ADAPTIVE WEIGHT SPARSITY FOR TRAINING DEEP NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

We introduce *adaptive weight sparsity*, an algorithm that allows a neural network to learn a sparse connection pattern during training. We demonstrate that the proposed algorithm shows performance benefits across a wide variety of tasks and network structures, improving state-of-the-art results for recurrent networks of comparable size. We show that adaptive weight sparsity outperforms traditional pruning-based approaches to learning sparse configurations on convolutional and recurrent networks. We offer insights into the algorithm's behavior, demonstrating that training-time adaptivity is crucial to the success of the method and uncovering an interpretable evolution toward small-world network structures.

1 INTRODUCTION

It is widely known that the structure of a neural network influences its performance. Contemporary neural networks almost exclusively use fixed connection patterns, many of which are defined by structured sparse connectivity. For instance, convolutional networks (LeCun et al., 1998) leverage block sparse connectivity for efficiency. Recent work on separable convolutions incorporates even more sparsity (Chollet, 2016). Even feed-forward network architectures may be considered upper-triangular block-sparse embeddings within a recurrent framework.

In our work we study a mechanism that allows the learning of general sparse network structures. Specifically we study the effects of coupling gradient-based training with network structure evolution. We introduce *adaptive weight sparsity*, an optimization method that learns to arrange connections within a network. This approach allows more efficient use of an allotted parameter budget. Controlling for number of nonzero parameters, there is no reason to expect dense connection patterns to be the most efficient arrangement. In fact, the success of structured sparse architectures suggests the opposite.

There are several benefits of adapting network structure during training: (1) An appropriate learning rule can progress incrementally towards structures that more efficiently process information. (2) The ability to remove and add connections allows more degrees of freedom in the optimization procedure for a fixed computational budget. Additional degrees of freedom increase the likelihood of converging to better results. (3) Constraining which network connections are active and trainable at any given time can allow learning to progress more rapidly and act as a regularizer.

The adaptive weight sparsity method supplements a given neural architecture with a tunable sparse structure (details in Section 3). A guided discrete search on the structure is coupled with standard gradient descent on the parameter matrices. Our approach is a novel method of combining these two learning problems. It allows the gradient to guide the combinatorial structure search, selecting which structural mutations should be made. In turn, the choice of connection structure supresses many would-be gradient entries. In this manner, the learned network structure induces a biased estimate of an implied dense gradient. These mutually influential processes ultimately lead the network toward better solutions.

We demonstrate that this approach improves the performance of a number of existing models. The results from our method are also well-suited to producing sparse networks for efficient inference in field deployments. In particular, adaptive weight sparsity outperforms pruning, the principal existing technique for improving the efficiency of inference. Furthermore, we expect weight sparsity during training to become increasingly important as improved general handling of sparse linear algebra

operations and emerging specialized hardware (Parashar et al., 2017; Nurvitadhi et al., 2017) provide additional acceleration for such methods.

2 Related Work

Recent work shows the promise of sparse neural network topologies. Separable convolutions (Chollet, 2016; Zhang et al., 2017) use additional block sparsity in the channel and spatial dimensions to achieve performance and efficiency gains beyond regular convolutions. Changpinyo et al. (2017) have shown that random fixed sparse layers are more parameter-efficient than dense layers in convolutional networks.

Another line of work focuses on pruning of trained networks for computationally efficient inference (Parashar et al., 2017). For example, Han et al. (2015b) train a dense network to completion, then remove weights with the lowest magnitudes, and finally perform a fine-tuning training step. Earlier work by Hassibi & Stork (1993) suggests the use of second-order derivatives for determining weight significance. Alternative pruning techniques have been investigated by Han et al. (2015a), Srinivas & Babu (2015), and Wen et al. (2016). Mariet & Sra (2015), Murray & Chiang (2015), and Hu et al. (2016) have considered pruning approaches at the neuron, rather than connection, level. Collins & Kohli (2014) and Srinivas et al. (2017) use regularization techniques to induce a sparse structure during training. Denil et al. (2013) developed a model that learns only a small number of connection parameters and predicts the rest. Adaptive weight sparsity is notably distinct from these approaches in that it maintains sparsity throughout training and adapts the connection structure continually.

Structure-space search has been attempted within a reinforcement learning framework in which an agent is trained to select neural architectures (Zoph & Le, 2016; Cai et al., 2017).

3 Adaptive Weight Sparsity

The adaptive weight sparsity algorithm is designed to prune small weights and activate weights in new locations. We introduce a binary $\{0, 1\}$ mask variable for every parameter in a model. In fact, the algorithm admits more efficient implementation by storing matrices in compressed sparse row format, but we adopt this notation here for convenience. When the mask is 0, the corresponding weight is not used by the model, and we say it is inactive.

The model is initialized by setting a target fraction α of the weights to be active. Active weights are initialized according to the initialization procedure of the baseline model. During the course of training we keep track of the active weights (a fraction a of all possible weights) and after every training step we adjust the sparsity structure. Inactive weights are set to zero at all times.

We apply the following procedure to each weight matrix of the network. In the pruning stage, we randomly sample a fraction β_1 of active weights and compute the mean $\overline{w}$ of their magnitudes. Note that sampling allows for computational efficiency but adds no expressive power to the algorithm, which is already probabilistic. We apply a sigmoid-based probability distribution to determine if an active weight with magnitude $|w|$ in the sample should be pruned by setting its mask to 0:

$$P(1{\to}0|w, \overline{w}, a) = \frac{1}{1 + \exp[g(\frac{|w|}{k\overline{w}} - 1 - s(a - \alpha))]}$$

Here the parameter s controls a term that biases the process to keep the fraction of active weights a close to its target value α. Absent this bias factor, k controls the pruning threshold, resulting in deactivations of weights of magnitude $k\overline{w}$ with probability 0.5. Finally, g is a smoothing term and in the limiting case of large g the probability distribution becomes a binary step function that deterministically forces pruning of weights whose magnitude falls below the threshold.

We introduce a similar stochastic procedure for activating new weights. We sample a fraction β_0 of the inactive parameters to obtain a set $\tilde{\mathbf{w}}$ of new candidate weight locations. We compute the average magnitude $\overline{\nabla}$ of the gradient $\nabla_w J$ of our loss with respect to each $w \in \tilde{\mathbf{w}}$ and activate each

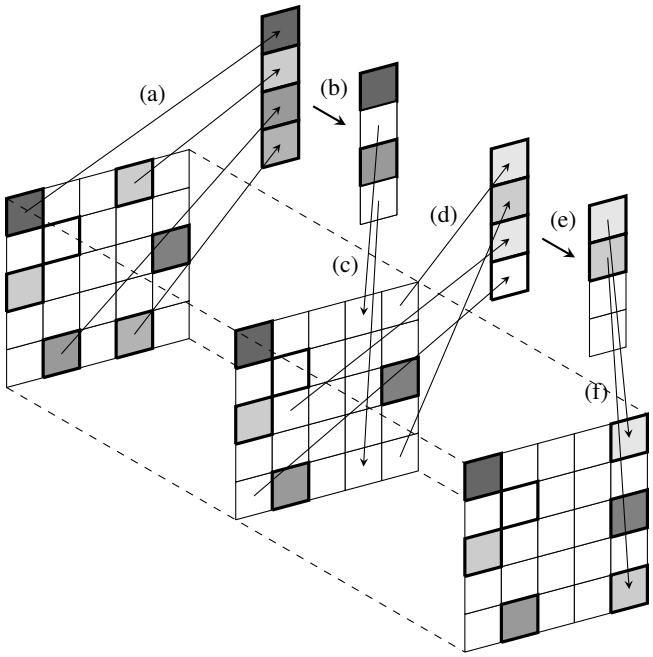

Figure 1: Steps of the adaptive sparsity algorithm: (a) sample active weights, (b) compute deactivating probability and identify pruning targets, (c) apply deactivation, (d) sample inactive weights and compute their gradients, (e) calculate activation probability and identify initialization targets, (f) initialize new weights.

w with probability

$$P(0\to 1|\nabla_w J, \overline{\nabla}, a) = \frac{1}{1 + exp[g(\frac{|\nabla_w J|}{l\overline{\nabla}} - 1 + s(a-\alpha))]} \,.$$

Here the parameter l determines the threshold of gradient magnitudes that force initialization of new weights. Activated weights are initialized with small values according to the gradients, with noise drawn from a uniform random interval of radius r:

$$w_{new} = c_g \frac{\partial J}{\partial w} + n, \quad n \sim U(-r, r) \,.$$

4 Experimental Results

We experiment with varying all hyperparameters described in Section 3. For simplicity and concreteness, in the experiments described in this paper we focus on the effect of α that controls the fraction of active weights and k that determines the rate of structural mutation. At specified epoch boundaries (N a multiple of epoch size) we analyze the full set of active weights ($\beta = 1$) and prune and initialize weights deterministically when corresponding thresholds are met ($g \to \infty$) initializing new values randomly ($c_g = 0$) with radius $r = 2k\overline{w}$. We also keep the number of nonzero parameters constant fixed in the course of single experiment ($s \to \infty$), and between experiments with different values of α by adjusting layer dimensions accordingly.

We prioritize obtaining results on a wide variety of common models and test our method on a fully connected network, a variational autoencoder (Kingma & Welling, 2013), an LSTM as in Karpathy et al. (2016), a small convolutional network, and the ResNet-110 model (He et al., 2016). We use ReLU nonlinearities and minimize cross-entropy loss using the ADAM (Kingma & Ba, 2015) optimizer and minibatches of size 128 with a learning rate of 0.001. Our model details are outlined in Table 1.

Table 1: Experimental Details

Model	Dataset	Layer Sizes (base model)	Training Epochs	Adaptive Frequency (epochs)	Notes
Fully Connected	MNIST	64-64-64	20	2	
Variational Autoencoder	MNIST	256-10-256	500	5	Used RMSProp, batch-size 256.
LSTM	War and Peace	512	100	1	Used early stopping. Dropout 0.5. 100 time steps.
Convolutional Network	CIFAR-10	32-64-64	200	5	3x3 filters. 10^{-3} L2 weight decay.
ResNet	CIFAR-10			5	See He et al. (2016)

We compare the adaptive weight sparsity method to both the baseline case of a fully dense network with the same number of parameters per layer and with a sparse control in which the location of active weights is random and fixed. The results are detailed below and are illustrated in Table 2 and Figure 2.

4.1 Adaptive Sparsity

Adaptive weight sparsity outperforms the baseline and the random sparse control across all models. The results are particularly pronounced for the LSTM network, for which adaptive weight sparsity outperforms the best published result for a network with the same number of parameters (Karpathy et al., 2016). On all experiments, a k value of 0.5 or 0.25 yields accuracy improvements compared to the baseline, giving a useful heuristic to facilitate hyperparameter tuning. Larger k values typically impair model performance, as too many weights are replaced at each epoch. Performance approaches that of the fixed random sparse control as $k \to 0$, as expected.

Interestingly, random sparse networks also outperform dense baselines somewhat on several experiments. Decreasing α reduces performance after a certain point – this is unsurprising, as allowing $\alpha \to 0$ tends towards network topologies with no path from input activations to outputs. The optimal α value varies with the model, though $\log \alpha = -0.5$ gives better-than-baseline performance across all models tested. It is notable that fixed random sparsity is more susceptible to poor performance with small alpha values than adaptive sparsity. This suggests that adaptivity makes a model more robust to network size and overall connection sparsity. The result is reasonable, as given enough training time, adaptive weight sparsity can produce connected paths and clusters of active weights within a sparse network.

4.2 Generalization and Training Accuracy

Our results show that adaptive sparsity improves generalization: Across all models, when we evaluate instances with the same loss value on the training set (by choosing snapshots from different points in the training process), those trained with adaptive sparsity consistently give better validation accuracy than the baseline model.

However, adaptive sparsity outperforms the other training methods on *both* the training and validation sets. It trains more quickly and achieves better accuracy. We conclude that adaptive weight sparsity has benefits beyond regularization. One notable exception to this pattern is our LSTM model which performs slightly worse than the baseline on the training set but significantly better on the validation set.

4.3 Comparison with Wide Dense Models

It is informative to compare a network trained with adaptive weight sparsity to a dense network with an equal number of *activations*. In several cases we observed the adaptive sparse model outperforms this dense counterpart despite its parameter deficit. This occurs in our tests with the CNN and with the LSTM for $\log \alpha = -0.5$. We attribute this gain to better regularization.

In other cases, the adaptive sparse models yield performance between that of the baseline and wide dense models, consistent with an interpretation (outlined in Section 5) that adaptive sparsity captures part of the expressivity of a wider network.

Table 2: Experimental Results

$-\log \alpha$	Random Fixed Sparse	Random Sparse Channel-wise	Adaptive Sparse (best k)	Pruned	Wide Dense (more params)
Fully Connected (% Error)	Baseline: 2.9				
0.5	2.6		2.3 (0.25)	2.4	2.2
1.0	2.5		2.3	1.8	1.9
1.5	2.6		2.2 (0.25)	**1.7**	1.9
2.0	2.5		2.3 (0.25)	1.9	1.9
Variational Autoencoder (Loss)	Baseline: 105.9				
0.5	105.5		103.2 (0.25)	**102.1**	100.8
1.0	108.2		103.5 (0.25)	104.4	100.7
LSTM (Cross-entropy Loss)	Baseline: 1.178		Best published result with same # parameters: 1.161		
0.5	1.154		**1.114 (0.5)**	1.122	1.129
1.0	1.171		1.144 (0.5)	1.122	1.098
Convolutional Network (% Error)	Baseline: 18.7				
0.5	18.4	19.4	**17.0 (0.25)**	17.8	17.6
1.0	19.1	20.5	17.3 (0.5)	18.3	17.9
ResNet-110 (% Error)	Baseline: 6.5				
0.5	6.4	6.2	**5.9 (0.25)**	6.4	5.65
1.0	6.2	6.2	**5.9 (0.25)**	6.5	5.1

Above: experimental results on validation sets. The bold figure indicates best-performing model for the given parameter budget. Baseline refers to the model with no sparsity or adaptivity. Note that "wide dense" figures refer to networks scaled by α^{-1} but not sparsified – these are given for reference, but contain more parameters than other models on the same row. Best-performing values of k for a given α are indicated parenthetically next to adaptive sparsity figures.

4.4 Comparison with Pruning

We compare adaptive weight sparsity to pruning, a technique that also results in a sparse model but by a different method. For comparison we use the method of Han et al. (2015b), pruning the smallest weights after training a dense network and subsequently fine-tuning the sparse network controlling for the final number of active parameters. Adaptive weight sparsity is superior for the more complicated models tested (convolutional, ResNet, and LSTM). The simplest two models (fully-connected and VAE on MNIST) perform better using the pruning method. This suggests that the optimization benefits offered by adaptive sparsity are effective at guiding optimization of sufficiently complex models.

5 Analysis of Weight Adaptivity

We distinguish between effects of the network topology learned through adaptivity and the effects of the adaptive process itself.

5.1 Role of Adaptivity

We find evidence that the adaptivity process itself is necessary to obtain the benefits of adaptive sparsity. To demonstrate this we take the sparsity pattern resulting from a training session with adaptive sparsity and use this as a fixed sparse structure for a new network trained from scratch (see Figure 3). Doing so eliminates much of the benefit of adaptive sparsity. In no instances does the resulting model achieve the same performance as one trained with adaptive sparsity. Across all our feedforward models, the learned topology accounts for roughly 30% of the observed improvement of adaptive sparsity over random fixed sparsity (and 0% on the recurrent network). We conclude that adaptive weight sparsity succeeds as an optimization method for other reasons in addition to the topology it learns.

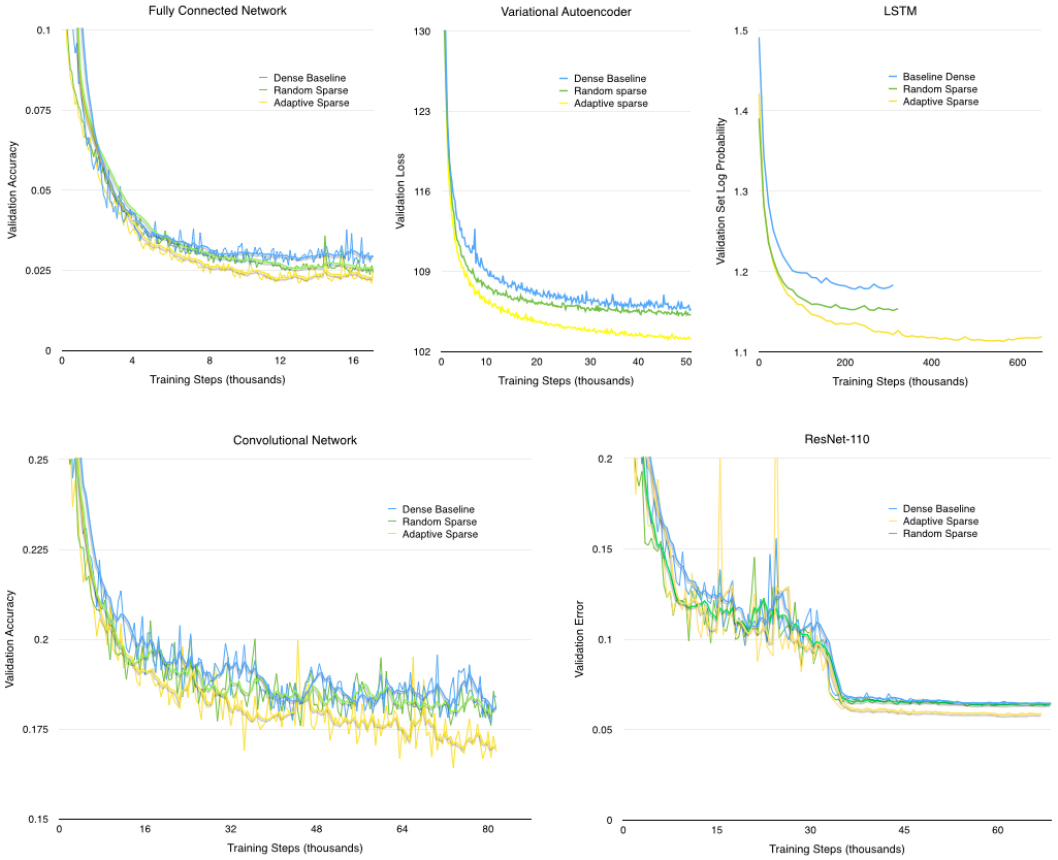

Figure 2: Results for best choices of α and the adaptive threshold k. Controlling for number of parameters, adaptive weight sparsity yielded the best results across all models. Note that the LSTM model employed early stopping. Best viewed in color.

These results are not too surprising, as adaptive sparsity produces connection patterns that are dependent on the associated parameter values. Dense local clusters arise in response to their utility, assuming learned roles the particular network being trained. In this sense, fixed sparse structures trained with gradient descent will always be sub-optimal compared to adaptive structures. We observe this effect to be most pronounced in an LSTM network where it is amplified by the recurrent temporal paths through the network.

We now consider how an adaptivity mechanism is able to overcome the limitations of fixed network structures. Consider adaptive sparsity as an optimization technique. The potential for adding and removing weights corresponds to the ability to periodically move in extra dimensions of parameter space. Allowing the connection pattern to adjust over time accounts for the possibility that different degrees of freedom may be needed at different stages in the optimization process. Furthermore, this adjustment allows for more efficient use of model parameters, as is empirically observed in the form of a decrease in low-strength weights during training (Figure 4).

These new degrees of freedom are obtained at the cost of pinning the smallest model weights to exactly zero. This costs a small amount in accuracy at the time of pruning in exchange for exploration of model structure space. Over time, this allows the model to converge to a point nearby a theoretical convergence point for the corresponding dense model. Hence, weight adaptivity allows a sparse model to capture, to some approximation, the expressive power of an equally wide but dense model. This interpretation suggests it may be useful to slowly increase model density over training time as an accelerated and better-regularized means for finding dense solutions. We leave this possibility for follow-up work.

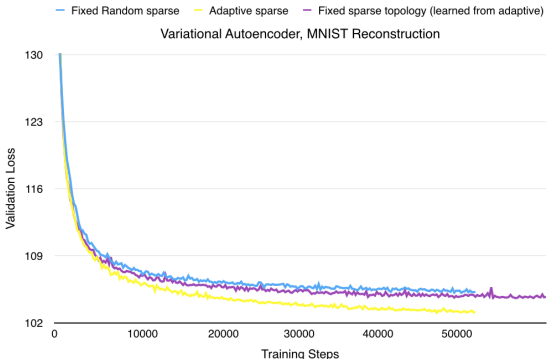

Figure 3: A representative example of training a network from scratch using the connection pattern learned by adaptive sparsity, with $\log \alpha = -0.5, k = 0.25$. On all models, this experiment yielded less improvement over random sparsity than full adaptivity, and sometimes none at all. Best viewed in color.

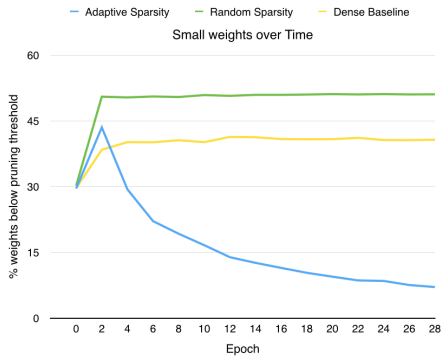

Figure 4: Number of low-strength weights (relative to mean) over time on a representative layer of the multi-layer perceptron model. For the non-baselines $\log \alpha = -2$. For the adaptive run, $k = 0.5$. Best viewed in color.

5.2 Learned Connectivity Patterns

We also investigate what general trends arise in the sparsity patterns learned via the adaptive sparsity mechanism. We are in part motivated by our results (e.g. Figure 3) which do show some performance improvements attributable to the final learned topologies themselves. Furthermore, we note that a connection structure can be important for model performance even if it does not give good results when trained from scratch. We consider the connectivity patterns as a graph and employ standard graph-theoretic metrics for analysis. We chart the evolution of four metrics over training time. The first, algebraic connectivity, is formally the second-smallest eigenvalue of the Laplacian matrix of the graph. Intuitively, it indicates how "connected" the graph structure is, roughly corresponding to how much modification would be required to split it into multiple disconnected components (a disconnected graph has an algebraic connectivity of zero) (Chung, 1997). Second, we take the GINI coefficient – a measure of inequality in a distribution – of the nodes of the graph by their degree (number of connections in and out) (Gini, 1921). A higher GINI coefficient indicates a tendency of certain nodes to accumulate a disproportionate number of connections. Third, we track the global clustering coefficient, which is proportional to the fraction of possible triangles present in a graph; this indicates the tendency of graph nodes to form clusters (Wasserman & Faust, 1994). Finally, we measure the average length of the shortest path between arbitrary nodes in the graph, which measures how quickly information may be propagated through the graph.

These results are most easily interpretable for a single-layer recurrent model, since its structure allows for all possible connections between nodes. Hence, we present results for our one-layer LSTM model, considering the four sets of self-connections (the output of the recurrent layer feeds into the input gate, the recurrent input, the forget gate, and the output gate of the next time step) as separate graphs.

The data (see Figure 5) show consistent trends: connectivity of the graphs decreases, node degree inequality increases, clustering increases, and shortest path length remains constant. The first two phenomena suggest a graph fragmenting into subregions dominated by hubs with many connections. The combination of increased clustering and constant shortest path length indicates, by definition, an evolution toward what is known as a small-world graph (Watts & Strogatz, 1998). As noted in Latora & Marchiori (2001), small-world network structures commonly arise in social, engineering, and biological contexts, and are characterized by high efficiency in propagating information. They are particularly notable for their prevalence in the brain (Bassett & Bullmore, 2006; Morelli et al., 2004; Stam, 2004). Future study may elucidate further the significance of these trends, particularly in light of of recent work by Bölcskei et al. (2017) establishing theoretical efficiency bounds on sparse neural network structures, and by Grohs et al. (2016) demonstrating the emergence of multiscale behavior in neural networks (consistent with a locally connected small-world structure). The difference in behavior among the four types of weights in the LSTM layer also remains to be explored. For now,

Figure 5: Evolution of a variety of graph metrics during the training of an LSTM model with weight adaptivity. Best viewed in color.

we may conclude that weight adaptivity naturally leads to loosely connected network topologies with tight clusters and permits efficient communication across the network despite this decrease in connectivity.

6 Conclusion

We have described a framework for adapting sparse connection patterns during training, and we have shown empirically that it outperforms densely connected networks with the same number of parameters for a wide variety of models and tasks. We have also provided evidence that adaptive weight sparsity outperforms traditional pruning methods. We attribute the success of the method both to more efficient learned model structures, which we find to resemble small-world networks, and to the role of reconfiguring weights during training as an optimization technique.

Future work could more fully explore the space of the adaptive weight sparsity algorithm, analyzing the effect of hyperparameters other than α and k. One might also experiment with extensions to the algorithmic framework, considering deactivation probability functions that depend on time since a parameter's last pruning, or on other factors not used in this work. Additionally, the improvements observed by applying adaptive weight sparsity after initially blocking convolutional weights channel-wise suggest that experimenting with different initial sparse graph structures could prove fruitful.

REFERENCES

Danielle Smith Bassett and ED Bullmore. Small-world brain networks. *The neuroscientist*, 12(6): 512–523, 2006.

Helmut Bölcskei, Philipp Grohs, Gitta Kutyniok, and Philipp Petersen. Optimal approximation with sparsely connected deep neural networks. *arXiv preprint arXiv:1705.01714*, 2017.

Han Cai, Tianyao Chen, Weinan Zhang, Yong Yu, and Jun Wang. Reinforcement learning for architecture search by network transformation. *arXiv preprint arXiv:1707.04873*, 2017.

Soravit Changpinyo, Mark Sandler, and Andrey Zhmoginov. The power of sparsity in convolutional neural networks. *arXiv preprint arXiv:1702.06257*, 2017.

François Chollet. Xception: Deep learning with depthwise separable convolutions. *arXiv preprint arXiv:1610.02357*, 2016.

Fan RK Chung. *Spectral graph theory*. Number 92. American Mathematical Soc., 1997.

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

Misha Denil, Babak Shakibi, Laurent Dinh, Nando de Freitas, et al. Predicting parameters in deep learning. In *Advances in Neural Information Processing Systems*, pp. 2148–2156, 2013.

Corrado Gini. Measurement of inequality of incomes. *The Economic Journal*, 31(121):124–126, 1921.

Philipp Grohs, Sandra Keiper, Gitta Kutyniok, and Martin Schäfer. α-molecules. *Applied and Computational Harmonic Analysis*, 41(1):297–336, 2016.

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

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*, pp. 1135–1143, 2015b.

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

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.

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

Andrej Karpathy, Justin Johnson, and Li Fei-Fei. Visualizing and understanding recurrent networks. *ICLR Workshop*, 2016.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *ICLR 2015*, 2015.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Vito Latora and Massimo Marchiori. Efficient behavior of small-world networks. *Physical review letters*, 87(19):198701, 2001.

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

Zelda Mariet and Suvrit Sra. Diversity networks. *arXiv preprint arXiv:1511.05077*, 2015.

Luis G Morelli, Guillermo Abramson, and Marcelo N Kuperman. Associative memory on a small-world neural network. *The European Physical Journal B-Condensed Matter and Complex Systems*, 38(3):495–500, 2004.

Kenton Murray and David Chiang. Auto-sizing neural networks: With applications to n-gram language models. *arXiv preprint arXiv:1508.05051*, 2015.

Eriko Nurvitadhi, Ganesh Venkatesh, Jaewoong Sim, Debbie Marr, Randy Huang, Jason Ong Gee Hock, Yeong Tat Liew, Krishnan Srivatsan, Duncan Moss, Suchit Subhaschandra, et al. Can fpgas beat gpus in accelerating next-generation deep neural networks? In *FPGA*, pp. 5–14, 2017.

Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel Emer, Stephen W Keckler, and William J Dally. Scnn: An accelerator for compressed-sparse convolutional neural networks. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pp. 27–40. ACM, 2017.

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

Suraj Srinivas, Akshayvarun Subramanya, and R Venkatesh Babu. Training sparse neural networks. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2017 IEEE Conference on*, pp. 455–462. IEEE, 2017.

Cornelius J Stam. Functional connectivity patterns of human magnetoencephalographic recordings: a small-worldnetwork? *Neuroscience letters*, 355(1):25–28, 2004.

Stanley Wasserman and Katherine Faust. *Social network analysis: Methods and applications*, volume 8. Cambridge university press, 1994.

Duncan J Watts and Steven H Strogatz. Collective dynamics of'small-world'networks. *nature*, 393 (6684):440, 1998.

Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems*, pp. 2074–2082, 2016.

Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *arXiv preprint arXiv:1707.01083*, 2017.

Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.