

ClosNets: a Priori Sparse Topologies for Faster DNN Training

Mihailo Isakov and Michel A. Kinsy
 Adaptive and Secure Computing Systems Laboratory
 Department of Electrical and Computer Engineering
 Boston University, Boston, USA

Abstract—Fully-connected layers in deep neural networks (DNN) are often the throughput and power bottleneck during training. This is due to their large size and low data reuse. Pruning dense layers can significantly reduce the size of these networks, but this approach can only be applied after training. In this work we propose a novel fully-connected layer that reduces the memory requirements of DNNs without sacrificing accuracy. We replace a dense matrix with products of sparse matrices whose topologies we pick in advance. This allows us to: (1) train significantly smaller networks without a loss in accuracy, and (2) store the network weights without having to store connection indices. We therefore achieve significant training speedups due to the smaller network size, and a reduced amount of computation per epoch. We tested several sparse layer topologies and found that Clos networks perform well due to their high path diversity, shallowness, and high model accuracy. With the ClosNets, we are able to reduce dense layer sizes by as much as an order of magnitude without hurting model accuracy.

I. INTRODUCTION

Training deep neural networks (DNN) is both time-consuming and power intensive. While the bulk of the computation in applications like image processing revolves around convolutional layers, this is not the case for many other applications, e.g., text-to-speech processing, machine translation and financial forecasting. Fully-connected layers, i.e., dense layers, are more common in those applications since their data exhibit more temporal correlations. Consequently, their computation requires a much larger memory footprint, more data movement, and longer processing time with significantly more power [1]. Several works have attempted to decrease the size of dense layers by pruning [1] or quantization [2]. However, most of these methods only speed up inference and not training. In this work we tackle the challenge of speeding up deep neural network training by decreasing the size of these dense layers. To achieve this we break up the dense matrices into products of sparse matrices. These products retain full connectivity while requiring less parameters. Our work is orthogonal to quantization and can further benefit from approaches such as ZipML [3].

To speed up the training of dense neural networks, we investigate the bottlenecks of accelerating dense layers. In order to determine whether networks are computationally or memory bound, we measure the time required for training on one epoch with varying batch sizes. While the number of operations required to train on an epoch is independent of the batch size, the number of times we have to load all weight matrices is equal to the number of batches. It can be noted in Figure 1 that the training time grows inversely with the batch size. We attribute this effect to the system being computationally bound for large batch sizes and memory bound for small batch sizes with increased number of batches. To reduce the impact of this memory

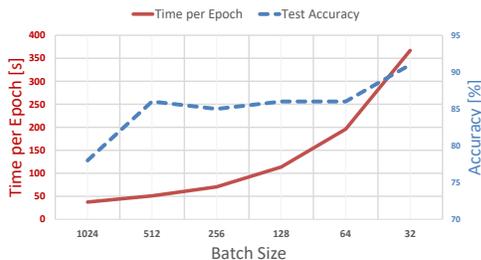


Fig. 1: Time required to train a ResNet18 on a single epoch of the CIFAR-10 dataset, with varying batch sizes.

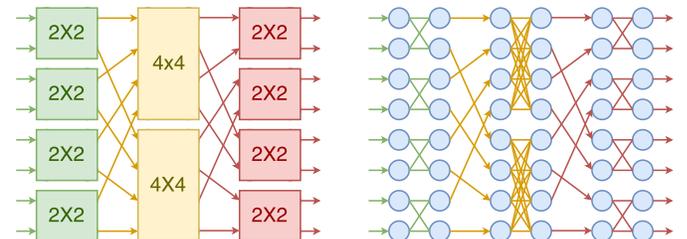
wall, one can either increase the bandwidth or decrease the amount of memory required for training. Here we focus on decreasing the DNN memory requirements. We explore both algorithmic modifications to the neural network structure and hardware customization techniques to reducing the size of the dense layers.

In this work we introduce the concept of predefined topology for sparse neural networks to enable faster inference and training, along with lower memory requirements and power usage. This predefined topology needs to have the following properties: (1) full connectivity, (2) shallowness, (3) pre-determined connectivity, (4) uniform and high path diversity, and (5) an efficient hardware implementation.

II. CLOSNETS

In the search for a structure that meets the above mentioned properties of a predefined topology, we examine several topologies, e.g., torus, hypercube, butterfly and mesh. Most of them do not satisfy the (2) shallowness requirement, as they require many cascading layers before (1) full connectivity is achieved. One topology that grants all of these properties is *Clos network*. A Clos network is a three-stage network in which each stage is composed of a number of crossbar switches [4]. While in the networking domain a Clos network is assumed to have the same number of input and output nodes, we define a more general Clos network as a 5-tuple (I, O, R_i, R_m, R_o) . In this characterization, I is the number of inputs, O is the number of outputs, R_i is the number of input routers, R_m is the number of middle routers, and R_o is the number of output routers. In Figure 2a we can see an example (16, 16, 4, 2, 4) Clos network.

The intuition behind Clos networks is that since each middle router acts as a crossbar between the input and output routers, there are as many paths between an input-output pair (i, o) as there are middle routers. This gives the network designer a simple way of preventing network contention by increasing the path diversity. To map Clos networks to the DNN domain, we replace all the input and output nodes with neurons, and each router becomes a fully-connected network of the same size with its own hidden neurons. It is important to note that the connections between the routers are a simple scatter operation. These connections do not have weights (i.e. they do not amplify or inhibit their signals), but purely permute the positions of the activations. In Figure 2b the network from Figure 2a is mapped to the neural network domain.



(a) A 16-input, 16-output Clos network with 4 input routers, 2 middle routers, and 4 output routers.

(b) The same network from Figure 2a mapped to the neural network domain.

Fig. 2: Clos in the networking and DNN domain.

The Clos neural networks have clear benefits over the other explored topologies. They are fully-connected and shallow. They have a parametrizable but uniform path diversity. Furthermore, they offer a simple hardware implementation. An added benefit of Clos networks is that we are not restricted to having either (1) the identical number of inputs and outputs, or (2) number of inputs/outputs being a power of two. For a given Clos network (I, O, R_i, R_m, R_o) , we can calculate the number of parameters P in the network, and path diversity D as:

$$P = R_m(I + O + R_i R_o), \quad D = R_m \quad (1)$$

We compare the model accuracies of different configurations of the baseline (dense) models, low-rank models, a priori pruned models and Clos models, with respect to the number of parameters of each model. Figure 3 shows the test accuracy of different networks trained on MNIST for 50 epochs with respect to their parameter count. As we reduce the parameter count, the networks degrade in accuracy. The performance degradation is more graceful for some networks compared to others. Notice that the Clos networks have comparable accuracy with the baseline networks while having $5.5\times$ less parameters.

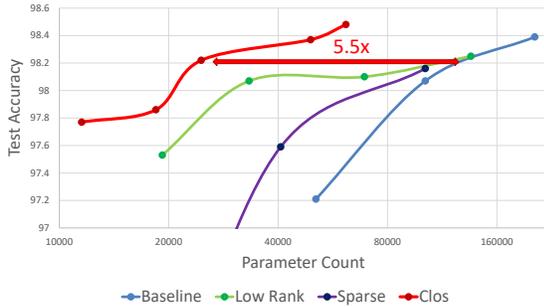


Fig. 3: Accuracy vs. parameters tradeoff for different network types.

III. ARCHITECTURE

We propose a simple hardware implementation for our Clos networks. For brevity, we will restrict ourselves to networks with the same number of inputs and outputs, and optimize for throughput, not area or latency. By observing Figure 2b, we can see that the network consists of several independent fully-connected layers, connected by a scatter operation. We can implement the fully-connected layers with a number of processing elements connected in a ring topology. Each processing element is calculating the value of one of the output neurons of the dense layer, with the input neuron values circling the ring. Once each ring has calculated the outputs of each neuron in a single column, the torus reconfigures to a set of perpendicular rings, one for each column in this case. Each torus node now calculates a new neuron output from the next column from Figure 2b, while the previous results are treated as inputs and circle the rings. We illustrate this on a small 4×4 Clos network as seen in Figure 4. We map this network to the 2×2 torus from Figure 4 (right). This leads us to a torus implementation as seen in Figure 5. The three router layers (input, middle, output) then map to three ring-AllReduce operations, a horizontal one over each row individually (green), a vertical one over each column (yellow), and again a horizontal one over each row (red). Notice that the positions of the neurons R_i , R_m and R_o change when mapped to torus nodes (positions 2 and 3 swap places). This is done to minimize data movement and allow single hop movement of neuron outputs. From figure 5, we see that neuron outputs are transmitted only in the east and south directions. This is true only for inference. During training, gradients backtrack their path through

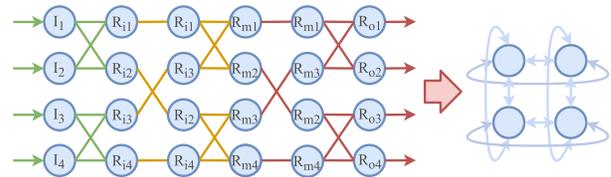


Fig. 4: (Left) A $(4, 4, 2, 2)$ Clos network with inputs I , intermediate results R_i and R_m , and outputs R_o . (Right) A 2×2 torus we want to map the Clos network to.

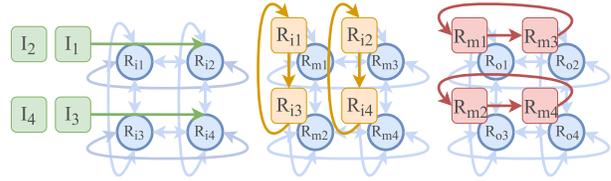


Fig. 5: The three rings router outputs from the Clos network calculated with a ring-AllReduce operation.

the network, flowing north and west. Therefore, we implement two router designs, one for inference – Figure 6 and the other for training – Figure 7 to support the communication patterns described above.

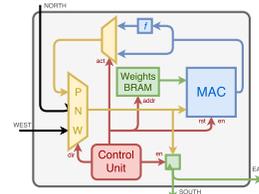


Fig. 6: Inference router, accepting inputs from north and west, and forward them east and south. Once the router computes the neuron activation, it sends it south or east.

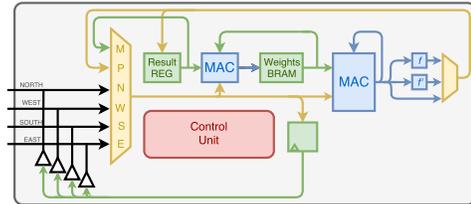


Fig. 7: A training router, with bidirectional links to all four directions. The router processes both forward and backward passes.

IV. CONCLUSION

In this work we introduce a novel approach for reducing the size of dense neural network layers. We present ClosNets - fully-connected cascades of sparse layers with the Clos topology. We show that ClosNets have comparable accuracy and $5.5\times$ smaller size over conventional fully-connected layers, and propose a simple torus-based implementation of the network.

REFERENCES

- [1] S. Han, H. Mao, and W. J. Dally, “Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding,” *CoRR*, vol. abs/1510.00149, 2015.
- [2] M. Courbariaux and Y. Bengio, “Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1,” *CoRR*, vol. abs/1602.02830, 2016.
- [3] H. Zhang, K. Kara, J. Li, D. Alistarh, J. Liu, and C. Zhang, “Zipml: An end-to-end bitwise framework for dense generalized linear models,” *CoRR*, vol. abs/1611.05402, 2016.
- [4] W. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003.