NEURAL NETWORKS WITH BLOCK DIAGONAL INNER PRODUCT LAYERS

Anonymous authors
Paper under double-blind review

ABSTRACT

Two issues continue to burden deep learning researchers and users: (1) neural networks are cumbersome tools that grow with the complexity of the learning problem, and (2) the activity of the fully connected, or inner product, layers remains mysterious. We make contributions to these two difficulties by considering a modified version of the fully connected layer we call a block diagonal inner product layer. These modified layers have weight matrices that are block diagonal, turning a single fully connected layer into a set of densely connected neuron groups. This method condenses network storage and speeds up the run time without significant adverse effect on the testing accuracy, thus offering a new approach to solve the first problem. Comparing the change in variance and singular values of the weights through training in a layer when varying the number of blocks gives insight into the second problem. The ratio of the variance of the weights remains constant throughout training. That is, the relationship in structure is preserved in the final parameter distribution. We observe that trained inner product layers have structure similar to that of truly random matrices with iid entries, and that each block in a block inner product layer behaves like a smaller copy, giving a better understanding of the nature of inner product layers.

1 INTRODUCTION

Today, it is well known that larger neural networks can better represent complex data and hence achieve higher accuracy than smaller networks (Hornik et al., 1989; Simonyan & Zisserman, 2014; Sermanet et al., 2014). While larger networks are more capable than their smaller counterparts, their size consumes significant storage and computational resources. Ideally, efforts to reduce memory requirements would also lessen computational demand, but often these competing interests force a trade-off. With the performance gain of large networks, we also renounce some understanding. The convolutional layers learn high-level features in the data; adding fully connected layers allows the network to learning non-linear combinations of these features and model more complex global patterns. While the activation of convolutional layers exhibits the features learned, the knowledge of the fully connected layers remains obscured. The fully connected layers are unwieldy and perplexing, yet they continue to be present in the most successful networks (Krizhevsky et al., 2012; Zeiler & Fergus, 2013; Simonyan & Zisserman, 2014). Our work addresses both memory and computational demand without compromise. Focusing our attention on the inner product layers, we decrease network complexity and begin to uncover the mechanism of inner product layers.

While larger network architectures achieve higher accuracy, there are a variety of methods to condense them without much harm to the network accuracy. One such technique that has gained popularity is pruning (Reed, 1993; Han et al., 2015a;b), but traditional pruning has disadvantages related to network runtime. Most existing pruning processes significantly slow down network training, and the final trained network is usually slower to execute (Han et al., 2015a). Sparse format operations require additional overhead that can greatly slow down performance unless one prunes nearly all weight entries, which can damage network accuracy.

Localized memory access patterns can be computed faster than non-localized lookups. By implementing block diagonal inner product layers in place of fully connected layers, we condense neural networks in a structured manner that speeds up the final runtime and does little harm to the final accuracy. Block diagonal inner product layers can be implemented by either initializing a purely

block diagonal weight matrix or by initializing a fully connected layer and focusing pruning efforts off the diagonal blocks to coax the dense weight matrix into structured sparsity. The first method also reduces the gradient computation time and hence the overall training time. The latter method can improve accuracy and supports the robustness of networks to *shaping*. That is, pruning can be used as a mapping between architectures–in particular, a mapping to more convenient architectures. Depending on how many iterations the pruning process takes, this method may also speed up training. We have converted a single fully connected layer into a group of smaller inner product learners whose combined efforts form a stronger learner, in essence boosting the layer. These methods also bring artificial neural networks closer to the architecture of biological mammalian brains, which have more local connectivity (Herculano-Houzel, 2012).

Another link with the mammalian brain is the relationship to random matrix theory. In neuroscience, synaptic connectivity is often represented by a matrix with entries drawn randomly from an appropriate distribution (Rajan & Abbott, 2006; Rajan, 2010). The distribution of the singular values of a large, random matrix behaves predictably according to the Marchenko-Pastur Law (Marchenko & Pastur, 1967). We show that this distribution also represents artificial neural activity matrices well. This relationship allows us to compare the behavior of inner product layers in networks that have related structure. Looking at the distribution of an inner product layer through training, we gain insight into neural networks as random dynamical systems, thereby uncovering a piece of the inner product layer "black box". Specifically, we observe that when varying the number of blocks in a layer, the initial ratio of the variance of the weights is preserved to first order throughout training.

2 Related work

Weight pruning comes in many forms including penalty and second derivative methods, sensitivity analysis and cross validation; it can be done iteratively during training or to a trained network before refining the surviving weights (Reed, 1993; Castellano et al., 1997; LeCun et al., 1990; Hassibi & Stork, 1992; Engelbrecht, 2001). With any of these methods, the result is a sparse network that takes less storage space than its fully connected counterpart. Han et al. (2015b) iteratively prune a network using the penalty method by adding a mask that disregards pruned parameters for each weight tensor. This means that the number of required floating point operations decreases, but the number performed stays the same. Furthermore, masking out updates takes additional time. Quantization and Huffman coding can be used to compress a trained network further (Xie & Jabri, 1992; Han et al., 2015a). Han et al. (2015a) report the average time spent on a forward propagation after pruning is complete and the resulting sparse layers have been converted to CSR format; for batch sizes larger than one, the sparse computations are significantly slower than the dense calculations.

Node pruning (He et al., 2014; Srinivas & Babu, 2015) or group lasso (Yuan & Lin, 2006; Lebedev & Lempitsky, 2016; Wen et al., 2016) could be used to speed up training and final execution time more easily than weight pruning since node pruning preserves some structure, but drastic node pruning can harm the network accuracy. In practice, node pruning requires additional weight fine-tuning to maintain accuracy. Sindhwani et al. (2015) propose structured parameter matrices characterized by low displacement rank that yield high compression rate as well as fast forward and gradient evaluation. However speedup is generally only seen for compression of large weight matrices, and their structured sparsity methods require parameter sharing which may be undesirable in some cases. Block sparsity yields compression with parameter flexibility. Other approaches include storing a low rank approximation for a layer's weight matrix (Sainath et al., 2013), using specific parameter sharing mechanisms (Chen et al., 2015), training smaller models on outputs of larger models (distillation) (Hinton et al., 2014), allowing neurons to read stale gradient updates (Ho et al., 2013) and using lower precision (Vanhoucke et al., 2011; M.Courbariaux et al., 2015; Gupta et al., 2015). Note that using lower precision or stale gradients are techniques that can be applied to any of the methods considered, including ours. Methods to condense and speedup convolutional layers, like predicting redundant parameters (Denil et al., 2013) and Group Lasso (Yuan & Lin, 2006; Lebedev & Lempitsky, 2016; Wen et al., 2016), can also be used in conjunction with our methods.

There is less work considering the distribution of weights in artificial neural networks. Initialization distributions to combat vanishing gradients are supported by theoretical variances for back propagation gradients under the assumption that the weights are independent, which is not valid beyond the first iteration (Glorot & Bengio, 2010; Sun, 2015). Random weights have been looked at as good

predictors of successful network architecture (Saxe et al., 2011). More recently, N.Tishby (2017) discussed the trend of the distribution of weight updates as they relate to the Mutual Information Plane. To the best of our knowledge, the effects of architecture on the change in distribution of the weights through training and the connection between trained inner product layer weights and random matrices have not been explored. In nuclear physics, the behavior of individual nuclei is the result of deterministic forces, but the behavior of a collection of interacting particles becomes chaotic. Wigner (1995) proposes a statistical description of such systems that is still widely used for studying physical systems with particle-particle interactions, like thermal conductivity. Wigner claimed that variation in the positions of compound nuclei resonances is well represented in terms of the eigenvalues of a large symmetric matrix with iid real entries. Brain activity behaves similarly at the macro level; in theoretical neuroscience, random matrices are used to model synaptic connections and to study brain plasticity (Sompolinsky et al., 1988; Rajan & Abbott, 2006; Rajan, 2010). Random matrices also appear in the study of large ecological systems (Allesina & Tang, 2015). Knowing that inner product layers in artificial neural networks are well modeled by random matrices opens the field to a new range of analytical tools that may support a specific network's robustness or plasticity.

3 METHODOLOGY

We consider two methods for implementing block diagonal inner product layers:

1. We initialize a layer with a purely block diagonal weight matrix and keep the number of connections constant throughout training.
2. We initialize a fully connected layer and iteratively prune entries off the diagonal blocks to achieve a block substructure.

Within a layer, all blocks have the same size. Method 2 is accomplished in three phases: a dense phase, an iterative pruning phase and a block diagonal phase. In the dense phase a fully connected layer is initialized and trained in the standard way. During the iterative pruning phase, focused pruning is applied to entries off the diagonal blocks using the weight decay method with L_1-norm. That is, if W is the weight matrix for a fully connected layer we wish to push toward block diagonal, we add $\alpha \sum_{i,j} |\mathbb{1}_{i,j} W_{i,j}|$ to the loss function during the iterative pruning phase, where α is a tuning parameter and $\mathbb{1}_{i,j}$ indicates whether $W_{i,j}$ is off the diagonal blocks in W. During this phase, masking out updates for pruned entries is more efficient than maintaining sparse format. When pruning is complete, to maximize speedup it is best to reformat the weight matrix once such that the blocks are condensed and adjacent in memory.[1] Batched smaller dense calculations for the blocks use cuBLAS strided batched multiplication (Nickolls et al., 2008). There is a lot of flexibility in method 2 that can be tuned for specific user needs. More pruning iterations may increase the total training time but can yield higher accuracy and reduce overfitting.

It should be mentioned that as the number of blocks increases, the overhead to perform cuBLAS strided batched multiplication can become noticeable; this library is not yet well optimized for performing many small matrix products (Masliah et al., 2016). However, with specialized batched multiplications for many small matrices, Jhurani & Mullowney (2015) attain up to 6 fold speedup.

4 EXPERIMENTS: SPEEDUP AND ACCURACY

Our goal is to reduce memory storage of the inner product layers while maintaining or reducing the final execution time of the network with minimal loss in accuracy. We will also see the reduction of total training time in some cases. To test this, we ran experiments on the MNIST (LeCun et al.) and CIFAR10 (Krizhevsky, 2009) datasets. All experiments are run on the Bridges' NVIDIA P100 GPUs through the Pittsburgh Supercomputing Center. We implement our work in Caffe (Jia et al., 2014) which provides suggested network architectures for these datasets. Training is done with batched gradient descent using the cross-entropy loss function on the softmax of the output layer. We report the forward time per inner product layer when the layers are purely block diagonal; this

[1] When using block diagonal layers, one should alter the output format of the previous layer and the expected input format of the following layer accordingly, in particular to row major ordering.

time only measures the matrix multiplication in the forward step and does not include the time to prune. We compare these times to the runtime of sparse matrix multiplication with random entries in CSR format using cuSPARSE (Nickolls et al., 2008). We also report the combined forward and backward time to do the three matrix products involved in gradient descent training when the layers are purely block diagonal. For brevity we refer to the block diagonal method as $(b_1, \ldots, b_n)$-BD; $b_i = 1$ indicates that layer i is fully connected. FC is short for all inner product layers being fully connected. We refer to networks that differ only by the number of blocks in a single inner product layer as *sister* networks.

4.1 MNIST

We experimented on the MNIST dataset with the LeNet-5 framework (LeCun et al., 1998). LeNet-5 has two convolutional layers with pooling followed by two inner product layers with ReLU activation. The first inner product layer, ip1, has a 500×800 weight matrix, and the output inner product layer, ip2, has a 10×500 weight matrix. We initialize the inner product layers using the Xavier weight filler (Glorot & Bengio, 2010), which samples a uniform distribution with variance $1/n_{in}$, where n_{in} is the number of neurons feeding into a node. We used a training batch size of 64 and all other hyperparameters provided by Caffe are left unchanged.

Figure 1 shows time and accuracy results for block diagonal method 1 without pruning. The (b_1, b_2)-BD architecture has $(800 \times 500)/b_1 + (500 \times 10)/b_2$ nonzero weights across both inner product layers. The points at which the forward sparse and forward block curves meet in Figure 1 (left) indicate the fully connected dense forward runtimes for each layer. There is speedup for $b_1 \leq 50$, or 8000 nonzero entries, when timing both forward and backward matrix products, and 1.6 times speedup when $b_1 = 100$, or 4000 nonzero entries, in the forward only case. Sparse format times are slow until there are less than 50 nonzero entries. Figure 1 (right) shows a slight decline in accuracy as the number of nonzero entries decreases. FC achieves a final accuracy of 99.11%. Without pruning, $(100, 10)$-BD has a final accuracy of 98.52%. In all cases testing accuracy remains within 1% of FC accuracy.

Using traditional iterative pruning with L_2 regularization, as suggested in (Han et al., 2015b), pruning every fifth iteration until 4000 and 500 nonzero entries survived in ip1 and ip2 respectively gave an accuracy of 98.55%, but the forward multiplication was more than 8 times slower than the dense fully connected case. Implementing $(100, 10)$-BD method 2 with pruning using 15 dense iterations and 350 pruning iterations gave a final accuracy of 98.65%. Thus we saw no great benefit to using pruning over initializing pure block diagonal inner product layers for the MNIST dataset.

In (Sindhwani et al., 2015), Toeplitz (3) has error rate 2.09% using a single hidden layer net with 1000 hidden nodes on MNIST. This method yields 63.32 fold compression over the fully connected setting. However from their Figure 3, this slows down the forward pass by around $1.5\times$ and the backward pass by around $5.5\times$. Our net with one hidden layer, 980 hidden nodes using $(49, 1)$-BD on MNIST has 29.43 fold compression and error rate 4.37% using method 2 with pruning. Our speedup is 1.53 for forward only and 1.04 when combining the forward and backward runtime. This is neat because the blocks of neurons in the hidden layer can only see a portion of the input images.

4.2 CIFAR10

We experimented on the CIFAR10 dataset with Krizhevsky's cuda-convnet (2012). Cuda-convnet has three convolutional layers with ReLu activation and pooling, followed by two fully connected layers with no activation. The first inner product layer, ip1, has a 64×1024 weight matrix, and the second has a 10×64 weight matrix. In all methods the inner product layer weights are initialized using a Gaussian filler with standard deviation 0.1, as suggested by Caffe. We used a training batch size of 100, and all other hyperparameters provided by Caffe's "quick" model are left unchanged. Caffe reports $\approx 75\%$ accuracy with the FC architecture.

Figure 2 shows time and accuracy results for the block diagonal method 1 without pruning. The (b_1, b_2)-BD architecture has $(1024 \times 64)/b_1 + (64 \times 10)/b_2$ nonzero entries across both inner product layers. In the ip1 layer, there is speedup for $b_1 \leq 32$, or 2048 nonzero entries, when timing both forward and backward matrix products, and $b_1 \leq 64$, or 1024 nonzero entries, in the forward only case. Again, sparse format performs poorly until there are less than 50 nonzero entries. Figure

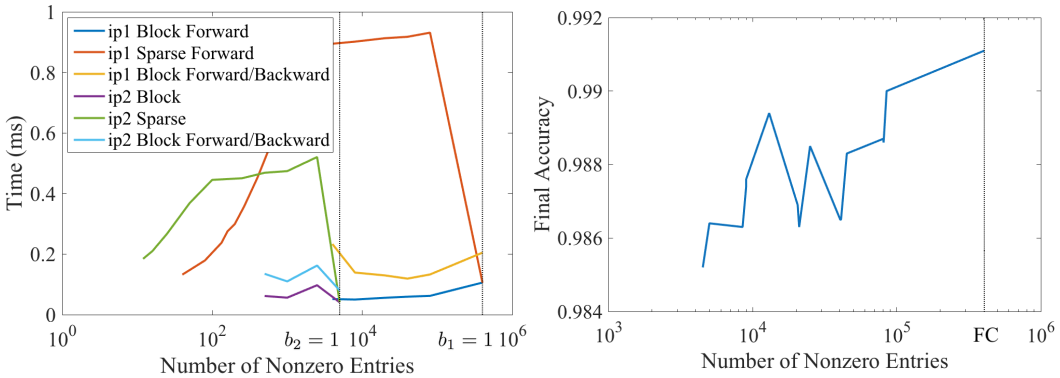

Figure 1: Time/Accuracy results using Lenet-5 on MNIST with batch size 64. (Left) For each inner product layer: forward runtimes of block diagonal and CSR sparse formats, combined forward and backward runtimes of block diagonal format. (Right) Accuracy versus total number of nonzero entries in the inner product layers after 10000 training iterations using block diagonal method 1.

2 (right) shows a decline in accuracy as the blocks increase. FC achieves a final accuracy of 76.29%. Without pruning, $(64, 2)$-BD has a final accuracy of 72.49%.

Using traditional iterative pruning with L_2 regularization pruning every fifth iteration until 1024 and 320 nonzero entries survived in the final two inner product layers gave an accuracy of 75.18%, but again the forward multiplication was more than 8 times slower than the dense fully connected computation. On the other hand, implementing $(64, 2)$-BD method 2 with pruning, which has corresponding numbers of nonzero entries, with 500 dense iterations and 1000 pruning iterations gave a final accuracy of 74.81%. This is a 35.97 fold compression of the inner product layer parameters with only a 1.5% drop in accuracy. The total forward runtime of $(64, 2)$-BD is 1.6 times faster than FC. To achieve comparable speed with sparse format we used traditional iterative pruning to leave 37 and 40 nonzero entries in the final inner product layers giving an accuracy of 73.01%. Thus implementing block diagonal layers with pruning yields comparable accuracy and memory condensation to traditional iterative pruning with faster final execution time.

Whole node pruning decreases the accuracy more than corresponding reductions in the block diagonal setting. Node pruning until ip1 had only 2 outputs, i.e. a 1024×2 weight matrix, and ip2 had a 2×10 weight matrix for a total of 2068 weights between the two layers gave a final accuracy 0f 59.67%. (64,2)-BD has a total of 1344 weights between the two inner product layers and had a final accuracy 15.14% higher with pruning.

The final accuracy on an independent test set was 76.29% on CIFAR10 using the FC net while the final accuracy on the training set itself was 83.32%. Using the (64,2)-BD net without pruning, the accuracy on an independent test set was 72.49%, but on the training set was 75.63%. With pruning, the accuracy of (64,2)-BD on an independent test set was 74.81%, but on the training set was 76.85%. Both block diagonal methods decrease overfitting; the block diagonal method with pruning decreases overfitting slightly more.

5 Random Matrix Theory Observations

The Marchenko-Pastur distribution describes the asymptotic behavior of the singular values of large random matrices with iid entries (1967). Let X be an $m \times n$ matrix with iid entries x_{ij} such that $\mathrm{E}[x_{ij}] = 0$ and $\mathrm{Var}[x_{ij}] = \sigma^2$. The Marchenko-Pastur theorem states that as $n, m \to \infty$ such that $m/n \to y > 0$, with probability 1 the empirical spectral distribution of $\frac{1}{n}XX^\top$ converges in distribution to the density

$$\mu_y(x) = \begin{cases} \frac{1}{2\pi\sigma^2 yx}\sqrt{(b-x)(x-a)} & \text{if } a \leq x \leq b \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

with point mass $1 - 1/y$ at the origin if $y > 1$ where $a = \sigma^2(1 - y^2)$ and $b = \sigma^2(1 + y^2)$.

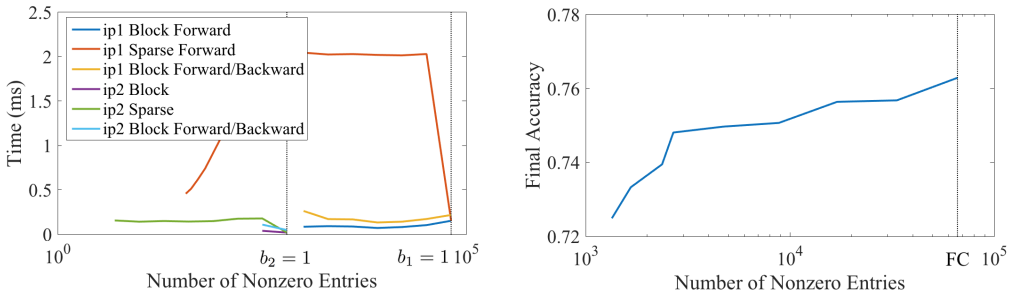

Figure 2: Time/Accuracy results of cuda-convnet on CIFAR10 with batch size 100. (Left) For each inner product layer: forward runtimes of block diagonal and CSR sparse formats, combined forward and backward runtimes of block diagonal format. (Right) Accuracy versus total number of nonzero entries in the inner product layers after 9000 training iterations using block diagonal method 1.

Network weights do not remain independent through training. Without momentum, W receives the update $W \leftarrow W - \frac{\lambda}{b}\sum_{i=1}^{b}\frac{\partial L(x_i)}{\partial W}$ in an iteration, where λ is the learning rate, b is the batch size, L is the loss function and x_i is sampled from the data distribution. One can easily verify,

$$\Delta\mathrm{Var}(W) = \lambda^2\mathrm{Var}\left(\frac{1}{b}\sum_{i=1}^{b}\frac{\partial L(x_i)}{\partial W}\right) - 2\lambda\mathrm{Cov}\left(W, \frac{1}{b}\sum_{i=1}^{b}\frac{\partial L(x_i)}{\partial W}\right) \tag{2}$$

using estimators for the right side of the equation, where a simple Hadamard (Schur) product provides a good estimate for the second term. In our experience, the covariance quickly became the dominating term. However, for a large enough weight matrix, the singular values of an inner product layer weight matrix behave according to the Marchenko-Pastur distribution even after thousands of training iterations. That is, after thousands of correlated updates, the weight matrix behaves like a matrix with iid entries. The assumption of independence fails at the micro level when calculating the change in variance of the weights, but is accurate at the macro level when considering the behavior of the weight matrix as an operator.

In this section we discuss only the first method for implementing block diagonal layers without pruning. While a relationship between fully connected layer weights and the corresponding block diagonal layer weights in trained sister networks is not evident from equation (2), indeed a relationship can be seen in the singular values of the weight matrices, and, in particular, in the change in the variance of the weight matrix entries throughout training. The initialization ratio of variances in corresponding layer weight matrices of sister networks persists through thousands of training iterations. This finding may provide a good mechanism for examining related network architectures and may support claims about a network's malleability or fitness.

5.1 MNIST

As mentioned, in our experiments on the MNIST dataset the inner product layer weights are initialized using the Xavier algorithm (Glorot & Bengio, 2010). Thus the initialization variance of the weights in ip1 is $b_1/800$ in the block diagonal case, where b_1 is the number of blocks, and the ratio of this variance over that of the fully connected case at initialization is b_1. Figure 3 (left) shows that this ratio is a good first order estimate for the final ratio after 10000 iterations. The final ratios are $5.03, 9.97, 19.96, 49.32$ and 96.99 for $b_1 = 5, 10, 20, 50$, and 100 respectively. We can see that the relationship deteriorates as the number of blocks increases. This holds for any random input order and when the sigmoid activation function is used for layer ip1 as long as the activation function is consistent across sister networks. When using the sigmoid activation function, the ratio seemed to deteriorate less quickly; e.g. the final variance ratio was 101.00 for $b_1 = 100$.

Figure 3 (right) compares the singular values of the ip1 layer weight matrix for varying numbers of blocks after 10000 training iterations to the singular values of a truly random 800×500 matrix initialized with variance 6.6×10^{-4}, the final variance of the FC ip1 layer weights. The singular values of the ip1 weight matrix follow the curve that the singular values of the truly random matrix

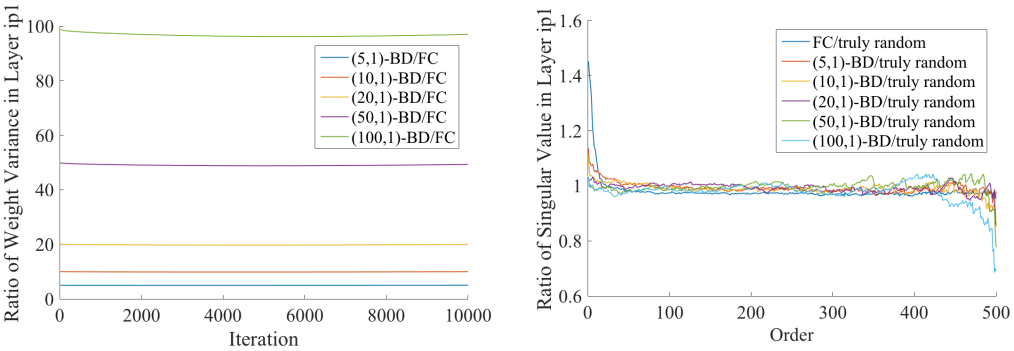

Figure 3: 10000 training iterations using Lenet-5 net on MNIST. (Left) Ratio of trained ip1 weight matrix singular values over singular values of a truly random matrix with the same dimensions. (Right) Ratio of variance of ip1 weight matrix entries in block diagonal setting over variance of ip1 weight matrix entries in the fully connected setting.

create with some error in the largest and smallest singular values. We believe the disparity in the extreme singular values is the result of over training (See Appendix B.1); using FC, the accuracy reaches 98.28% by iteration 1000. In the block diagonal method, we aggregate the singular values of each block and sort them to create the corresponding curves in Figure 3 (right). The individual block spectral distributions are identical to each other. The decrease in matrix size by a factor of b_1 cancels with the increase in variance by a factor of b_1 in equation (1), resulting in singular values that all follow the same curve. Thus, the ratio of singular values of the trained ip1 layer weight matrix over the singular values of the truly random matrix is ≈ 1. Additional supporting figures comparing the PDFs of the singular values and the Kullback-Leibler divergence can be found in Appendix B.1.

If we make ip2 a BD layer as well, the change in variance in ip1 sees minimal effect. For $b_1 = 1, 2, 5, 10, 50, 100$ and $b_2 = 1, 2, 5, 10$, the final variance in the ip1 layer weights using (b_1, b_2)-BD over the final variance in the ip1 layer weights using $(b_1, 1)$-BD is ≈ 1 with error ≤ 0.05. [2]

5.2 CIFAR10

In our experiments on CIFAR10, the first inner product layer weights are initialized using a Gaussian filler with standard deviation 0.1. Thus the ratio of variance of the weights in ip1 in the block diagonal case over that of the fully connected case at initialization is 1. Figure 4 (left) indicates that this ratio is a good first order estimate for the final ratio after 9000 iterations at which time the ratios are $1.02, 1.02, 1.01, 1.11, 1.21$ and 1.5 for $b_1 = 2, 4, 8, 16, 32$, and 64 respectively. Again, the relationship deteriorates as the number of blocks grows, and this holds for any random input order.

We compared the singular values of the weight matrix in layer ip1 for varying numbers of blocks after 9000 training iterations to the singular values of a truly random 1024×64 matrix initialized with variance 7×10^{-3}, the final variance of the FC ip1 layer weights after training. The singular values of the fully connected ip1 layer weight matrix follow the curve that the singular values of the truly random matrix create. As in the MNIST experiments, we aggregate the singular values of each block in the block diagonal method and sort them. In this case, variance of the ip1 layer weights remains relatively constant for varying numbers of blocks, so, by equation (1), the decrease in matrix size by a factor of b_1 must appear in the ratio of singular values as $1/\sqrt{b_1}$. Figure 4 (right) shows this relationship. The ratio of singular values of the trained weight matrix in layer ip1 over the singular values of the random matrix is $\approx 1/\sqrt{b_1}$ for b_1 blocks with deterioration as b_1 grows. Additional supporting figures comparing the PDFs of the singular values and the Kullback-Leibler divergence can be found in Appendix B.2.

If in addition we make the ip2 a BD layer, again, the change in variance in ip1 sees minimal effect. For $b_1 = 1, 2, 4, 8, 16, 32, 64$, the final variance in the ip1 layer weights using the $(b_1, 2)$-BD method over the final variance in the ip1 layer weights using the $(b_1, 1)$-BD method is ≈ 1 with error ≤ 0.03.

[2] The ip2 layer weights also *appear* to have the same behavior, but the matrix size is much smaller so the estimate of the variance is lower order and the asymptotic assumptions of Marchenko-Pastur are far from met.

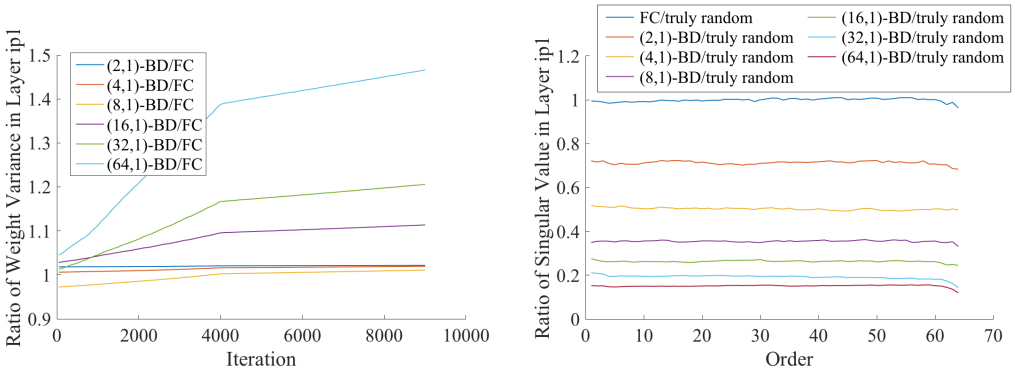

Figure 4: 9000 training iterations using cuda-convnet on CIFAR10. (Left) Ratio of trained ip1 weight matrix singular values over singular values of a truly random matrix with the same dimensions. (Right) Ratio of variance of ip1 weight matrix entries in block diagonal setting over variance of ip1 weight matrix entries in the fully connected setting.

6 Conclusion

We have shown that block diagonal inner product layers can reduce network size, training time and final execution time without significant harm to the network performance. We have also shown that random matrix theory gives informative results about relationships in network structure that are preserved through training.

While traditional iterative pruning can reduce storage, the random indices of the surviving weights make sparse computation inefficient, slowing down the training and final execution time of the network. Our block diagonal methods address this inefficiency by confining dense regions to blocks along the diagonal. Without pruning, block diagonal method 1 allows for faster training time. Method 2 preserves the learning with focused, structured pruning that reduces computation for speedup during execution. In our experiments, method 2 saw higher accuracy than the purely block diagonal method for the more complex learning problem, CIFAR10; however, the increase in accuracy came in exchange for slightly more time to train the network. There is great deal of flexibility in our block diagonal methods that can be tuned to an individual problem. These methods may also make larger network architectures more feasible to train and use since they convert a fully connected layer into a collection of smaller inner product learners working jointly to form a stronger learner.

There is a lot of room for additional speedup with block diagonal layers. Dependency between layers poses a noteworthy bottleneck in network parallelization. With structured sparsity like ours, one no longer needs a full barrier between layers. Additional speedup would be seen in software optimized to support weight matrices with organized sparse form, such as blocks, rather than being optimized for dense matrices. For example, for many small blocks, one can reach up to 6 fold speedup with specialized batched matrix multiplication (Jhurani & Mullowney, 2015). Hardware has been developing to better support for sparse operations, and a block format may be especially suitable for training on evolving architectures such as neuromorphic systems. These systems, which are far more efficient than GPUs at simulating mammalian brains, have a pronounced 2-D structure and are ill-suited to large dense matrix calculations (Merolla et al., 2014; Boahen, 2014).

Comparing block diagonal and fully connected layers in sister networks highlighted that the training process for neural networks can be viewed as a random dynamical system. The relationship in the change in variance through training between the block diagonal and fully connected case suggests an invariant pattern that may help evaluate network fitness. We emphasize that this is a nontrivial relationship surviving various datasets, network architectures, and activation functions. The group behavior is that of a random matrix with independent entries, but the individual weight updates have complex dependencies. Similar random activity occurs in the mammalian brain and suggests looking at random matrix theory to support a network's plasticity or robustness. This could also be a catalyst for looking at optimal distributions for a trained network. Random matrix theory has been indispensable to the advancement of nuclear physics, quantum physics, neuroscience, and ecology, and has the potential to elevate artificial neural network analysis in the same manner.

ACKNOWLEDGMENTS

All experiments are run on the Bridges' NVIDIA P100 GPUs through the Pittsburgh Supercomputing Center.

REFERENCES

S. Allesina and S. Tang. The stability-complexity relationship at age 40: a random matrix perspective. *Population Ecology*, 57(1):63–75, 2015.

K. Boahen. Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations. *Proceedings of the IEEE*, 102(5):699716, 2014.

G. Castellano, A. M. Fanelli, and M. Pelillo. An iterative pruning algorithm for feedforward neural networks. *IEEE Transactions on Neural Networks*, 8(3):519–531, 1997.

W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen. Compressing neural networks with the hashing trick. *International Conference on Machine Learning*, 2015.

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

A. P. Engelbrecht. A new pruning heuristic based on variance analysis of sensitivity information. *IEEE Transactions on Neural Networks*, 12(6):1386–1399, 2001.

X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. *International Conference on Artificial Intelligence and Statistics*, 9:249–256, 2010.

S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan. Deep learning with limited numerical precision. *International Conference on Machine Learning*, pp. 1737–1746, 2015.

S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *International Conference on Learning Representations*, 2015a.

S. Han, J. Pool, J. Tran, and W. J. Dally. Learning both weights and connections for efficient neural networks. *Neural Information Processing Systems*, pp. 1135–1143, 2015b.

B. Hassibi and D. G. Stork. Second order derivatives for network pruning: Optimal brain surgeon. *Advances in neural information processing systems*, pp. 164–171, 1992.

T. He, Y. Fan, Y. Qian, T. Tan, and K. Yu. Reshaping deep neural network for fast decoding by node-pruning. *IEEE International Conference on Acoustic, Speech and Signal Processing*, pp. 245–249, 2014.

S. Herculano-Houzel. The remarkable, yet not extraordinary, human brain as a scaled-up primate brain and its associated cost. *National Academy of Sciences*, 2012.

G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *NIPS workshop*, 2014.

Q. Ho, J. Cipar, H. Cui, J. Kyu Kim, S. Lee, P. B. Gibbons, G. A. Gibson, G. R. Ganger, and E. P. Xing. More effective distributed ml via a stale synchronous parallel parameter server. *Advances in Neural Information Processing Systems*, pp. 1223–1231, 2013.

K. Hornik, M. Stinchcombe, H. White D. Achlioptas, and F. McSherry. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

C. Jhurani and P. Mullowney. A gemm interface and implementation on nvidia gpus for multiple small matrices. *Journal of Parallel and Distributed Computing*, 75:133–140, 2015.

Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv:1408.5093*, 2014.

A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, Computer Science, University of Toronto, 2009. URL `https://www.cs.toronto.edu/~kriz/cifar.html`.

A. Krizhevsky. cuda-convnet. Technical report, Computer Science, University of Toronto, 2012. URL `https://code.google.com/p/cuda-convnet/`.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, pp. 1106–1114, 2012.

S. Kullback and R.A. Leibler. On information and sufficiency. *Annals of Mathematical Statistics*, 22(1):79–86, 2015.

V. Lebedev and V. Lempitsky. Fast convnets using group-wise brain damage. *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

Y. LeCun, C. Cortes, and C. J.C. Burges. The mnist database of handwritten digits. Technical report. URL `http://yann.lecun.com/exdb/mnist/`.

Y. LeCun, J. S. Denker, and S. A. Solla. Optimal brain damage. *Advances in Neural Information Processing Systems*, pp. 598–605, 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.

V. A. Marchenko and L. Pastur. Distribution of eigenvalues for some sets of random matrices. *Mathematics of the USSR-Sbornik*, 1(4):457–483, 1967.

I. Masliah, A. Abdelfattah, A. Haidar, S. Tomov, M. Baboulin, J. Falcou, and J. Dongarra. High-performance matrix-matrix multiplications of very small matrices. *Euro-Par 2016: Parallel Processing*, 9833:659–671, 2016.

M.Courbariaux, J.-P.David, and Y.Bengio. Low-precision storage for deep learning. *ICLR*, 2015.

P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, B. Brezzo, I. Vo, S. K. Esser, R. Appuswamy, B. Taba, A. Amir, M. D. Flickner, W. P. Risk, R. Manohar, and D. S. Modha. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673, 2014.

J. Nickolls, I. Buck, M. Garland, and K. Skadron. Scalable parallel programming with cuda. *ACM Queue*, 6(2):40–53, 2008.

N.Tishby. Information theory of deep learning. June 2017.

K. Rajan. What do random matrices tell us about the brain? *Grace Hopper Celebration of Women in Computing*, 2010.

K. Rajan and L. F. Abbott. Eigenvalue spectra of random matrices for neural networks. *Physical Review Letters*, 97(18), 2006.

R. Reed. Pruning algorithms-a survey. *IEEE Transactions on Neural Networks*, 4(5):740–747, 1993.

M. Rudelson and R. Vershynin. Non-asymptotic theory of random matrices: extreme singular values. *Proceedings of the International Congress of Mathematicians*, 3:1576–1602, 2010.

T. N. Sainath, B. Kingsbury, V. Sindhwani, E. Arisoy, and B. Ramabhadran. Low-rank matrix factorization for deep neural network training with high-dimensional output targets. *IEEE International Conference on Acoustic, Speech and Signal Processing*, 2013.

A. M. Saxe, P. Wei Koh, Z. Chen, M. Bhand, B. Suresh, and A. Y. Ng. On random weights and unsupervised feature learning. *International Conference on Machine Learning*, 2011.

P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *ICLR*, 2014.

K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv:1409.1556*, 2014.

Vikas Sindhwani, Tara Sainath, and Sanjiv Kumar. Structured transforms for small-footprint deep learning. pp. 3088–3096, 2015. URL http://papers.nips.cc/paper/5869-structured-transforms-for-small-footprint-deep-learning.pdf.

H. Sompolinsky, A. Crisanti, and H. Sommers. Chaos in random neural networks. *Physical Review Letters*, 61(3):259–262, 1988.

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

Kaiming He Xiangyu Zhang Shaoqing Ren Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *arXiv:1502.01852*, 2015.

V. Vanhoucke, A. Senior, and M. Z. Mao. Improving the speed of neural networks on cpus. *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 29*, pp. 2074–2082. Curran Associates, Inc., 2016. URL http://papers.nips.cc/paper/6504-learning-structured-sparsity-in-deep-neural-networks.pdf.

E. Wigner. Characteristic vectors of bordered matrices with infinite dimensions. *Annals of Mathematics*, 62(3):548–564, 1995.

Y. Xie and M.A. Jabri. Analysis of the effects of quantization in multilayer neural networks using a statistical model. *IEEE Transactions on Neural Networks*, 3(2):334–338, 1992.

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

M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. *arXiv:1311.2901*, 2013.

APPENDIX

A KULLBACK-LEIBLER DIVERGENCE

The Kullback-Leibler divergence (also called relative entropy) is a measure of how one probability distribution diverges from a second probability distribution (Kullback & Leibler, 2015). The Kullback-Leibler divergence from Q to P, denoted $D_{KL}(P||Q)$, is the information gain if P is used instead of Q, or the amount of information lost when Q is used to approximate P.

$$D_{KL}(P||Q) = \sum_{i \in I} P(i) \log \left(\frac{P(i)}{Q(i)} \right)$$

With a Kullback-Leibler divergence of 0, one can expect similar, if not identical, behavior of the two different distributions.

B RANDOM MATRIX THEORY ADDITIONAL SUPPORTING MATERIAL

B.1 MNIST

The ratio of the variance and singular values of the ip1 layer weight matrix in block diagonal setting over that of the ip1 layer weight matrix in the fully connected setting after 10000 training iterations on the MNIST dataset using Lenet-5 framework can be seen in Figure 3. These ratios highlight the

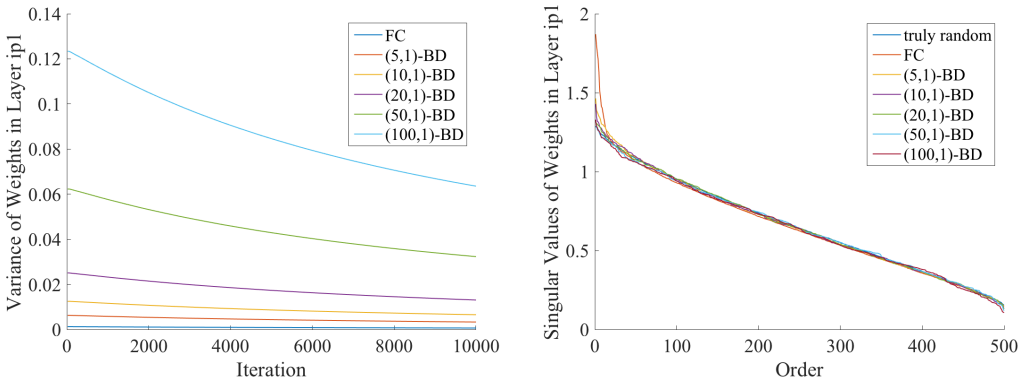

Figure 5: 10000 iterations of training Lenet-5 framework with ReLu activation and Caffe prescribed hyperparameters on MNIST. (Left) Variance of weight matrix entries in layer ip1 in both the fully connected and block diagonal setting. (Right) Singular values of ip1 weight matrix in block diagonal and fully connected setting.

relationship to random matrix theory best, but here we include the variance and the singular values without taking a ratio because they may also be informative for some readers. See Figure 5.

Figure 6 (left) compares the theoretical probability density function of the singular values of a truly random matrix with independent entries to the measured distribution of the ip1 layer weight matrix singular values for the FC architecture after 10000 training iterations. Figure 6 (right) uses the Kullback-Leibler divergence (Appendix A) to measure the difference between the theoretical probability density function of the singular values of a truly random matrix to the measured distribution of the ip1 layer weight matrix singular values for the $(b_1, 1)$-BD architecture after 10000 training iterations. For $b_1 = 1, 5, 10, 20, 50,$ and 100 the Kullback-Leibler divergence stays between 0.05 and 0.081 with the highest values when $b_1 = 1$, i.e. the FC case. One can see from Figures 3 (right) and 5 (right) that the disparity in the larger singular values decreases as b_1 increases.

For comparison, the average Kullback-Leibler divergence from the discrete measured empirical spectral distribution of $\frac{1}{500}RR^\top$ where R is an 800×500 matrix with standard normally distributed entries to the Marchenko-Pastur distribution $\mu_{0.625}$ as in equation (1) is $D_{KL}(\mu_{0.625}||\lambda_{\frac{1}{500}RR^\top}) = 0.014$ where the average is taken over over 20 initializations. This divergence is the same magnitude as the measured divergence in Figure 6 (right).

It should be mentioned that $D_{KL}(\lambda_{\frac{1}{500}W_{b_1}W_{b_1}^\top}||\mu_{0.625})$ is infinite because of the difference in support, but this disparity in support reduces as b_1 increases. This disparity may be incurred by the relatively small matrix dimensions, as asymptotic random matrix theory often does not represent the extreme singular values well in the finite case (Rudelson & Vershynin, 2010), but is more likely caused by overtraining. Using FC, the accuracy reaches 98.28% by iteration 1000. After 1000 iterations, the ratio of the largest singular value of the trained weight matrix in layer ip1 over the largest singular value of a truly random matrix with the same dimensions and variance is 1.16. Figure 3 (right) shows that this ratio is 1.44 after the full 10000 iterations. $D_{KL}(\mu_{0.625}||\lambda_{\frac{1}{500}W_{b_1}W_{b_1}^\top}) = 0.05$ after only 1000 iterations.

The relationship between inner product layer weight matrices and random matrix theory was first seen in inner product layer nets without convolutional layers. With interest leaning toward convolutional nets we did not include this in the main work, but we include this here for completeness. The relationship is stronger in deeper layers and for larger weight matrices.

In a 3 layer network in which both hidden layers have 500 nodes and ReLu activation, we compare FC to (1,100,1)- BD on MNIST where in both cases ip2 is initialized with Xavier variance (Glorot & Bengio, 2010) and all other hyperparameters as perscribed by Caffe (Jia et al., 2014). After 10000 iterations, the ratio of variance of weight matrix entries in layer ip2 in block diagonal setting over variance of weight matrix entries in layer ip2 in the fully connected setting is 106.42.

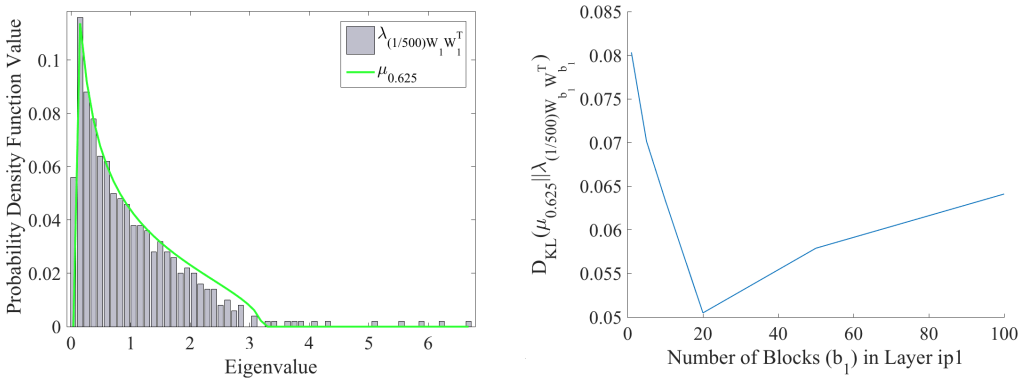

Figure 6: Let $\lambda_{\frac{1}{500}W_{b_1}W_{b_1}^\top}$ be the measured empirical spectral distribution of $\frac{1}{500}W_{b_1}W_{b_1}^\top$ where W_{b_1} is the ip1 layer weight matrix in the $(b_1, 1)$-BD architecture after 10000 training iterations on MNIST using Lenet-5 that has been re-centered to have mean zero and variance $1/b_1$. (Left) Bar graph of $\lambda_{\frac{1}{500}W_1W_1^\top}$ with plot of the Marchenko-Pastur distribution $\mu_{0.625}$ as in equation (1). (Right) The Kullback-Leibler divergence from $\lambda_{\frac{1}{500}W_{b_1}W_{b_1}^\top}$ to the Marchenko-Pastur distribution $\mu_{0.625}$, $D_{KL}(\mu_{0.625}||\lambda_{\frac{1}{500}W_{b_1}W_{b_1}^\top})$.

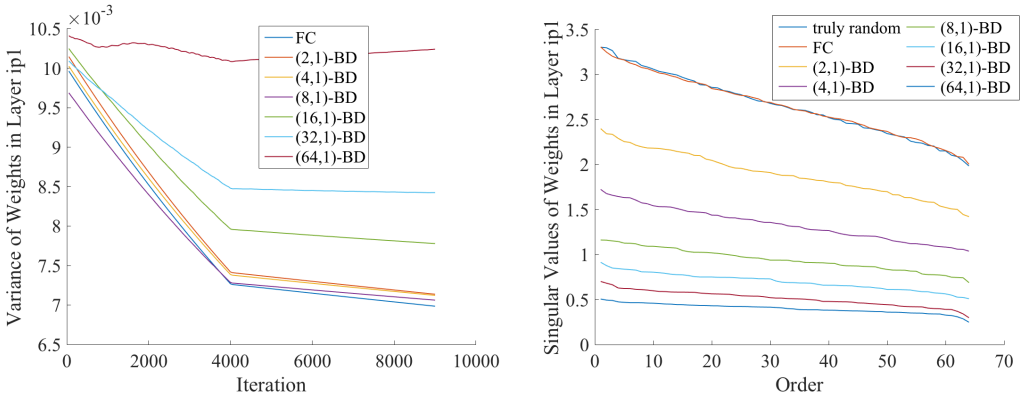

Figure 7: 9000 training iterations using cuda-convnet on CIFAR10. (Left) Variance of weight matrix entries in layer ip1 in both the fully connected and block diagonal setting. (Right) Singular values of ip1 weight matrix in block diagonal and fully connected setting.

Let $\lambda_{\frac{1}{500}W_1W_1^\top}$ be the measured empirical spectral distribution of $\frac{1}{500}W_1W_1^\top$ where W_1 is the ip2 layer weight matrix in the FC architecture with only 3 inner product layers after 10000 iterations. The final variance of the ip2 layer weights in FC is 0.0012. If R is a 500×500 truly random matrix with independent entries that have variance 0.0012, then $D_{KL}(\lambda_{\frac{1}{500}RR^\top}||\lambda_{\frac{1}{500}W_1W_1^\top}) = 0.0701$.

B.2 CIFAR10

The ratio of the variance and singular values of the ip1 layer weight matrix in block diagonal setting over that of the ip1 layer weight matrix in the fully connected setting after 9000 training iterations on CIFAR10 using cuda-convnet can be seen in Figure 4. These ratios highlight the relationship to random matrix theory best, but here we include the variance and the singular values without taking a ratio because they may also be informative for some readers. See Figure 7.

Figure 8 (left) compares the theoretical probability density function of the singular values of a truly random matrix with independent entries to the measured distribution of singular values the ip1 layer weight matrix for the FC architecture after 9000 training iterations. Figure 8 (right) uses the Kullback-Leibler divergence (Appendix A) to measure the difference between the theoretical

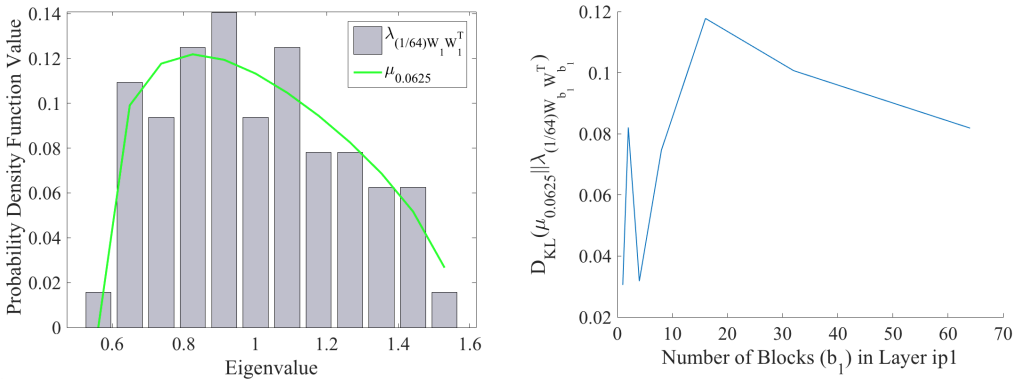

Figure 8: Let $\lambda_{\frac{1}{64}W_{b_1}W_{b_1}^\top}$ be the measured empirical spectral distribution of $\frac{1}{64}W_{b_1}W_{b_1}^\top$ where W_{b_1} is the ip1 layer weight matrix in the $(b_1, 1)$-BD architecture after 9000 training iterations on the CIFAR10 dataset using cuda-convnet framework that has been re-centered to have mean zero and variance $1/b_1$. (Left) Bar graph of $\lambda_{\frac{1}{64}W_1W_1^\top}$ with plot of the Marchenko-Pastur distribution $\mu_{0.0625}$ as in equation (1). (Right) The Kullback-Leibler divergence from $\lambda_{\frac{1}{64}W_{b_1}W_{b_1}^\top}$ to the Marchenko-Pastur distribution $\mu_{0.0625}$, $D_{KL}(\mu_{0.0625}||\lambda_{\frac{1}{64}W_{b_1}W_{b_1}^\top})$.

probability density function of the singular values of a truly random matrix to the measured distribution of singular values the ip1 layer weight matrix for the $(b_1, 1)$-BD architecture after 9000 training iterations. For $b_1 = 1, 2, 4, 8, 16, 32$, and 64 the Kullback-Leibler divergence stays between 0.03 and 0.12.

For comparison, the average Kullback-Leibler divergence from the discrete measured empirical spectral distribution of $\frac{1}{64}RR^\top$ where R is an 1024×64 matrix with standard normally distributed entries to the Marchenko-Pastur distribution $\mu_{0.0625}$ as in equation (1) is $D_{KL}(\mu_{0.0625}||\lambda_{\frac{1}{64}RR^\top}) = 0.0218$ where the average is taken over over 20 initializations. At its largest discrepancy, this is 5.4 times less than the measured divergence in Figure 8 (right).