average level of sparsity without additional hyperparameter tuning which is measured using the ratio of the ℓ_1 and ℓ_2 norms. Instead of projecting each vector of the weight matrix individually, or using sparsity as a regularizer, we project all vectors together to achieve an average target sparsity, where the sparsity levels of the individual vectors of the weight matrix are automatically tuned. Our projection operator has the following guarantees – (A) it is fast and enjoys a runtime linear in the size of the vectors; (B) the solution is unique except for a measure set of zero. We utilize our projection operator to obtain the desired sparsity of deep learning models in a single run with a negligible performance hit, while competing methods require sparsity hyperparameter tuning. Even with a single projection of a pre-trained dense model followed by fine-tuning, we show empirical performance competitive to the state of the art. We support these claims with empirical evidence on real-world datasets and on a number of architectures, comparing it to other state of the art methods including DeepHoyer.

[*]NG acknowledges the support by the European Research Council (ERC starting grant no 679515), and by the Fonds de la Recherche Scientifique - FNRS and the Fonds Wetenschappelijk Onderzoek - Vlanderen (FWO) under EOS Project no O005318F-RG47.

[†]SS and VP are currently at JP Morgan AI Research. This paper was prepared for information purposes by the AI Research Group of JPMorgan Chase & Co and its affiliates ("J.P. Morgan"), and is not a product of the Research Department of J.P. Morgan. J.P. Morgan makes no explicit or implied representation and warranty and accepts no liability, for the completeness, accuracy or reliability of information, or the legal, compliance, financial, tax or accounting effects of matters contained herein. This document is not intended as investment research or investment advice, or a recommendation, offer or solicitation for the purchase or sale of any security, financial instrument, financial product or service, or to be used in any way for evaluating the merits of participating in any transaction.

[‡]SP and RO were in part supported by NIH grants 1R01DA040487-01A1 and 2RF1MH121885.

1 Introduction

Deep learning models are increasingly getting bigger in size and in depth (number of layers) to provide state of the art performance on a wide-range of tasks in computer vision He et al. (2016) and natural language processing Vaswani et al. (2017). However, deploying big models in production systems or edge devices can be challenging as constrained on size and real-time inference are difficult to meet for large models. To address this issue, the field employs two popular approaches: (i) pruning the learned models using sparsity inducing regularizers such as l_1 norm or l_0-constraints Han et al. (2015); Zhang et al. (2018) and (ii) using distillation Ba & Caruana (2014); Ren et al. (2020) to transfer information from a larger teacher network to a smaller student network while maintaining similar level of test performance. We will focus on the first approach, namely sparsity.

In this paper, we design a new sparse projection method for a set of feature vectors $\{x_i \in \mathbb{R}^{n_i}\}_{i=1}^r$ to achieve a desired average sparsity level that is measured using the ℓ_1/ℓ_2 ratios $\frac{||x_i||_1}{||x_i||_2}$ (see below for a precise definition). This projection is inspired by the works of Hoyer (2004); Potluru et al. (2013); Thom et al. (2015) in which each feature vector x_i is independently projected, and used for sparse dictionary learning. The key difference with our projection is that the feature vectors achieve an average target sparsity level: some may end up dense, while others extremely sparse based on the problem. Therefore, our approach has three main advantages: (1) only one sparsity parameter has to be chosen, (2) the sparsity levels of the feature vectors are automatically tuned to achieve the desired average sparsity hence allowing different feature vectors to have different sparsity levels, and (3) our projection has more degrees of freedom hence will generate sparse feature vectors that are closer to the ones we start out with.

We define a novel explicit grouped sparse projection (GSP), and derive several equivalent reformulations. This measure of sparsity is highly interpretable making it simple for the user to set sparsity requirements on the models (Section 2). Also, we provide an efficient algorithm (linear in the size of the problem) to compute this projection, based on the Newton's method (Algorithm 1). Finally, we apply these new projections on several tasks utilizing deep networks (Section 3): *a*) We show that models endowed with our sparsity get the intended level of sparsity in a single training run and produce accuracy comparable to the competition that requires hyperparameter search. *b*) We also show, that a single projection of large models achieves desired sparsity level at the accuracy competitive with the baseline.

2 Grouped Sparse Projection (GSP)

Formulation of grouped sparse projection Let us first present the sparse projection problem for a single vector x, along with a reformulation that will be particularly useful to project to a set of vectors. Given $x \in \mathbb{R}^n_0$ and a sparsity level $s \in [0,1]$, the sparse projection problem can formulated as follows

$$\min_{\tilde{x} \in \mathbb{R}^n} ||x - \tilde{x}||_2 \quad \text{such that} \quad \text{sp}(\tilde{x}) \geq s. \tag{1}$$

We note that the objective function can be rewritten as

$$||x - \tilde{x}||_2^2 = ||x||_2^2 - 2x^T\tilde{x} + ||\tilde{x}||_2^2.$$

Let us introduce the auxiliary variables $\alpha = ||\tilde{x}||_2 \geq 0$, and let us use the change of variables $\tilde{x} = \alpha\bar{x}$ with $||\bar{x}||_2 = 1$. Note that $\text{sp}(\tilde{x}) = \text{sp}(\alpha\bar{x})$ since $\text{sp}(.)$ is invariant to scaling. Hence, α does not appear in the sparsity constraints. Moreover, α can be optimized easily. We have

$$\alpha^* = \arg\min_{\alpha \geq 0} ||x - \alpha\bar{x}||_2 = \max(0, \bar{x}^T x),$$

since $||\bar{x}||_2 = 1$. For $\alpha^* > 0$, we have

$$||x - \alpha^*\bar{x}||_2^2 = ||x||_2^2 - 2\alpha^*\bar{x}^T x + (\alpha^*)^2 = ||x||_2^2 - (\bar{x}^T x)^2.$$

We notice that the sign of the entries of $\bar{x}$ can be chosen freely since the constraints are not influenced by flipping the sign of entries of $\bar{x}$. This implies that, at optimality,

- the entries of $\bar{x}$ will have the same sign as the entries of x, and

- $\alpha^* > 0$ since $x \neq 0$ and $\bar{x} \neq 0$.

Therefore, equation 1 can be reformulated as

$$\max_{\bar{x} \in \mathbb{R}_0^n} \bar{x}^T |x| \quad \text{such that} \quad ||\bar{x}||_2 = 1, \bar{x} \geq 0 \text{ and } \operatorname{sp}(\bar{x}) \geq s. \tag{2}$$

In fact, the optimal solution of equation 1 is given by $\tilde{x}^* = (|x|^T \bar{x}^*)\ \mathrm{sign}(x) \circ \bar{x}^*$ where $\bar{x}^*$ is an optimal solution of equation 2.

Given a set of non-zero vectors $\{x_i \in \mathbb{R}_0^{n_i}\}_{i=1}^r$, the main goal of this paper is to find a set of non-zero vectors $\{\tilde{x}_i \in \mathbb{R}_0^{n_i}\}_{i=1}^r$ as close to $\{x_i \in \mathbb{R}_0^{n_i}\}_{i=1}^r$ as possible and that has an average target sparsity larger than a given $s \in [0, 1]$. Mathematically, we define this *grouped sparse projection* problem as follows:

$$\max_{\bar{x}_i \in \mathbb{R}_0^{n_i}, 1 \leq i \leq r} \sum_{i=1}^r \bar{x}_i^T |x_i| \tag{GSP}$$

$$\text{such that} \quad ||\bar{x}_i||_2 = 1, \bar{x}_i \geq 0 \text{ and } \frac{1}{r} \sum_{i=1}^r \operatorname{sp}(\bar{x}_i) \geq s.$$

The main reason for the choice of this formulation is that it makes equation GSP much faster to solve. In fact, as for equation 1 in Thom et al. (2015), we will be able to reduce this problem to the root finding problem of a nonincreasing function in one variable. In particular, using the objective function $\min_{\tilde{x}_i \in \mathbb{R}_0^{n_i}, 1 \leq i \leq r} \sum_{i=1}^r ||x_i - \tilde{x}_i||_2$ (or $\sum_{i=1}^r ||x_i - \tilde{x}_i||_2^2$) would not allow such an effective optimization scheme. In other words, we are not using the Euclidean distance to measure the error, but rather the inner product between these two vectors. We formally define the problem in Appendix A. We outlined the complete algorithm GSP in Algorithm 1 and present its formulation and computational cost analysis in Appendix B.

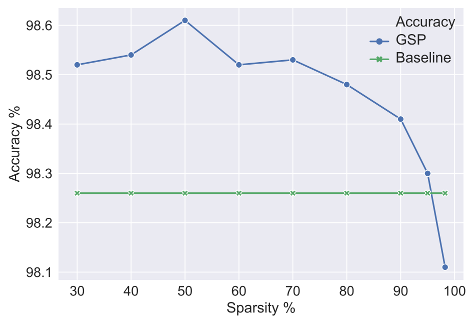
3 Experiments

Pruning methods has been an area of extensive investigation with the goal of identifying and removing unimportant weights in deep neural networks. Heuristic pruning methods typically eliminate smaller weights to produce sparse models. However, the downside of this approach is longer training time and also lack of guarantees of optimality due to insufficient theoretical understanding of these methods. Other approaches formulate this as a sparsity inducing optimization problem, using l_1 regularization, and optimizing it using gradient-based algorithms. Some authors have directly optimized over the pseudo l_0-norm by utilizing stochastic approximation.

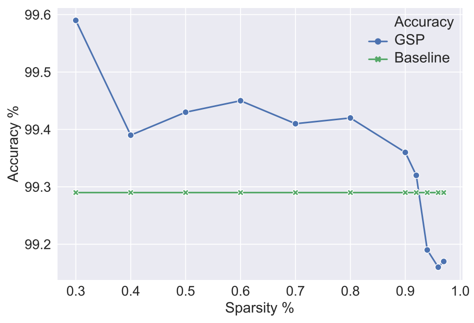
We use our sparse formulations to create a sparse representation of feedforward, CNNs and ResNet models. Using our GSP operator we prune the parameters, inducing sparsity in the weights of these architectures.

Algorithm 1 Grouped sparse projection (GSP)

```
1: INPUT: $\{x_i \in \mathbb{R}^{n_i}\}_{i=1}^r$, the average sparsity $s \in [0, 1]$,
   the accuracy $\epsilon$, the parameter $r_l \in [1/2, 1)$
2: OUTPUT: $\{\tilde{x}_i \in \mathbb{R}^{n_i}\}_{i=1}^r$, with average sparsity in $[s - \epsilon, s + \epsilon]$
3: $\underline{\mu} = 0$, $\bar{\mu} = \tilde{\mu}$, $\mu^* = 0$, $\Delta = \bar{\mu} - \underline{\mu}$.
4: while $|g(\mu^*)| > r\epsilon$ do
5:     $\mu^{old} = \mu^*$.
6:     // Newton's step
7:     $\mu^* = \mu^* + \frac{k - g(\mu^*)}{g'(\mu^*)}$
8:     if $\mu^* \notin [\underline{\mu}, \bar{\mu}]$ then
9:         // Bisection method if Newton's step fails
10:        $\mu^* = \frac{\underline{\mu} + \bar{\mu}}{2}$
11:    end if
12:    if $g(\mu^*) > 0$ then
13:        $\underline{\mu} = \mu^*$
14:    else
15:        $\bar{\mu} = \mu^*$
16:    end if
17:    $(\bar{\mu}, \underline{\mu}) = \mathrm{NewtonPatch}(\bar{\mu}, \underline{\mu}, \mu^*, \mu^{old}, \Delta, r_l, g)$
18:    $\Delta = \bar{\mu} - \underline{\mu}$.
19:    // $\mathcal{D}$ contains the set of discontinuous points
20:    if $\Delta < \epsilon\mu^*$ and $|\mu^* - \mu| < \epsilon\mu^*$ for some $\mu \in \mathcal{D}$ then
21:        break;
22:    end if
23: end while
24: $\alpha_i^* = |x_i|^T \bar{x}_i(\mu^*)$
25: $\tilde{x}_i = \mathrm{sign}(x_i) \circ (\alpha_i^* \bar{x}_i(\mu^*))$
```



(a) Lenet-300-100 Feed Forward Network.



(b) Lenet-5 Convolutional Neural Network.

Figure 1: Sparsity vs Accuracy. Sparsity is not imposed on the baseline model.

(a) LeNet-300-100 and LeNet-5

Methods	Nonzero weights left after pruning	
	LeNet-300-100	LeNet-5 CNN
Original dense model total parameters	266.2k	430.5k
Han et al. (2015)	21.8k (8%)	36k (8%)
Zhang et al. (2018)	11.6k (4.37%)	6.1k (1.4%)
Lee et al. (2018)	13.3k (8%)	8.6k (2.0%)
Ma et al. (2019)	6.4k (2.40%)	5.4k (1.3%)
Yang et al. (2019)	4.6k (1.74%)	3.5k (0.8%)
GSP	10.6k (4%)	17.2k (4%)
GSP†	4.6k (1.76%)	4.7k (1.1%)

(b) ResNet-56 and ResNet-110

	Architecture	Sparsity%	Accuracy%	Drop%
DH	Resnet-56	84.64	92.71	0.42
GSP	Resnet-56	85.78	92.71	0.42
DH	Resnet-110	83.94	92.76	1.24
GSP	Resnet-110	85.90	92.86	0.76
GSP	Resnet-110	90.72	92.45	1.17

Table 1: Table 1a: Element-wise pruning results on LeNet-300-100 Feed Forwards model and LeNet-5 CNN model for GSP compared to recent network pruning techniques. † For this sparsity range, GSP resulted in a slight accuracy loss of 0.15% and 0.29% for LeNet-300-100 and LeNet-5 respectively. Table 1b: Sparsity vs Accuracy for ResNet-56 and ResNet-110 for a single projection followed by finetuning (Single-Shot), in comparison to a full pipeline of DeepHoyer (DH) in the CIFAR-10 dataset. The last two columns report both the mean performance of the model and the drop in performance compared to baseline accuracy of the pretrained model.

3.1 Layer Sparsity

We use the Grouped Sparse Projection operator to project each layer of neural network architectures separately. For our experiments we project each layer with the same sparsity parameter to reduce complexity. Although as easily we could have imposed an exact and different sparsity level for each. The layer-wise projection for the LeNet-300-100 fully connected network and the LeNet-5 CNN network Lecun et al. (1998) on the MNIST dataset makes a convincing test of our GSP operator. We further run tests on the CIFAR-10 dataset Krizhevsky et al. (2009) with ResNet models of depth 56 and 110 He et al. (2016) for single shot sparsity which we discuss in subsection - 3.2. The results of these experiment are summarized in Table-1 and the details of these experimental are presented in the Appendix C.

3.2 Single Shot Weight Pruning

GSP extracts the sparse set of vectors $\tilde{x}_i$ close to the set x_i, s.t. $i \in [m]$ where m is the number of neurons in the layer. It can therefore also be used to make a model sparse by a single projection step instead of having to repeatedly apply the projection steps during the training process. We take a pretrained model for each task and project the model once layerwise with our choice of sparsity. Then we finetune the surviving connections on the training dataset. This is in contrast to the popular regularization based techniques which need to train alongside the sparsity inducing regularizer, then prune the weights and finally finetune the surviving weights to generate the final sparse model. Thus we cut the training of sparse models short by the whole regularization phase when pretrained models are available.

We use this technique on LeNet-300-100 and LeNet-5 on the MNIST dataset and ResNet-56 and ResNet-110 on the CIFAR-10 Dataset. We compare the pruning results with DeepHoyer Yang et al. (2019) on ResNet-56 and ResNet-110 in Table-1(b) which relies on an extra sparsity inducing regularizer phase with 164 epochs of training to first induce sparsity, and then relies on a pruning threshold based parameter depending on the standard deviation of the weights, followed by finetuning for similar number of epochs. In contrast, we use a preset sparsity value for GSP, project the model once, and finetune the surviving weights. We also train the models with the DeepHoyer sparsity inducer Yang et al. (2019) and tweak their parameters to generate results in comparable sparsity range. The results in Table 1 demonstrates that GSP still performs comparable or better in a single shot than DeepHoyer for ResNet-56 and ResNet-110 on CIFAR-10 dataset.

4 Conclusions

In this paper, we have proposed a novel grouped sparse projection algorithm. It allows projecting a set of vectors onto a set of sparse vectors whose average sparsity is defined via a single interpretable parameter. The proposed projection algorithm was shown to be particularly useful in learning sparse deep learning models. As demonstrated, at the extremely high sparsity values (> 0.90) the proposed training in a single run produces deep models of desired level of sparsity with negligible drops in performance. Somewhat surprisingly, GSP-imposed lower sparsity values noticeably improve model performance over the baseline. Besides using GSP during the training, we demonstrate that a single projection of a model trained without constraints results in performance comparable with the baseline and competitive to other similar methods. Interpretability of sparsity, ability to obtain it in a single training run, efficiency of the projection, and theoretical guarantees emphasise the potential of GSP in making practical impact on sparse deep learning. However, our projection is not limited to the listed applications and could also be used in dictionary learning, and other types of neural network architectures such as Transformers.

References

Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger (eds.), *Advances in Neural Information Processing Systems*, volume 27, pp. 2654–2662. Curran Associates, Inc., 2014. URL https://proceedings.neurips.cc/paper/2014/file/ea8fcd92d59581717e06eb187f10666d-Paper.pdf.

Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding, 2015.

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*, pp. 770–778, 2016.

Patrik O Hoyer. Non-negative matrix factorization with sparseness constraints. *Journal of machine learning research*, 5(Nov):1457–1469, 2004.

Niall Hurley and Scott Rickard. Comparing measures of sparsity. *IEEE Transactions on Information Theory*, 55(10):4723–4741, 2009.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

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

Bernd Kummer et al. Newton's method for non-differentiable functions. *Advances in mathematical optimization*, 45(1988):114–125, 1988.

Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791.

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

Rongrong Ma, Jianyu Miao, Lingfeng Niu, and Peng Zhang. Transformed ℓ_1 regularization for learning sparse deep neural networks. *Neural Networks*, 119:286–298, 2019.

VK Potluru, SM Plis, J Le Roux, BA Pearlmutter, VD Calhoun, and TP Hayes. Block coordinate descent for sparse nmf. *International conference on learning representations (ICLR)*, 2013.

Xingkai Ren, Ronghua Shi, and Fangfang Li. Distill bert to traditional models in chinese machine reading comprehension (student abstract). In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 13901–13902, 2020.

M. Thom, M. Rapp, and G. Palm. Efficient dictionary learning with sparseness-enforcing projections. *International Journal of Computer Vision*, 114(2-3):168–194, 2015.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NIPS*, 2017.

Huanrui Yang, Wei Wen, and Hai Li. Deephoyer: Learning sparser neural network with differentiable scale-invariant sparsity measures. *arXiv preprint arXiv:1908.09979*, 2019.

Tianyun Zhang, Shaokai Ye, Kaiqi Zhang, Jian Tang, Wujie Wen, Makan Fardad, and Yanzhi Wang. A systematic DNN weight pruning framework using alternating direction method of multipliers. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 184–199, 2018.

APPENDICES

A GROUPED SPARSE PROJECTION (GSP) AND HOYER SPARSITY

Problem Definition Given a vector $x \in \mathbb{R}^n$, a meaningful way to measure its sparsity is to consider the following measure Hoyer (2004): For $x \neq 0$, we define the sparsity of x as

$$\mathrm{sp}(x) = \frac{\sqrt{n} - \frac{||x||_1}{||x||_2}}{\sqrt{n} - 1} \in [0, 1], \tag{3}$$

We have that $\mathrm{sp}(x) = 0 \iff ||x||_2 = ||x||_1 \iff x(i) = c$ for all i and for some constant c, while $\mathrm{sp}(x) = 1 \iff ||x||_0 = 1$, where $||x||_0$ counts the number of nonzero entries of x. The advantages of $\mathrm{sp}(x)$ compared to $||x||_0$ is that $\mathrm{sp}(x)$ is smooth. For example, with this measure, the vector $[1, 10^{-6}, 10^{-6}]$ is sparser than $[1, 1, 0]$, which makes sense numerically as $[1, 10^{-6}, 10^{-6}]$ is very close to the 1-sparse vector $[1, 0, 0]$. Also, $\mathrm{sp}(x)$ is invariant to scaling (that is, $\mathrm{sp}(x) = \mathrm{sp}(\alpha x)$ for any $\alpha \neq 0$). Note that for any two vectors w and z, $\mathrm{sp}(w) \leq \mathrm{sp}(z) \iff \frac{||w||_1}{||w||_2} \geq \frac{||z||_1}{||z||_2}$. Note also that $\mathrm{sp}(x)$ is not defined at 0, nor for $n = 1$. A property that will be useful later is its nonincreasingness under the so-called soft thresholding operator: Given a vector x and a parameter $\lambda \geq 0$, it is defined as

$$\mathrm{st}(x, \lambda) = \mathrm{sign}(x) \circ [|x| - \lambda e]_+,$$

where $\circ$ is the component-wise multiplication, and $[.]_+$ is the projection onto the nonnegative orthant, that is, $\max(0, .)$.

Lemma .1 (Lemma 3, Thom et al. (2015)). Let $x \in \mathbb{R}^n_0$, and let $\tilde{\lambda}$ be the second largest entry of the vector $|x|$. For $\lambda \leq \tilde{\lambda}$, $\mathrm{sp}\left(\mathrm{st}(x, \lambda)\right)$ is strictly decreasing.

Note that for λ between the largest and second largest entry of $|x|$, $\mathrm{st}(x, \lambda)$ is 1-sparse with $\mathrm{sp}\left(\mathrm{st}(x, \lambda)\right) = 1$, hence $\mathrm{sp}\left(\mathrm{st}(x, \lambda)\right)$ is constant. Interestingly, for $x = c\,e$ for some constant c, it is not possible to sparsify x (because we cannot differentiate between its entries) and $\mathrm{st}(x, \lambda)$ is constant for all $\lambda < c$. We refer the reader to Thom et al. (2015); Hurley & Rickard (2009) for a more detailed discussion on the sparsity measure in equation 3.

B ALGORITHM FOR GROUPED SPARSE PROJECTION

Let us reformulate equation GSP focusing on the sparsity constraint: we have

$$\begin{aligned}\sum_{i=1}^{r} \mathrm{sp}\left(\bar{x}_i\right) &= \sum_{i=1}^{r} \frac{\sqrt{n_i} - ||\bar{x}_i||_1}{\sqrt{n_i} - 1} \\ &= \sum_{i=1}^{r} \frac{\sqrt{n_i}}{\sqrt{n_i} - 1} - \sum_{i=1}^{r} \frac{||\bar{x}_i||_1}{\sqrt{n_i} - 1} \geq rs.\end{aligned}$$

Denoting $k_s = \sum_{i=1}^{r} \frac{\sqrt{n_i}}{\sqrt{n_i}-1} - rs$ and $\beta_i = \frac{1}{\sqrt{n_i}-1}$, equation GSP can be reformulated as follows

$$\begin{aligned}&\max_{\bar{x}_i \in \mathbb{R}^{n_i}, 1 \leq i \leq r} \sum_{i=1}^{r} \bar{x}_i^{\,T} |x_i| \\ &\text{such that } \bar{x}_i \geq 0, ||\bar{x}_i||_2 = 1\, \forall i \text{ and } \sum_{i=1}^{r} \beta_i e^T \bar{x}_i \leq k_s.\end{aligned} \tag{4}$$

We used $||\bar{x}_i||_1 = e^T \bar{x}_i$ since $\bar{x}_i \geq 0$. Note that the maximum of equation 4 is attained since the objective function is continuous and the feasible set is compact (extreme value theorem). Let us introduce the Lagrange variable $\mu \geq 0$ associated with the constraint $\sum_{i=1}^{r} \beta_i e^T \bar{x}_i \leq k_s$. The Lagrange dual function with respect to μ is given by

$$\begin{aligned}\ell(\mu) &= \max_{x_i \geq 0, ||x_i||_2 = 1, i \in [r]} \sum_{i=1}^{r} x_i^T |x_i| - \mu \left(\sum_{i=1}^{r} \beta_i x_i^T e + k_s \right) \\ &= \max_{x_i \geq 0, ||x_i||_2 = 1, 1 \leq i \leq r} \sum_{i=1}^{r} x_i^T (|x_i| - \beta_i \mu e) + \mu k_s. \end{aligned} \tag{5}$$

The dual problem is given by $\min_{\mu \geq 0} \ell(\mu)$. The optimization problem to be solved to compute $\ell(\mu)$ is separable in variables x_i's, hence can be solved individually for each x_i. Let us denote $\bar{x}_i(\mu)$ the optimal solution of equation 5: For each i, there are two possible cases, depending on the value of μ:

1. $|x_i| - \mu\beta_i e \nleq 0$: the optimal $\bar{x}_i(\mu)$ is given by

$$\bar{x}_i(\mu) \;=\; \frac{\left[|x_i| - \mu\beta_i e\right]_+}{\left\|\left[|x_i| - \mu\beta_i e\right]_+\right\|_2} \;=\; \frac{\mathrm{st}(|x_i|, \mu\beta_i)}{\|\,\mathrm{st}(|x_i|, \mu\beta_i)\|_2}.$$

   This formula can be derived from the first-order optimality conditions. Note that this formula is similar to that in Thom et al. (2015). The difference is that the x_i's share the same Lagrange variable μ.

2. $|x_i| - \mu\beta_i e \leq 0$: the optimal $\bar{x}_i(\mu)$ is given by the 1-sparse vector whose nonzero entry corresponds to the largest entry of $|x_i| - \mu\beta_i e$, that is, of $|x_i|$. Note that if the largest entry of $|x_i|$ is attained for several indices, then the optimal 1-sparse solution $\bar{x}_i{}^*$ is not unique.

   Note also that this case coincides with the case above for μ in the interval between the largest and second largest entry of x_i.

It is interesting to observe that for $\mu = 0$, we have $\bar{x}_i(0) = \frac{|x_i|}{||x_i||_2}$. If $\bar{x}_i(0)$ is feasible, that is, $\sum_{i=1}^{r} \beta_i e^T \bar{x}_i(0) \leq k_s$, then it is optimal since the error of GSP is zero: this happens when the x_i's are already sparse enough and does not need to be projected. Otherwise, the constraint $\sum_{i=1}^{r} \beta_i e^T \bar{x}_i \leq k_s$ will be active at optimality (it can be checked that otherwise $\bar{x}_i$ can locally be made closer to $|x_i|$, aligning $\bar{x}_i$ in the direction $|x_i|$). Finally, unless the solution for $\mu = 0$ is feasible (which can be checked easily), we need to find the value of μ such that

$$g(\mu) \;=\; \sum_{i=1}^{r} \beta_i e^T \bar{x}_i(\mu) - k_s \;=\; 0,$$

that is, find a root of $g(\mu)$. Let us denote $\tilde{\mu}$ the smallest value of μ such that $\bar{x}_i(\mu)$ are all 1-sparse so that $e^T \bar{x}_i(\mu) = 1$ hence

$$\begin{aligned} g(\tilde{\mu}) &= \sum_{i=1}^{r} \beta_i - k_s = \sum_{i=1}^{r} \frac{1}{\sqrt{n_i} - 1} - \sum_{i=1}^{r} \frac{\sqrt{n_i}}{\sqrt{n_i} - 1} + rs \\ &= r(s-1) \leq 0. \end{aligned} \tag{6}$$

The value $\tilde{\mu}$ is given by the second largest entry among the vectors $|x_i|/\beta_i$'s. In fact, if μ is larger than the second largest entry of $|x_i|$, then $\tilde{x}_i(\mu)$ is 1-sparse (see above). If the largest and second largest entry of a vector $|x_i|$ are equal, then $g(\mu)$ is discontinuous. This is an unavoidable issue when one wants to make a vector sparse: if the largest entries are equal to one another, one has to decide which one will be set to zero. For example, the vectors $[1, 0]$ and $[0, 1]$ are equally good 1-sparse vectors to approximate $[1, 1]$.

For $0 \leq \mu \leq \tilde{\mu}$, the function $g(\mu)$ is strictly decreasing because the soft tresholding operator increases the sparsity function $\mathrm{sp}(.)$ (Lemma .1). This implies that if $g(\mu)$ is not discontinuous around $g(\mu) = 0$, it has a single root, which we denote μ^*.

A possible way to find this root of $g(\mu)$ is to use bisection. However, this can be relatively slow with linear convergence rate. Although $g(\mu)$ is not differentiable everywhere (it is not at the points where an entry of $\bar{x}_i(\mu)$ becomes zero) and can be discontinuous, we observe that using Newton's method, with initial point $\mu = 0$ performs very well (see below for some numerical experiments). Newton's method was also successfully used in Thom et al. (2015), and this is a standard methodology in optimization because points where the function is not differentiable typically form a set of measure zero; see for example Kummer et al. (1988). The reason to choose $\mu = 0$ as the initial point is because $g(\mu)$ decreases initially fast (all entries of $\bar{x}_i$ corresponding to a nonzero entry of x_i are decreasing) while it tends to saturate for large μ (see below for an example). In particular, we cannot initialize μ at values larger than $\tilde{\mu}$ since $g(\mu)$ is constant ($g'(\mu) = 0$) for all $\mu \geq \tilde{\mu}$.

Because g is not differentiable everywhere, we have added a safety procedure using the bisection method (which is guaranteed to converge linearly with rate 1/2) when Newton's method goes outside of the current feasible interval $[\underline{\mu}, \bar{\mu}]$ containing the solution μ^* and which is updated at each step. This is outlined in Algorithm-2. Moreover, because of discontinuity, Newton's method could stagnate locally and not converge (although in practice, we have not observed this behavior). For this reason, we have also added a safety procedure that guarantees the algorithm to converge linearly: if the feasible interval $[\underline{\mu}, \bar{\mu}]$ has not decreased by at least a factor of $1/2 \leq r_l < 1$, then bisection is used.

We have also added a procedure to detect discontinuity in which case the algorithm returns μ such that $g(\mu + \epsilon\mu) < 0 < g(\mu - \epsilon\mu)$, where ϵ is a desired accuracy (see below for an example). Note that the discontinuous points can be precomputed and corresponds to the largest entries of the x_i's when they are not uniquely attained. We will denote $\mathcal{D}$ the set of discontinous points of $g(\mu)$.

Algorithm 1 summarizes our algorithm. Note that the accuracy ϵ does not need to be high in practice, say 0.001, since there will not be a significant difference between a vector of sparsity s and sparsity $s \pm 0.001$.

Algorithm 2 NewtonPatch($\bar{\mu}, \underline{\mu}, \mu^*, \mu^{old}, \Delta, r_l, g$)

```
1:  // Newton's method fails linear convergence
2:  if $\bar{\mu} - \underline{\mu} > r_l\Delta$ and $|\mu^{old} = \mu^*| < (1 - r_l)\Delta$ then
3:      // Bisection method used
4:      $\mu^* = \frac{\underline{\mu}+\bar{\mu}}{2}$
5:      if $g(\mu^*) > 0$ then
6:          $\underline{\mu} = \mu^*$
7:      else
8:          $\bar{\mu} = \mu^*$
9:      end if
10: end if
11: Return: $(\bar{\mu}, \underline{\mu})$
```

Computational cost The main computational cost of Algorithm 1 is to compute $g(\mu)$ and $g'(\mu)$ at each iteration, which requires $O(N)$ operations where $N = \sum_{i=1}^{r} n_i$. Most of the computational cost resides in computing the $\bar{x}_i(\mu)$'s and some inner products. The computational cost per iteration is therefore linear in the size of the data. In practice, we observed that Netwon's method converge very fast and does not require much iterations. Let us take $n_i = 1000$ for all $i = 1, 2, \ldots, 100$ and generate each entry of the x_i's using the normal distribution $N(0, 1)$. For each sparsity level, we generate 100 such data points and Table 2 reports the average and maximum number of iterations needed by Algorithm 1. In all cases, it requires less than 4 iterations for a target accuracy of $\epsilon = 10^{-4}$.

	s_0	$s = 0.7$	$s = 0.8$	$s = 0.9$	$s = 0.95$	$s = 0.99$
Average	20.86%	3.88	3.78	3.98	3.75	3.77
Maximum	21.01%	4	4	4	4	4

Table 2: Average and maximum number of iteration for Algorithm 1 to perform grouped sparse projection of 100 randomly generated vectors of length 1000, with precision $\epsilon = 10^{-4}$. The column s_0 gives the average and maximum initial sparsity of the 100 randomly generated vectors x_i's.

Example 1. Let us take x_i as the ith row of the matrix

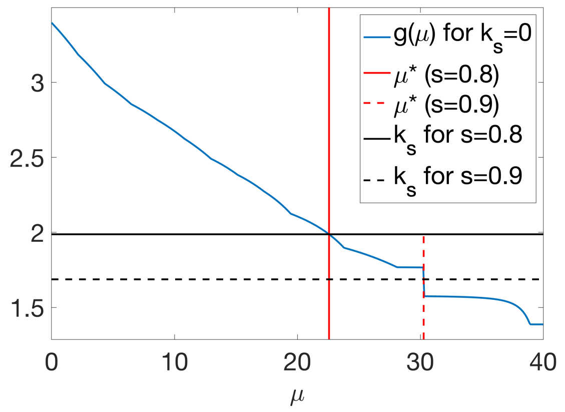


Figure 2: Function $g(\mu)$ for $k = 0$ and roots for $s \in [0.8, 0.9]$.

$$\begin{pmatrix} 1 & 2 & 14 & 9 & -14 & 9 & -1 & 5 & -11 & 7 \\ 8 & 2 & -6 & -13 & -24 & -13 & -6 & 1 & 4 & -11 \\ -3 & -2 & 3 & -1 & -6 & 3 & 18 & -2 & -2 & -19 \end{pmatrix}.$$

We have that the average sparsity of the x_i's is 33.03%. Projecting the x_i's to have an average sparsity of 80% ($s = 0.8$) with accuracy 10^{-4}, we obtain

$$\begin{pmatrix} 0 & 0 & 14.68 & 0 & -14.68 & 0 & 0 & 0 & -2.31 & 0 \\ 0 & 0 & 0 & -5.17 & -27.37 & -5.17 & 0 & 0 & 0 & -1.13 \\ 0 & 0 & 0 & 0 & 0 & 0 & 17.31 & 0 & 0 & -19.61 \end{pmatrix}$$

with 4 iterations of Algorithm 1 and with average sparsity 80.0001%. Now using $s = 0.9$, we obtain

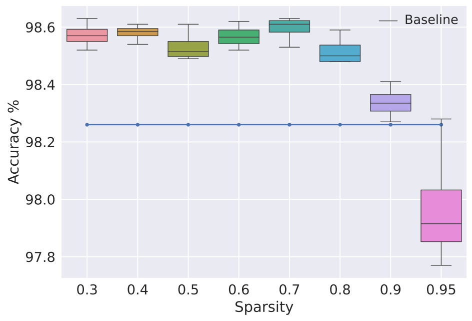
$$\begin{pmatrix} 0 & 0 & 14 & 0 & -14 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -24 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 16.29 & 0 & 0 & -20.37 \end{pmatrix}$$

with 12 iterations of Algorithm 1 and with average sparsity 87.36%. The reason for this larger number of iterations and inaccurate sparsity is that $g(\mu)$ is not continuous for this level of sparsity; see Figure 2 for an illustration. In practice, this is not likely to happen when r is large because the gap of $g(\mu)$ at a discontinuous point is smaller (the influence of a single vector x_i is comparatively less important).

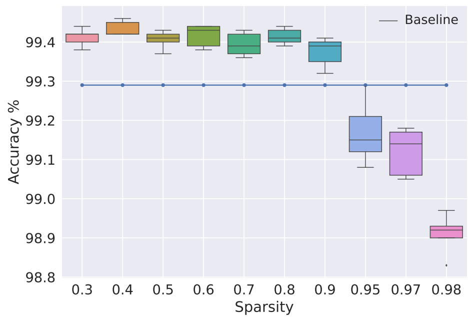
In fact, the sparsity 90% cannot be achieved (any sparsity s between 87.36% and 93.75%): increasing s to 92.5%, the entry -14 above is replaced by zero to get sparsity 93.75%.

C Layer Experiment Details

Since the GSP operator gives us the freedom to choose which layers to make sparse, we choose the fully connected layers and the convolutional layers in the mentioned networks. For the fully connected layers, we project the connections in each layer separately, with a target sparsity s. This ensures the weights of that particular layer have a Hoyer Sparsity measure of s. For the convolutional layers, we project each $k \times k$ filters treating each as a vector $x_i \in \mathbb{R}^n$ in our formulation, where $n = k^2$ and all the filters in a particular layer are projected at once. All the models in this work were implemented utilizing the PyTorch deep learning framework.



(a) Lenet-300-100 Feed Forward Network.



(b) Lenet-5 Convolutional Neural Network.

Figure 3: Sparsity vs Accuracy for LeNet-300-100 and LeNet-5 over multiple random restarts. Performance of the baseline model without imposed sparsity (hence constant value) is included for comparison.

C.1 Experiments on MNIST

In this work, we used the MNIST handwritten dataset Lecun et al. (1998) consisting of greyscale images of size 28×28 pixels. For accessing the dataset we used the torchvision dataset API included in the PyTorch library. The MNIST dataset was used for the experiments on LeNet-300-100 Feed Forward Network and LeNet-5 Convolutional Neural Network. We utilized the full 60000 images in the training set for learning our models and evaluated them on the 10000 images in the testing set. We normalized both the training set and the testing set to have zero mean and a variance of one. For optimization during the training process, we used the Adam optimizer Kingma & Ba (2014) with a learning rate of 0.001 and decay rate of 0.1 for every 85 epochs.

For the experiments with intermittent projections during the training phase, we first update the models using backpropagation and combine it with our sparse projection operator every 80 mini-batches for 150 epochs. Next, we set the s fraction of the lowest parameters of the model to zero where s is the sparsity constraint set in the GSP algorithm. Note that since we do the projection in a set interval of minibatches, the final model might have some weights close to zero but not exactly zero. Hence, to maintain the sparsity constraints in induced layerwise projection, we need to prune the weights of the layers after each projection, keeping the largest $1 - s$ fraction of the parameters. Finally, we finetune the surviving parameters for the remaining 150 epochs.

We perform multiple random restart experiments on the MNIST dataset for the LeNet-300-100 and LeNet-5 models over a range of sparsity values and show a box plot for the accuracy at each sparsity level. The results are shown n Fig-3(a) and 3(b). We notice an increase in model performance for most of the sparsity values, and a negligible accuracy loss in the extremely sparse regime.

C.2 Experiments on CIFAR-10

We use the CIFAR-10 dataset Krizhevsky et al. (2009) to train and test the Resnet-56 and Resnet-110 models for our experiments. The CIFAR-10 dataset was accessed through the dataset API of the torchvision package of PyTorch. We performed the standard preprocessing on the data which included horizontal flip, random crop and normalization on the training set. With the CIFAR-10 dataset we perform two different types of experiments. Similar to the experiments on the MNIST dataset, we perform a layerwise induced GSP integrated with the training phase. We also perform single shot pruning with a single projection of the model weights followed by the finetuning phase (as described in the main paper). For the sparsity induced training experiments, we learn the weights of the Resnet models for 164 epochs, performing a projection with the GSP every 200 iterations. The learning rate was set at 0.1 at the start, and was then decayed by 0.1 at epochs 81 and 122. Similar to the MNIST experiments, we prune the weights keeping the largest $1 - s$ fraction of the parameters. Finally, we finetune the model with the surviving parameters for 164 epochs.

Methods	Architecture	Sparsity%	Accuracy%	Drop%
DH	Resnet-56	84.64	92.71	0.42
Induced GSP	Resnet-56	88.33	92.04	1.09
	Resnet-56	89.50	91.62	1.51
One Shot GSP	Resnet-56	85.78	92.71	0.42
DH	Resnet-110	83.94	92.76	1.24
Induced GSP	Resnet-110	95.18	92.67	0.95
One Shot GSP	Resnet-110	85.90	92.86	0.76
	Resnet-110	90.72	92.45	1.17

Table 3: Sparsity vs Accuracy for ResNet-56 and ResNet-110 for both layerwise and one-shot GSP, in comparison to a full pipeline of DeepHoyer (DH) in the CIFAR-10 dataset. The last two columns reports both the mean performance of the model and the drop in performance compared to the respective baseline accuracy of the pretrained model used for that experiment.

Layerwise GSP induced training was carried out on the CIFAR-10 dataset with the Resnet-56 and Resnet-110 models. For these experiments we separately projected the parameters of each layer of the Resnet models every 200 mini-batches, right after the backpropagation step. The models were trained with GSP induction for a total of 164 epochs. Then we fixed the nonzero weights and finetuned the models for another 164 epochs. The results of these experiments along with those of DeepHoyer Yang et al. (2019) and our single-shot GSP experiments are presented in Table 3 for comparison.