AUTOMATIC PARAMETER TYING IN NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Recently, there has been growing interest in methods that perform neural network compression, namely techniques that attempt to substantially reduce the size of a neural network without significant reduction in performance. However, most existing methods are post-processing approaches in that they take a learned neural network as input and output a compressed network by either forcing several parameters to take the same value (parameter tying via quantization) or pruning irrelevant edges (pruning) or both. In this paper, we propose a novel algorithm that *jointly learns and compresses* a neural network. The key idea in our approach is to change the optimization criteria by adding k independent Gaussian priors over the parameters and a sparsity penalty. We show that our approach is easy to implement using existing neural network libraries, generalizes ℓ_1 and ℓ_2 regularization and elegantly enforces parameter tying as well as pruning constraints. Experimentally, we demonstrate that our new algorithm yields state-of-the-art compression on several standard benchmarks with minimal loss in accuracy while requiring little to no hyperparameter tuning as compared with related, competing approaches.

1 INTRODUCTION

Neural networks represent a family of highly flexible and scalable models that have rapidly achieved state-of-the-art performance in diverse domains including computer vision (Krizhevsky et al., 2012; Girshick et al., 2014; He et al., 2016), speech (Hinton et al., 2012; Deng et al., 2013), and sentiment analysis (Glorot et al., 2011). Despite their successes, the storage requirements of large, modern neural networks make them impractical for certain applications with storage limitations (e.g., mobile devices). Moreover, as they are often trained on small datasets compared to their number of parameters (typically in the millions for state-of-the-art models), they can potentially overfit. In recent work, Denil et al. (2013) showed that a large proportion of neural network parameters are in fact not required for their generalization performance, and interest in model compression has surged.

A variety of methods have been proposed to perform compression including pruning (LeCun et al., 1990; Han et al., 2015), quantization (Han et al., 2016; K. Ullrich, 2017; Chen et al., 2015), low-rank approximation (Denil et al., 2013; Denton et al., 2014; Jaderberg et al., 2014), group lasso (Wen et al., 2016), variational dropout (Molchanov et al., 2017), etc. Here, we focus on the quantization/parameter tying approach to compression. Parameter tying assumptions occur naturally in the construction of convolutional neural networks (CNNs), but in these applications, the parameters to be tied are usually selected in advance of training. Recent work has focused on automatic parameter tying, i.e., automatically discovering which parameters of the model should be tied together. Nowlan & Hinton (1992) proposed a soft parameter tying scheme based on a mixtures of Gaussians prior and suggested a gradient descent method to jointly optimize both the weights in the network and the parameters of the mixture model. Chen et al. (2015) proposed a random parameter tying scheme based on hasing functions. Han et al. (2016) proposed a compression scheme that involved layer-wise parameter tying (using k-means clustering) as well as thresholding to remove edges with low weight. This work demonstrated that high compression rates are achievable without much loss in accuracy. Building on the work of (Nowlan & Hinton, 1992), K. Ullrich (2017) imposed a Gaussian mixture prior on the parameters to encourage clustering. At convergence, they proposed clustering the weights by assigning them to the mixture component that generates each weight with highest probability. Louizos et al. (2017) proposed a full Bayesian approach to compression using scale

mixture priors. This approach has the advantage that posterior distributions can be used to estimate the significance of individual bits in the learned weights. Louizos et al. (2017) demonstrated that this approach can yield state-of-the-art compression results for some problems.

While much of the previous work has demonstrated that significant compression can be achieved while preserving the accuracy of the final network (in many cases $\approx 1\%$ loss in accuracy), many of these approaches have a number of drawbacks that limit their practical applications. The Gaussian mixture approach of Nowlan & Hinton (1992) and K. Ullrich (2017) can be computationally expensive as these models have an extensive number of hyperparameters that are somewhat sensitive and must be tuned using a validation set. Moreover, even when the parameters are known, the GMM approach suffers from well known local minima issues. These local minimas are in addition to the local minimas encountered while training a neural network which in turn incurs high computational cost. The approach of Han et al. (2016) only applies parameter tying layerwise, which potentially limits the ultimate reduction in the number of weights that describe the network (codebook storage can become expensive especially for deep networks with hundreds of layers). The parameter tying approach of Chen et al. (2015) is also only applied layerwise, and it typically requires more clusters, i.e., larger K, before the random weight sharing is effective (our experiments confirm that random parameter tying yields poor results when the number of distinct parameters is too small). Finally, the full Bayesian approach has a number of additional parameters to tune (e.g., constraints on variances, careful initialization of each of the variational parameters, etc.). The Bayesian approach also requires sampling for prediction (it can be done deterministically, but this incurs some additional loss). We hope to argue that such sophisticated methods may not be necessary to achieve good compression in practice.

The approach to compression in this work considers an independent Gaussian prior, that is, each parameter is non-probabilistically assigned to one of K independent Gaussian distributions, and the prior penalizes each weight by its ℓ_2 distance to the mean of its respective Gaussian. This prior places no restriction on which weights can be tied together (e.g., weights from the input could be tied to weights into the output), reduces the number of hyperparameters that need to be tuned compared to Gaussian mixtures, and requires only a small change to the typical gradient descent updates in the backpropagation method. We observe that quantization alone is not enough to achieve the desired level of sparsity, and we demonstrate experimentally that adding a standard ℓ_1 penalty on top of a K-means prior yields state-of-the-art results on standard benchmark data sets. Finally, while our approach can be applied to quantize and sparsify a pretrained network, the proposed methodology is efficient enough to implement and train from scratch using existing libraries.

2 Parameter Tying in Neural Networks

We consider the general problem of learning a neural network by minimizing the regularized loss function

$$\mathcal{L}(\boldsymbol{W}) = E_D(\boldsymbol{W}) + \lambda R(\boldsymbol{W}),$$

where $\boldsymbol{W}$ is the set of network parameters of size N, E_D is the loss on training data D, and R is a regularization function ostensibly chosen to penalize overly complex models and to improve generalization performance. The regularizer R is usually chosen to be the L1 or L2 norm of $\boldsymbol{W}$, which encourages sparse parameter vectors or bounded parameter vectors, respectively.

In this work, we examine an alternative form of regularization using parameter tying: $\boldsymbol{W}$ is partitioned into K sets, and parameters in each set are constrained to be equal, i.e., $\boldsymbol{W}$ contains only K distinct values. Formally, for each $k \in \{1, \ldots, K\}$ let $C_k \subseteq \{1, \ldots, N\}$ be disjoint sets, or clusters, of parameter indices, such that $\cup_{k=1}^{K} C_k = \{1, \ldots, N\}$. If the parameters indexed by C_k are required to share the same value, then learning under parameter tying yields the constrained optimization problem

$$\min_{\boldsymbol{W}} \mathcal{L}(\boldsymbol{W})$$
$$\text{subject to } w_i = w_j, \forall k \in \{1, \ldots, K\}, i, j \in C_k$$

In the neural networks community, parameter tying is a fundamental component of convolutional neural networks (CNNs), where weights are shared across a specific layer. In practice, we often encounter high-dimensional problems with no obvious structure and with no explicit knowledge

about how model parameters should be tied. This motivates our goal of discovering parameter tying without prior knowledge, that is, automatic parameter tying, in which we optimize with respect to both the parameter weights and the cluster assignments. In general, this problem will be intractable as the number of possible partitions of the set of weights into clusters, the Bell number, grows exponentially with the size of the set.

Instead, we consider a relaxed version of the problem, in which parameters are softly constrained to take values close to their average cluster values. To achieve this, we add a clustering distortion penalty, $J(\boldsymbol{W}, \boldsymbol{\mu})$, on the parameters to form a new combined optimization problem. Here, we consider the K-means loss, defined to be the sum of the distance between each parameter and its corresponding cluster center.

$$J(\boldsymbol{W}, \boldsymbol{\mu}) \triangleq \frac{1}{2} \sum_n \min_k \|w_n - \mu_k\|_2^2, \tag{1}$$

where $\boldsymbol{\mu} \in \mathbb{R}^K$ is the vector of cluster centers. Note that the ℓ_2 distance measure in (1) could be replaced with a variety of other metrics (e.g., if the ℓ_1 distance is selected, then (1) can be optimized by the K-medians algorithm). From a Bayesian point of view, given a fixed $\boldsymbol{\mu}$, $J(\cdot, \boldsymbol{\mu})$ can be considered as a prior probability over the weights that consists of K independent Gaussian components with different means and shared variances.

While K-means has been used for quantization of the weights after training, e.g., see Han et al. (2016) and Gong et al. (2015), we propose to incorporate it directly into the objective as a prior. The hope is that this prior will guide the training towards a *good* parameter tying from which hard-tying will incur a relatively small loss. Indeed, the primary observation of this paper, is that J is a highly effective prior that induces quantization. Compared to a Gaussian mixture prior, J has fewer parameters/hyperparameters to learn/tune. In addition, J is a more natural prior if we believe that the data is actually generated from a model with finitely many parameters: we expect both models to perform comparably when the distinct parameters are far apart from each other, but as the clusters move closer together, the GMM prior leads to clusters with significant overlap. In the worst case, the GMM prior converges to a mixture such that each weight has almost exactly the same probability of being generated from each mixture component. This yields poor practical performance. In contrast, J forces each weight to commit to a single cluster, which seems to result in a lower loss in accuracy when hard-tying. In addition, the maximum likelihood objective for the GMM prior can encounter numerical issues if any of the variances tend to zero, this can happen as components are incentivized to reduce variances by eventually collapsing onto parameters. This problem can be alleviated by a combination of setting individual learning rates for the GMM and model parameters, annealing the GMM objective (Nowlan & Hinton, 1992), or imposing hyperpriors on the Gaussian parameters to effectively lower-bound the variances (K. Ullrich, 2017). Even with these modifications, significant tuning may still be required to produce good solutions.

3 Sparse Automatic Parameter Tying

Similar to Nowlan & Hinton (1992), we observed that optimizing an objective under a prior that encourages clustering does not typically yield a significant cluster near zero. A large cluster near zero encourages model sparsity: weights that are effectively zero can be dropped from the model and nodes that have only zero weights can also be dropped.

Following the approach of Han et al. (2016), if we store the original parameters of a model using b-bit floats (typically 16 or 32), and quantize them so that they only take K distinct values, then we only need to store the cluster means, $\boldsymbol{\mu}$, in full precision and the quantized parameters by their index, corresponding roughly to a compression rate of

$$r = \frac{Nb}{N \log_2 K + Kb}. \tag{2}$$

For a parameter-heavy model such that $N \gg K$, the denominator in (2) is dominated by $N \log_2 K$, so most of the savings from quantization comes from storing parameter indices with $\log_2 K$ instead of b bits. While this compression rate is often significant, larger gains[1] can be achieved by encouraging a large *zero cluster*. To this end, we add an additional sparsity-inducing weight penalty $E_S(\boldsymbol{W})$

[1]For example, by using the sparse encoding scheme proposed by (Han et al., 2016).

to the learning objective, resulting in the joint Automatic Parameter Tying (APT) objective,

$$\min_{\boldsymbol{W},\boldsymbol{\mu}} \mathcal{L}(\boldsymbol{W},\boldsymbol{\mu}) = E_D(\boldsymbol{W}) + \lambda_1 J(\boldsymbol{W},\boldsymbol{\mu}) + \lambda_2 E_S(\boldsymbol{W}), \tag{3}$$

The case in which $\lambda_2 = 0$, corresponding to no sparsity inducing prior, will be referred to as *plain* APT. In this work, we consider the lasso penalty $E_S(\boldsymbol{W}) \triangleq ||\boldsymbol{W}||_1$, and find experimentally that this additional penalty significantly increases model sparsity without significant loss in accuracy, for large enough K.

We propose a two-stage approach to minimize (3). In stage one, soft-tying, the objective is minimized using standard gradient/coordinate descent methods. In stage two, hard-tying, the soft clustering penalty is replaced with a hard constraint that forces all weights in each cluster to be equal. The resulting objective function can then be minimized using projected gradient descent. Unfortunately, (3) is not a convex optimization problem, even if E_D is convex, as J, the K-means objective is not convex. So, our methods will only converge to local optima in general. We note that unlike the GMM penalty (Nowlan & Hinton, 1992) the K-means problem can be solved exactly in polynomial time in the one-dimensional (1-D) case using dynamic programming (Wang & Song, 2011).

In our experiments, we use a fast implementation of 1-D K-means that produces a comparable solution to the method proposed in (Wang & Song, 2011), but requires much less time. Also K is selected using a validation set, though nonparametric Bayesian methods could be employed to automatically select K in practice (e.g., DP-means (Kulis & Jordan, 2012)).

3.1 Soft-Tying (coordinate descent)

The APT objective (3) decomposes into functions only of $\boldsymbol{W}$ and $\boldsymbol{\mu}$, each of which can be optimized separately with standard algorithms. Typically, the K-means objective is formulated in terms of an $N \times K$ matrix $\mathbf{A}$ of auxiliary variables, such that $a_{nk} \in \{0, 1\}$ indicates whether parameter w_n belongs to cluster k, and $\sum_k a_{nk} = 1$. The K-means penalty can then be written as

$$J(\boldsymbol{W},\boldsymbol{\mu},\mathbf{A}) \triangleq \frac{1}{2}\sum_{k=1}^{K}\sum_{n=1}^{N} a_{nk}\|w_n - \mu_k\|_2^2,$$

which yields the joint optimization problem

$$\min_{\boldsymbol{W},\boldsymbol{\mu},\mathbf{A}} \mathcal{L}(\boldsymbol{W},\boldsymbol{\mu},\mathbf{A}) = E_D(\boldsymbol{W}) + \lambda J(\boldsymbol{W},\boldsymbol{\mu},\mathbf{A}), \tag{4}$$

where $\boldsymbol{A}$ is restricted to satisfy the above constraints. We can find a local optimum of this objective iteratively with a simple block coordinate descent algorithm that alternates between performing backpropagation/gradient descent and updating cluster assignments.

In our implementation we use the standard formulation of the K-means problem, but specialized to 1D: we take advantage of the fact that comparisons can be done on entire sets of parameters, if we sort them in advance, and operate on partitions of parameter clusters. In practice we've found the approximate version to quickly converge to solutions comparable in quality to the dynamic programming version, while requiring much less time. Depending on the size of $\boldsymbol{W}$, updating the cluster assignments and means could be expensive to perform frequently; instead, we have found good performance by running K-means only once every 1000 or so parameter updates, and only running it long enough (say 100 iterations) to obtain an approximate solution. As shown in our experiments, the frequency of K-means updates does not significantly impact the results.

3.2 Hard-Tying (projected descent)

We replace the soft-tying procedure with hard-tying once the combined objective has been sufficiently optimized. An annealing schedule can be used to gradually increase the importance of J or E_S. However, in our experiments we simply ran the soft-tying procedure for a fixed number of iterations followed by a fixed number of iterations of hard-tying.

In the case of hard-tying, the regularized objective (3) can be optimized via projected gradient descent where the derivatives are first calculated using backpropagation and then all components of the gradient corresponding to the parameters in cluster k are replaced by their average to yield the

projected update (i.e., projecting $\frac{\partial E_D}{\partial \boldsymbol{W}}$ onto the subspace determined by $\mathbf{A}$). We note that this is distinct from the gradient update suggested by Han et al. (2016), which sums the partial derivatives of all parameters in the same cluster but does not compute the average. This difference arises as Han et al. (2016) only allow weight sharing within each layer, while the projected gradient method we propose can handle weight tying even across layers.

4 EXPERIMENTS

In our implementation of APT, we use Tensorflow (Abadi et al., 2015) to optimize E w.r.t to $\boldsymbol{W}$, and a C++ procedure for the 1D K-means clustering problem. Note that soft-tying APT can be readily optimized by any modern neural network library that provides auto-differentiation (in which case the cluster centers are updated by gradients). However, since Tensorflow does not handle arbitrary parameter tying constraints, we use an additional C++ procedure for the hard-tying assignment. For backpropagation, we used a version of the adaptive step size procedure Adadelta (Zeiler, 2012), which is based on Adagrad (Duchi et al., 2011) and is supported by Tensorflow. Our initial set of experiments using stochastic gradient descent plus momentum required more tuning and resulted in significantly higher variance in the final answer than Adadelta.

Unless otherwise mentioned, we initialize the neural network parameters using the method proposed by Glorot & Bengio (2010), and initialize the cluster centers heuristically by evenly distributing them along the range of initialized network parameters. As our experiments are concerned with classification problems, we use the standard cross-entropy objective as our data loss. All of our experiments were performed on quad-core Intel i7 2.7 GHz machines with 16 GB RAM.

We consider three different sets of experiments. First, we investigate the performance of APT in isolation on LeNet-300-100 examining the effect of APT regularization, number of clusters versus accuracy, and number of K-means updates. Inspired by recent work on the generalization performance of neural networks, our second set of experiments aims to understand whether or not parameter tying has any effect on the generalization performance of neural networks. Finally, our third set of experiments, compares the performance of APT and other state-of-the-art methods under various compression metrics.

4.1 ALGORITHMIC BEHAVIOR

We begin by optimizing the APT objective using LeNet-300-100 on MNIST. We trained with soft-tying for twenty thousand iterations, and switched to hard-tying for another twenty thousand iterations. Figure 1 depicts a typical weight distribution produced by APT versus training without any regularization using the same initialization and learning rate. As expected, APT lead to a clear division of the weights into clusters, the largest of which are clustered quite closely to zero. Figure 2a shows the evolution of the clusters selected by APT versus iteration. Generally, the soft-tying phase of APT does not fundamentally change the quality of the learned solution (as measured by classification performance) or convergence speed, compared to without APT. However, hard-tying for small K can lead to significant loss in accuracy. In this demonstration, $K = 8$ is sufficient for representing the parameters learned with soft-tying: although the projection from $\boldsymbol{W}$ to the subspace determined by $\mathbf{A}$ resulted in significant additional loss, hard-tying was able to gradually recover from it by discovering a nearby point in the solution space that lowers the loss. We observe that hard-tying mostly preserves the soft-tying solution for sufficiently large K, but is generally unable to find a better solution than the original one through soft-tying. For this particular model, when selecting $K = 2$ and hard-tying the accuracy drops from 98% to 96% while for $K \geq 32$ almost no loss in accuracy is observed when hard-tying. Additional experimental results, including a visualization of the weights produced by APT with and without ℓ_1 can be found in Appendix A.

We also explored the effect of coordinate switching frequency on the learning outcome, for which we reran the previous experiments with different numbers, t, of parameter updates between k-means runs, for $t \in \{1, 10, 100, 1000, 10000, 20000\}$. We observed that APT was generally not sensitive to k-means frequency, except for very small K, justifying our heuristic for only running k-means infrequently. The extreme case of $t = 20000$ corresponds to not running k-means, and hence not updating $\mathbf{A}$ at all, effectively random tying the parameters (based on their random initial values). Random tying is disastrous for small K, which simply cannot effectively cover the range of soft-tied

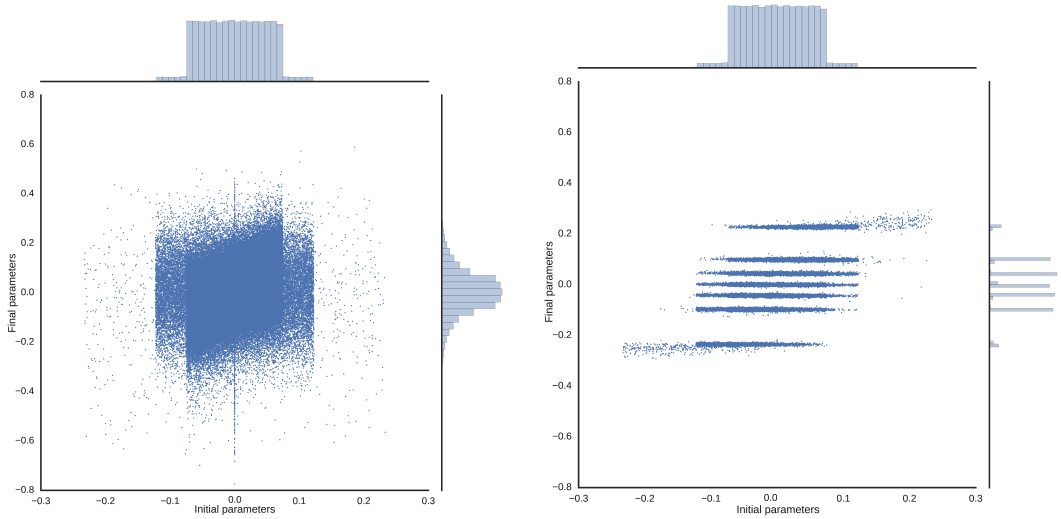
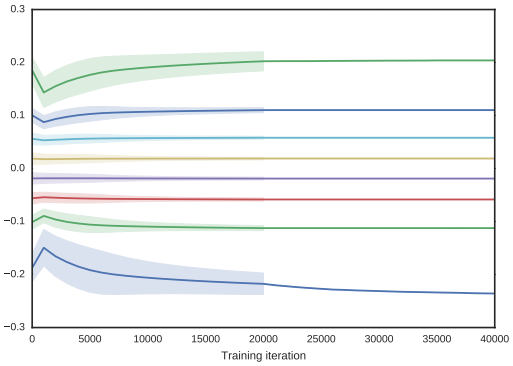
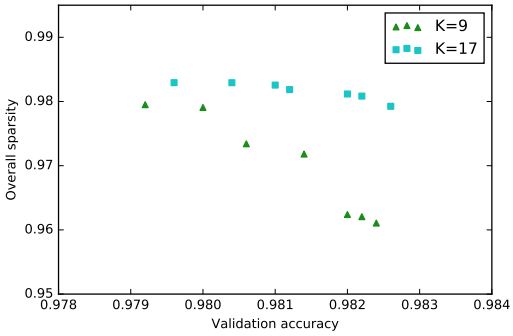


Figure 1: Joint histograms of parameters before and after training, with soft-tying APT (right) and without (left) an additional k-means loss. The parameters are initialized with scaled uniform distributions proposed in (Glorot & Bengio, 2010) and $K = 8$.



(a) Cluster centers, shaded by 1 standard deviation of points in the cluster. Note that, as a property of J (independent Gaussians), μ (unless degenerately initialized) will tend to oppose and repel each other, unlike in the case of GMM where they tend to merge.



(b) Sparsity versus accuracy (on the validation set) trade-off for APT, obtained by taking the Pareto frontier of typical hyper-parameter search results.

Figure 2: Experimental results for APT on LeNet-300-100.

parameters and induces significant quantization loss. Although specialized training methods, e.g., (Courbariaux et al., 2016), exist for training networks with $K = 2$ or 3, our current formulation of APT is not able to effectively quantize with such a small number of clusters.

4.2 Effect on Generalization

Recently, Zhang et al. (2016) observed that the traditional notion of model complexity associated with parameter norms captures very little of neural networks' generalization capability. Traditional regularization methods, like ℓ_2 (weight decay), do not introduce fundamental phase change in the generalization capability of deep networks, and bigger gains can be achieved by simply changing the model architecture rather than tuning regularization. The paper left open questions of how to *correctly* describe neural network's model complexity, in a way that accurately reflects the model's generalization. In this section, we explore a different notion of model complexity characterized by the number of free parameters in parameter-tied networks, where the tying is discovered through optimization.

Regularization	Average Test Error	Standard Deviation
Early Stopping	84.3	3.1
ℓ_1	86.8	2.3
ℓ_2	88.3	3.7
APT ($K = 5$)	88.6	3.4
APT ($K = 10$)	88.6	3.3
GMM ($K = 5$)	92.8	1.5
GMM ($K = 10$)	92.4	1.67

Table 1: Average test error for a variety of different regularization schemes for the bit-string shift detection problem.

To assess whether APT indeed improves model generalization, we compared the performance of APT against a GMM prior, on a toy problem where the latter approach was previously shown to be effective. We reproduced the original bit-string shift detection experiment as described in (Nowlan & Hinton, 1992), where the task was to detect whether a binary string was shifted left or right by one bit, with input being the pattern and its shifted version.[2] In the original experiment, all methods were stopped as soon as the training error becomes zero, which when used alone counts as an implicit form of regularization so we will refer to it as "early stopping". We compare APT with "early stopping", ℓ_2 norm penalty, APT, and GMM prior, using a model selection criteria in which we consider a model's performance at the first iteration where its error on the training set becomes zero. After some initial tuning, we found a common set of gradient step sizes, $\{0.01, 0.03, 0.1, 0.3\}$, and maximum training budget (fifty thousand iterations) such that all methods considered could converge to 0 training error. For ℓ_2, APT, and GMM, we set the search grid for the regularization parameter to $\{1e-7, 1e-6, ..., 1e-3\}$, higher values of λ all resulted in poor convergence, and for GMM in particular, which uses four individual learning rates for the network parameters, mixture means, mixture variances, and mixture proportions, we brute-force searched all combinations for each learning rate in the common set of step sizes.

The results are presented in Table 1. As was observed by Zhang et al. (2016), while all methods have essentially zero error on the training data, there is only a mild effect on the test error due to the particular choice of regularization for a fixed network structure. This suggests that parameter tying and sparsity do not act strongly to improve regularization performance.

4.3 Sparsity and Compression Results

We compare sparse APT against other neural network compression or pruning methods, including Deep Compression (DC) (Han et al., 2016), SWS (K. Ullrich, 2017), Bayesian Compression (BC) (Louizos et al., 2017), and Sparse Variational Dropout (Sparse VD) (Molchanov et al., 2017) on LeNet architecture on MNIST. We use the standard MNIST train/test data split (LeCun et al., 1998) and form a validation set of 5000 examples from training data. In addition, to illustrate the effectiveness and scalability of our approach, we also experimented with the VGG-11 network on the CIFAR-10 dataset, which contains 9.8 million parameters. Since VGG-11 was originally designed for ImageNet, we adapted it to CIFAR-10 by replacing its 1000-class output layer with a 10-class one. We were not able to experiment on ImageNet due to hardware resource constraints. We trained our networks from scratch[3] without any data augmentation or other regularization(e.g., batch normalization or dropout) . We first perform soft-tying training with a sparse APT penalty for a fixed budget of iterations and then switch to hard-tying until the validation error increases consecutively or a separate budget is reached.

The results of our experiments are presented in Table 2. There, we report the error of the trained network on the test set, the fraction of non-zero weights, a pruning score based on the CSC sparse matrix encoding, and max compression as in (K. Ullrich, 2017). Note that Louizos et al. (2017) eval-

[2](Nowlan & Hinton, 1992) considered an implementation of GMM prior using simple SGD with momentum, so we also used it in this experiment instead of Adadelta.

[3]We observed similar accuracy and compression levels by initializing to pretrained networks.

Network	Method	Error %	$\frac{\lvert w \neq 0 \rvert}{\lvert w \rvert}\%$	Pruning	Max. Compression
LeNet-300-100	DC	1.6	8.0	6	40
	SWS	1.9	4.3	12	64
	Sparse VD	1.8	2.2	21	113
	BC-GNJ	-	10.8	9 (1.8)	58 (1.8)
	BC-GNS	-	10.6	9 (1.8)	59 (2.0)
	APT	2.0	1.7	29	143
LeNet-5-Caffe	DC	0.7	8.0	6	39
	SWS	1.0	0.5	100	162
	Sparse VD	1.0	0.7	63	365
	BC-GNJ	-	0.9	108 (1.0)	572 (1.0)
	BC-GNS	-	0.6	156 (1.0)	771 (1.0)
	APT	1.0	0.5	100	346
VGG-11	APT	21	3.5	14	123

Table 2: Comparison of APT ($K = 17$) with other compression and/or sparsity-inducing methods.

uate the compression criteria separately for each of their variations of BC, instead of with a single trained network; therefore, we report their errors in parentheses following the sparsity/compression statistics, as was done by Louizos et al. (2017). The maximum compression scores for all methods (except APT) are obtained by clustering the final weights into 32 clusters (this achieved the best compression rate (Louizos et al., 2017)). For APT, all reported scores were achieved with $K = 17$ clusters, resulting in 16 distinct non-zero parameter values.

Overall, we observe that APT outperforms most (or all) of the competitors on each data set with the exception of the BC methods in terms of max compression on LeNet-5, this occurs even though APT manages to find a sparser solution than both BC variants. The explanation for this is that the maximum compression score uses a sparse matrix representation of the weight matrix and then further compresses the matrix using Huffman coding. As Huffman coding performs best with non-uniform distributions, the primary difference between the APT and the BC solutions is that the BC solutions do not return many equal sized clusters. While our main goal was to achieve sparsity with a small number of parameters, if maximum compression is desired, the variances of the independent Gaussian prior could be tuned to induce a significantly more non-uniform distribution which may yield higher compression rates on LeNet-5.

From these experiments, we observed that APT can learn a good solution from a random initialization using about the same number of iterations as no-regularization/ℓ_2. This distinguishes it from other compression methods, which can suffer significant accuracy loss ($\geq 2\%$) if not initialized using a pretrained network. This appears to be a common problem, especially for Bayesian methods (e.g., Sparse VD and BC). The SWS method, which achieves comparable levels of sparsity to APT on LeNet-5, also starts from a pretrained network. Despite this, we found that, as previously observed by K. Ullrich (2017), the SWS method requires significant computation time for hyperparameter tuning to yield good solutions.

More generally, APT can be used to trade-off between accuracy and sparsity depending on the application. This can be done using a validation set to select the desired performance trade-off. Figure 2b illustrates part of the sparsity/accuracy trade-off curve for two different values of K. When $K = 9$, sparsity can be increased at a significant loss to accuracy, while at $K = 17$, additional sparsity can be gained with only moderate accuracy loss. In practice, selecting the smallest value of K that exhibits this property is likely to yield good accuracy and compression. In fact, the existence of such a K provides further evidence that, for a fixed structure, sparsity and quantization do not improve generalization performance.

REFERENCES

M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

W. Chen, J. Wilson, S. Tyree, K. Weinberger, and Y. Chen. Compressing neural networks with the hashing trick. In *International Conference on Machine Learning (ICML)*, pp. 2285–2294, 2015.

M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio. Binarized neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 4107–4115, 2016.

L. Deng, G. E. Hinton, and B. Kingsbury. New types of deep neural network learning for speech recognition and related applications: An overview. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pp. 8599–8603. IEEE, 2013.

M. Denil, B. Shakibi, L. Dinh, M. Ranzato, and N. de Freitas. Predicting parameters in deep learning. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 2148–2156, 2013.

E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 1269–1277, 2014.

J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research (JMLR)*, 12(Jul):2121–2159, 2011.

R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 580–587, 2014.

X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 249–256, 2010.

X. Glorot, A. Bordes, and Y. Bengio. Domain adaptation for large-scale sentiment classification: A deep learning approach. In *Proceedings of the 28th international conference on machine learning (ICML)*, pp. 513–520, 2011.

Y. Gong, L. Liu, M. Yang, and L. Bourdev. Compressing deep convolutional networks using vector quantization. In *International Conference on Learning Representations (ICLR)*, 2015.

S. Han, J. Pool, J. Tran, and W. J. Dally. Learning both weights and connections for efficient neural networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems (NIPS)*, pp. 1135–1143, 2015.

S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. In *International Conference on Learning Representations (ICLR)*, 2016.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

G. E. Hinton, L. Deng, D. Yu, G. E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

M. Jaderberg, A. Vedaldi, and A. Zisserman. Speeding up convolutional neural networks with low rank expansions. In *Proceedings of the British Machine Vision Conference (BMVC)*, 2014.

M. Welling K. Ullrich, E. Meeds. Soft weight-sharing for neural network compression. In *International Conference on Learning Representations (ICLR)*, 2017.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 1097–1105, 2012.

B. Kulis and M. I. Jordan. Revisiting k-means: New algorithms via Bayesian nonparametrics. In *Proceedings of the 29th International Conference on Machine Learning (ICML)*, pp. 513–520, July 2012.

Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 396–404, 1990.

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.

C. Louizos, K. Ullrich, and M. Welling. Bayesian compression for deep learning. In *Advances in Neural Information Processing Systems (NIPS)*, 2017.

D. Molchanov, A. Ashukha, and D. Vetrov. Variational dropout sparsifies deep neural networks. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pp. 2498–2507, International Convention Centre, Sydney, Australia, Aug 2017.

S. J. Nowlan and G. E. Hinton. Simplifying neural networks by soft weight-sharing. *Neural computation*, 4(4):473–493, 1992.

H. Wang and M. Song. Ckmeans. 1d. dp: optimal k-means clustering in one dimension by dynamic programming. *The R journal*, 3(2):29, 2011.

W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems (NIPS)*. 2016.

M. D. Zeiler. ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701, 2012. URL `http://arxiv.org/abs/1212.5701`.

C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals. Understanding deep learning requires rethinking generalization. 2016.

A ADDITIONAL EXPERIMENTAL RESULTS

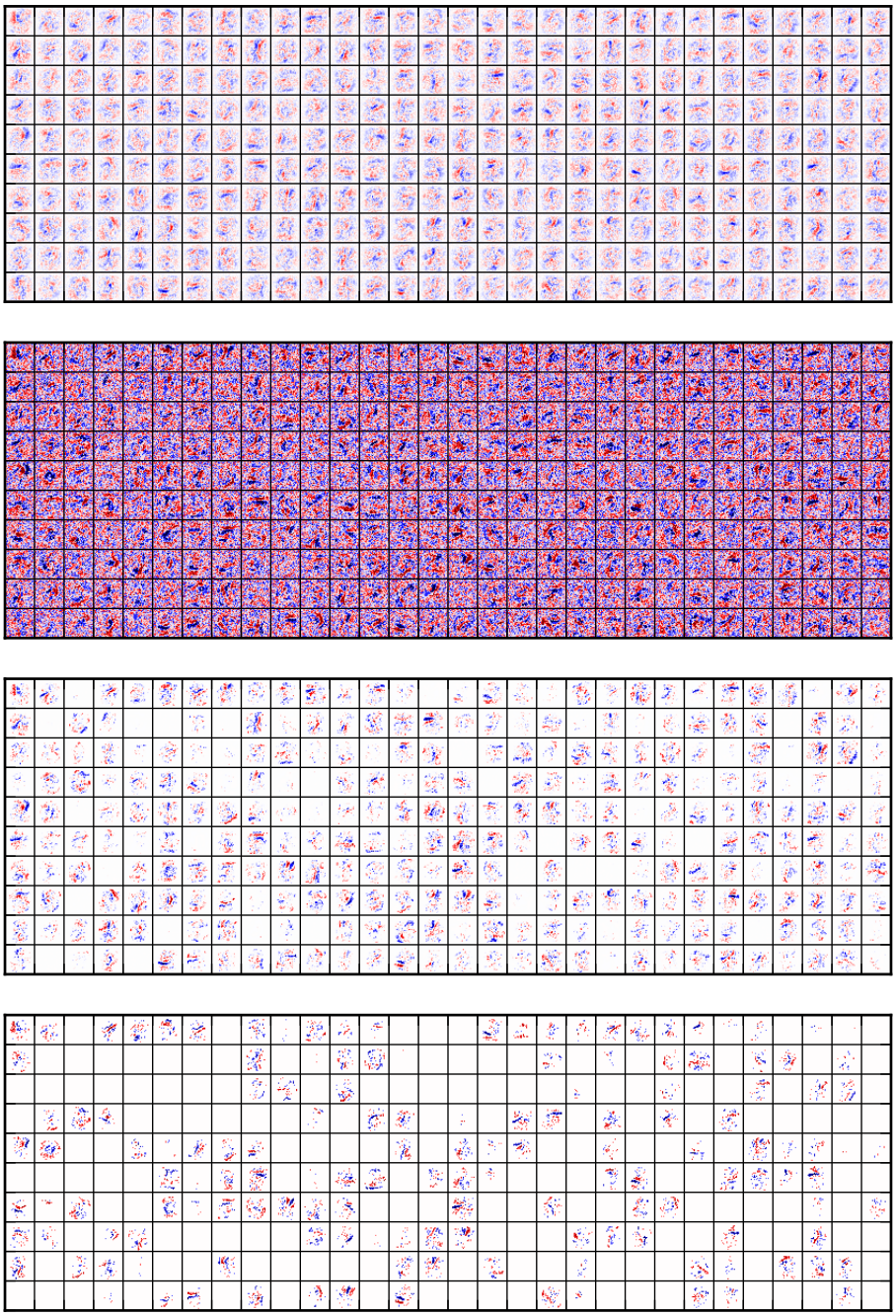


Figure 3: Visualization of first layer weights in LeNet 300-100, learned with L2, APT-ℓ_1 (K=8),ℓ_1, and APT+ℓ_1 (K=9, sparsity=0.96, cr=90), all starting from the same initialization and achieving similar error rates (1.8%, 1.81%, 1.83%, 1.9%). All colors are on an absolute scale from -0.3 to 0.3 centered at 0. Note that APT by itself does not pursue small-norm solutions (in fact, does the opposite for small K).

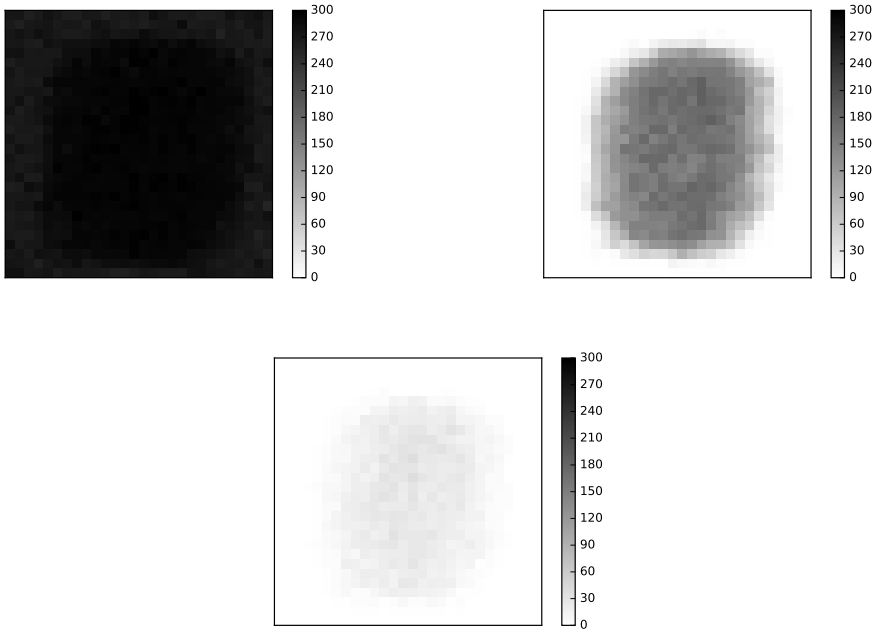


Figure 4: Comparing the the number of input nodes pruned by L2, L1, and sparse APT, on LeNet-300-100. In the network trained with L2 and L1, we consider connections with absolute value less than $1e-3$ pruned, for illustration.