Accelerating Graph Convolutional Networks

Adityanarayanan Radhakrishnan [1] **Emmanuel Mensah** [1]

1. Introduction and Related Work

With the success of convolutional networks for image processing and recurrent networks for language/sequential data processing, recent research has turned to constructing other architectures for different modes of data. In particular, several datasets involve the use of a known graph structure relating features of the data (i.e. knowledge graphs for search, Bayesian networks for causal inference, hierarchical models for genes etc.). In turn, we are interested in studying accelerators for the recently proposed graph convolutional networks, which take advantage of such graph structures when modeling the data.

In this work, we analyze data flows for graph convolutional networks and provide an algorithm to accelerate graph convolutional networks based on a clique approximation of the underlying graph. By decomposing the underlying graph into a set of cliques, the weight matrices corresponding to each layer will become block diagonal, and so we can process each block diagonal component independently. This technique has recently been used with convolutional networks on images as well (4), (1) by decomposing images into a set of patches and then training convolutional networks on patches independently. However, we are not aware of any specific accelerators for this task. The closest related work would involve handling sparsity in neural networks (i.e. zero-skipping in Eyeriss (2)).

An outline of our work is as follows. In Section 2, we describe the mathematical framework for our work. In Section 3, we first analyze the number of MACs saved by using graph convolution compared to fully connected networks, and then analyze tiling by cliques and describe a weight stationary dataflow for graph convolution. In Section 4, we use the simulator from lab 4 to estimate how much energy and cycles are saved by graph convolution in comparison to fully connected networks. We then conclude with a discussion of extensions to our work.

2. Setting

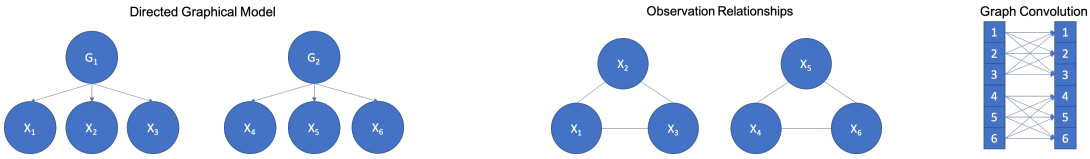
We consider the following motivating example of learning the generative factors producing a set of observations. Let $G_1 \sim U[-2,-1], G_2 \sim U[1,2]$ represent two random variables following uniform distributions with disjoint support. We now let G_1, G_2 serve as generators for a set of random variables $\{X_1, X_2, \dots X_6\}$ as follows:

$$X_1 = w_1 G_1 \ , \ X_2 = w_2 G_1 \ , \ X_3 = w_3 G_1$$
$$X_4 = w_4 G_2 \ , \ X_5 = w_5 G_2 \ , \ X_6 = w_6 G_2$$

where weights $w_i \in \mathbb{R}$. That is, X_1, X_2, X_3 are generated as multiples of G_1 and X_4, X_5, X_6 are generated as multiples of G_2. The directed graphical model for this problem is shown in Figure 1a. Now given observations of random variables $X_1, X_2, \dots X_6$, we wish to learn latent factors Z_1, Z_2 that act as proxies for G_1, G_2. That is, we wish for our learned latent factors to mimic the true generative factors as closely as possible. To this end, a first approach would be to solve this problem using a 1 hidden layer linear fully connected autoencoder with 2 hidden units. As shown in Figure 2, after training on 100 observations, visualizing the output of the 2 hidden units for 1000 test examples yields that the 2 hidden units (the latent factors) are actually (negatively) correlated. This is undesirable, as the true generative factors here are actually independent.

To resolve this issue, we take advantage of the graph structure and construct a graph convolution approach. Namely (following theory from graphical models (3)), we know that observing $X_1, \dots, X_6$ corresponds to the graphical model shown in Figure 1b. Hence, instead of using all the connections in a fully connected network, we keep only the connections between X_1, X_2, X_3 and the connections between X_4, X_5, X_6 (as shown in Figure 1c). After training on the same data as the fully connected network, the graph convolutional network produces a representation that almost exactly matches the true generative factors (shown in Figure 2).

In the example shown, there is a clear reduction of MACs in the graph convolutional network as compared to the fully connected network. In particular if m denotes the number of edges in the graph structure and n denotes the number of vertices, then the number of multiplications in the graph convolutional network is exactly $2m+n$. The proof follows from the fact that each node has a number of connections equal to its degree plus one connection to itself. If the graph itself is fully connected (i.e. $\binom{n}{2}$ edges), then this formula gives n^2 multiplications just like in the standard fully connected layer.



(a) The directed graphical model for generating observations.

(b) The undirected graphical model for observations.

(c) Graph convolution.

Figure 1. Example of graph convolution.

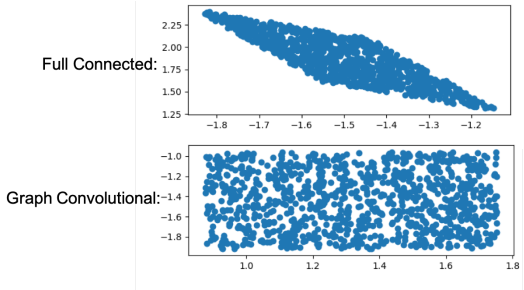


Figure 2. Learned latent representation of fully connected and graph convolutional networks. The graph convolutional network learns the true independent generative factors nearly perfectly while the fully connected correlates its latent factors.

In the setting described thus far, our true graph was decomposable into cliques allowing for a nice factorization of multiplications in the graph convolutional network into computation on cliques (i.e. only inputs $1, 2, 3$ are needed for outputs $1, 2, 3$ in our setting). However, in general the graph structure may not be decomposable into disjoint cliques. To address this issue, we first transform approximate our graph structure using a disjoint set of cliques (for example splitting the graph greedily into maximum cliques). Such approaches are similar to the ensemble approaches used in image classification when images are chunked into independent patches prior to classification (4), (1). For the rest of this work, we assume that our graph structure is decomposed into disjoint cliques, and in the next section, we provide a simple accelerator for graph convolutional networks.

3. Accelerating Graph Convolution through Weight Stationary Dataflow

Decomposing a graph into cliques allows for the following simple idea for accelerating convolutional networks: we can tile our computation based on clique size. For simplicity, we begin with the case we process cliques of equal sizes and postpone a discussion of utilization concerns for cliques of several sizes to the end of the section.

Suppose that our underlying graph structure can be composed into $\mathcal{C}$ cliques of size c. Then from the previous section, we know that the number of non-gated MACs for the graph convolutional network and fully connected network will be:

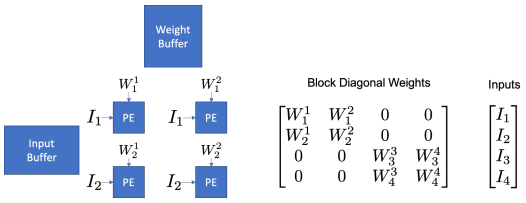
$$2\mathcal{C}\binom{c}{2} + \mathcal{C}c = \mathcal{C}c^2 \quad \text{for graph convolution} \tag{1}$$

$$(c\mathcal{C})^2 = c^2\mathcal{C}^2 \quad \text{for fully connected} \tag{2}$$

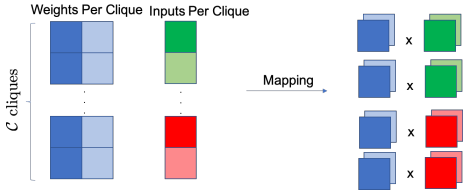
Now, a naive way to accelerate graph convolutional architectures is by just using an accelerator for a fully connected layer, but with a sparse (block diagonal) weight matrix corresponding to the clique decomposition. Using this approach will save energy through zero-gating, but we will still have comparable energy usage to the fully connected case in buffering 0s to the PEs.

To avoid this issue, we consider a weight stationary dataflow in which we instead buffer only weights of cliques and tile our computation by cliques (shown in Figure 3a). Namely, we want the weights of as many cliques as possible to be delivered to each PE and then multi-cast input coordinates across PEs corresponding to a given clique. Next, to make use of the PE array in the simulator in lab 4, we perform the following mapping (shown in Figure 3b). Namely, as the PEs in lab 4 are for convolutional networks with C input and M output channels, we instead reshape our weights and input to be of sizes $1 \times 1 \times c \times c$ and $1 \times 1 \times c \times 1$ respectively (i.e. the input is transformed to be like a 1-dimensional convolutional kernel across the weights).

In order to properly simulate our clique decomposition in the lab 4 simulator, we needed to ensure that only clique weights were being delivered. However, as we were unable to modify the lab to appropriately deliver just the required operands, we made the following simplification to upper bound the amount of cycles and energy needed for computation: we considered only processing one clique at a time in the simulator and then multiplying the cycles and energy by the number of cliques $\mathcal{C}$. Although this may upper bound the number of cycles and energy, in general when the clique sizes can be different, this approach leads to severe under-utilization of the PE array. To remedy this problem, we note that it is possible to group cliques together in software prior to delivery on PEs so that several cliques can



(a) Weight stationary dataflow for clique processing: weights form a block diagonal matrix and are loaded onto PEs. Inputs corresponding to a clique are multicast to the PEs.



(b) Transformation of clique processing into one dimensional convolution for processing in lab 4 simulator. Multiplication are treated as 1D convolution of inputs across clique weights.

Figure 3. Description of dataflow and mapping for clique processing.

be processed at once if the number of weights across all cliques are less than the size of the PE array.

4. Evaluation

We now use lab 4 to provide a comparison of how our clique processing technique reduces the amount of energy and cycles required for inference in a graph convolutional network when compared to a fully connected network with no sparsity and a fully connected network with sparsity indicated by the graph structure.

We begin with a comparison of the number of cycles required to process a graph convolutional network with 2 equally sized cliques and a fully connected network with no known sparsity (i.e. the example shown in Section 2) on a 4 x 4 PE array. In Table 1, we see that as input size (and thus clique size) increase, the number of cycles is dominated by computation (as opposed to data delivery) and so we see a roughly 2x savings in cycles and energy required to process graph convolutional networks.

However, we now show that even with sparsity, clique processing significantly reduces the number of cycles and energy required for graph convolutional networks. In Table 2, we see that when processing on a single PE (ws_PE or ws_PE_sparse in lab 4) that there is roughly a 2x speedup and 1.5x speedup in the number of cycles required by our clique processing approach and the sparse neural network without zero skipping and with zero skipping respectively. Note that the energy for MACs is the same in all settings

Input Size	Clique Size	Network	Cycles	Energy
8	4	GCNN	70	1162
8	-	FCNN	73	2016
18	9	GCNN	276	6284
18	-	FCNN	352	7116
32	16	GCNN	462	10142
32	-	FCNN	883	17027
128	64	GCNN	7062	128214
128	-	FCNN	14203	255419

Table 1. Comparing total energy consumption and number of cycles between Graph Convolutional Neural Network (GCNN) and Fully Connected Neural Network (FCNN) using a 4x4 PE array.

for a given inputs size as zeros are gated in the simulator. Hence, the energy savings are predominantely from the reduced scratchpad usage.

Input Size	Clique Size	Zero Skip Enabled	Network	Cycles	Total Energy	MAC Energy	Scratchpad Energy
8	4	No	GCNN	98	200	160	40
8	-	No	SCNN	193	240	160	80
8	-	Yes	SCNN	129	232	160	72
32	16	No	GCNN	1538	3200	2560	640
32	-	No	SCNN	3073	3840	2560	1280
32	-	Yes	SCNN	2049	3712	2560	1152
128	64	No	GCNN	24578	51200	40960	10240
128	-	No	SCNN	49153	61440	40960	20480
128	-	Yes	SCNN	32769	59392	40960	18432

Table 2. Comparing total energy consumption and number of cycles between Graph Convolutional Neural Network (GCNN) and Sparse Fully Connected Neural Network (SCNN) using a single PE element.

5. Conclusion

In this work, we present a novel accelerator for graph convolutional networks through clique approximation of the underlying graph structure. We present a weight stationary dataflow to process cliques of inputs at a time, and we present theory for the number of MACs reduced and simulations for the improvement in cycles and energy improvements with our technique.

Although there will be a tradeoff in accuracy and speedup when the clique approximation is not faithful to the true graph structure, in the cases where a true clique decomposition exists and is used, our method will provide a significant speedup without sacrificing accuracy.

6. Contributions

Emmanuel Mensah and Adit Radhakrishnan worked together on the tiling, dataflow, mapping, and making the workloads for the lab 4 simulator. Emmanuel worked on running the simulations for 1PE and PE arrays (Tables 2, 1). Adit worked on the theory, problem setting, and training GCNNs. Although the idea to use graph convolutional networks was based on work from Caroline Uhler's group, the idea to decompose a graph into cliques for efficient processing is new and just for the course.

References

[1] Wieland Brendel and Matthias Bethge. Approximating cnns with bag-of-local-features models works surprisingly well on imagenet. In *International Conference on Learning Representations (ICLR)*, 2019.

[2] Yu-Hsin Chen, Tushar Krishna, Joel Emer, and Vivienne Sze. Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. In *IEEE International Solid-State Circuits Conference, ISSCC 2016, Digest of Technical Papers*, pages 262–263, 2016.

[3] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models Principles and Techniques*, volume 1. MIT Press, 2009.

[4] Adityanarayanan Radhakrishnan, Charles Durham, Ali Soylemezoglu, and Caroline Uhler. Patchnet: Interpretable neural networks for image classification. In *NeurIPS Workshop on Machine Learning for Healthcare*, 2018.