Inference and Learning Hardware Architecture for Neuro-Inspired Sparse Coding Algorithm

Chester Liu and Zhengya Zhang
Department of Electrical Engineering and Computer Science
University of Michigan, Ann Arbor, MI, 48109-2122
Email: cwhliu@umich.edu, zhengya@umich.edu

***Abstract*—In this paper we present three hardware architectures designed to accelerate the inference operation of a neuro-inspired sparse coding algorithm. The memory and communication requirement of the three architectures are compared, and we show that one architecture outperforms the other two in scalability. A hardware system consists of an accelerator and a general purpose processor is proposed for the inference and learning operation. Two optimizations are proposed to further improve the overall performance by skipping unnecessary computations and autonomously learning the feature set.**

I. Introduction

Neuro-inspired sparse coding algorithm extracts salient features from the input; for instance, features in the shape of edge and dot are extracted from images [1]. The algorithm is agnostic to the input type and has been applied to other types of input such as audio [2] and video [3]. In extracting the features, as shown on the left in Fig. 1, the input is encoded into a sparse representation using a feature set that has been learned for the input type. The sparse representation can be processed by a downstream post-processing for applications such as classification and recognition. The input can be reconstructed, or decoded, from the sparse representation, as shown on the right in Fig. 1.

Recently, there is a growing interest in hardware acceleration of this type of algorithm [3]–[5], as it creates sparse output, which provides great opportunity for optimization, and it is trained by unsupervised learning, making it suitable for applications that require continuous learning. In this paper, we compare the memory and communication requirement of three accelerator architectures (Section III), followed by two optimization techniques proposed to improve the overall system performance (Section IV).

II. Sparse Coding

For a set of input data, sparse coding learns a feature set, and finds a sparse representation for each individual input so as to minimize the coding error between the original input and the one reconstructed from the sparse representation. In sparse coding, an input is represented, or encoded, by a coefficient vector containing many zero values. An input can also be reconstructed, or decoded, as a linear combination of the features with the coefficients. Mathematically, sparse coding can be described by the following optimization problem

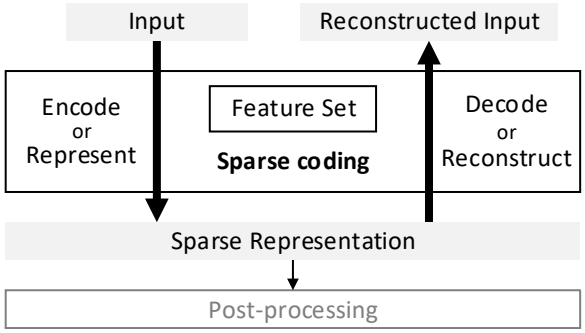

Fig. 1. Sparse coding operations overview

$$\min_{\mathbf{\Phi},\mathbf{a}} \left(\|\mathbf{x} - \mathbf{\Phi a}\|^2 + \gamma S(\mathbf{a}) \right) \tag{1}$$

where $\mathbf{x}$ is an input, $\mathbf{\Phi a}$ is the reconstructed input in which $\mathbf{\Phi}$ is the feature set and $\mathbf{a}$ is the coefficient vector, S is an activation function used to enforce sparsity by penalizing non-zero coefficients, and γ controls the weighting between the coding error $\|\mathbf{x} - \mathbf{\Phi a}\|^2$ and the sparsity penalty $S(\mathbf{a})$.

One of the earliest sparse coding algorithms solves the optimization of (1) using conjugate gradient descent (CGD) method [1]. However, a hardware implementation of CGD is inefficient. Rozell *et al*. introduced a sparse coding algorithm, named locally competitive algorithm (LCA) [6], that makes use of a neural network with recurrent connections to optimize (1). Compared to CGD, the recurrent neural network (RNN) used by LCA can be parallelized and mapped to hardware efficiently, making it more practical for hardware implementations.

Given a set of input data, *learning* refers to the optimization of (1) over the feature set. And for a given input, the process of determining the coefficients that minimize (1) is called *inference*. In LCA, before the optimal feature set is found, the optimization of (1) alternates between inference and learning. In performing the inference operation, (1) is optimized over $\mathbf{a}$ while $\mathbf{\Phi}$ is held constant. During the learning operation, $\mathbf{\Phi}$ is updated while $\mathbf{a}$ remains fixed. Once the optimal feature set is learned, it can be fixed for the inference operation, and

978-1-5386-3603-9/18/$31.00 ©2018 IEEE

the learning operation can be disabled unless an incremental learning is required to adapt the feature set to new inputs.

A. LCA Inference Operation

Inference for LCA is carried out over iterations. In the RNN used by LCA, each neuron retains a neuron potential that is updated in each iteration by the following equation

$$u' = u + \eta \Delta u \tag{2}$$

where u is the current potential, Δu is the potential update, u' is the new potential, and η is a user-defined step size. Depending on the application, a typical inference converges in a few tens of iterations, and the inference result is defined as $\mathbf{a} = T(\mathbf{u})$ in which

$$T(u) = \begin{cases} u - \lambda, & \text{if } u > \lambda \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

where T is an activation function and λ is a user-defined threshold value that is used to control the sparsity: larger λ results in higher sparsity, i.e., more zero values.

The computation of the potential update consists of a feedforward term for the input to excite the neurons, and a feedback term for the neurons to inhibit or "compete" with each other to represent the input. For a network containing N neurons, the potential update vector is calculated by

$$\Delta \mathbf{u} = \Phi^T \mathbf{x} - (\Phi^T \Phi - I)\mathbf{y} - \mathbf{u} \tag{4}$$

where $\mathbf{x}$ is a M-dimensional input vector, Φ is a $M \times N$ feature set matrix in which each column stores a neuron's feature, I is a $N \times N$ identity matrix, and $\mathbf{y}$ is equal to $T(\mathbf{u})$, which is a N-dimensional neuron output vector. In (4), $\Phi^T \mathbf{x}$ accounts for the feedforward excitation which measures feature's resemblance to the input; feedback inhibition between two neurons is stronger if they share similar feature, and is described by $-(\Phi^T \Phi - I)\mathbf{y}$ in which the identity matrix removes self inhibition as LCA requires features to have unit L2 norm; and $-\mathbf{u}$ is the neuron potential leakage. Equation (4) can be rearranged into

$$\Delta \mathbf{u} = \Phi^T (\mathbf{x} - \Phi \mathbf{y}) + \mathbf{y} - \mathbf{u} \tag{5}$$

Note that inhibitions between neurons are calculated in (4), whereas in (5) a reconstruction is carried out. Both (4) and (5) are in vector form as they calculate the potential update for all neurons. For an individual neuron it is calculated by

$$\Delta u_i = \boldsymbol{\phi}_i^T \mathbf{x} - \sum_{j=1, j \neq i}^{N} ((\boldsymbol{\phi}_j^T \boldsymbol{\phi}_j) y_j) - u_i \tag{6a}$$

$$\Delta u_i = \boldsymbol{\phi}_i^T (\mathbf{x} - \sum_{j=1}^{N} (\boldsymbol{\phi}_j y_j)) + y_i - u_i \tag{6b}$$

where (6a) and (6b) are the scalar form of (4) and (5), respectively.

B. LCA Learning Operation

Learning for LCA is unsupervised as it self-adapts the features Φ to better represent the input without requiring any label. Rozell *et al*. mapped the optimization of (1) over Φ to the recurrent neural network and derived the following learning rule

$$\begin{aligned} \Phi' &= \Phi + \alpha \Delta \Phi \\ \Delta \Phi &= (\mathbf{x} - \Phi \mathbf{a}) \mathbf{a}^T \end{aligned} \tag{7}$$

where $\Delta \Phi$ is the feature update, Φ' is the updated feature set, and α is a user-defined learning rate. For a given input, inference is performed before learning can be carried out as the inference result, i.e., $\mathbf{a}$, is required in calculating the feature update. In (7), $\Phi \mathbf{y}$ is the reconstructed input and $(\mathbf{x} - \Phi \mathbf{y})$ represents the coding error. Intuitively, in performing learning, (7) can be understood as minimizing the coding error introduced by each individual neuron.

III. Inference Hardware Architectures

In this section we compare three different architectures designed to accelerate the inference operation for LCA. The first two architectures are designed based on (6a). Neurons in these architectures have fully-connected feedback connections with other neurons, and the feedback computation is distributed to the neurons. The two architectures differ in the type of data sent from neurons for feedback computation. Neurons in the third architecture, which is based on (6b), are connected to a centralized hub responsible for the feedback computation and there is no connection between neurons.

A. Distributed Feature Feedback Architecture (DFFA)

In this architecture, the potential update is calculated using (6a), and each neuron retains a feature, a potential, and an output coefficient, as illustrated in Fig. 2a. In computing the feedforward excitation, the input is sent to all neurons, and is multiplied with the feature of each individual neuron in parallel. When a neuron's potential exceeds the threshold, namely $y > 0$, the neuron inhibits other neurons by broadcasting its feature, which will be used by other neurons in computing the feedback inhibition.

The total memory storage requirement for features in this architecture is $O(NM)$, as each neuron needs to store its own feature. To support broadcast, a neuron requires M connections to the input and $(N-1)M$ connections to the other neurons, requiring in total $O(NM)$ feedforward connections and $O(N^2 M)$ feedback connections. By distributing the feedforward and feedback computation to neurons, this architecture achieves high modularity and parallelism. However, this architecture has limited scalability since the number of feedback connections grows quadratically with the neuron number.

B. Distributed Coefficient Feedback Architecture (DCFA)

This architecture shares a similar structure with DFFA, as illustrated in Fig. 2b. The feedforward computation is the same as in DFFA, and the potential update is also calculated using

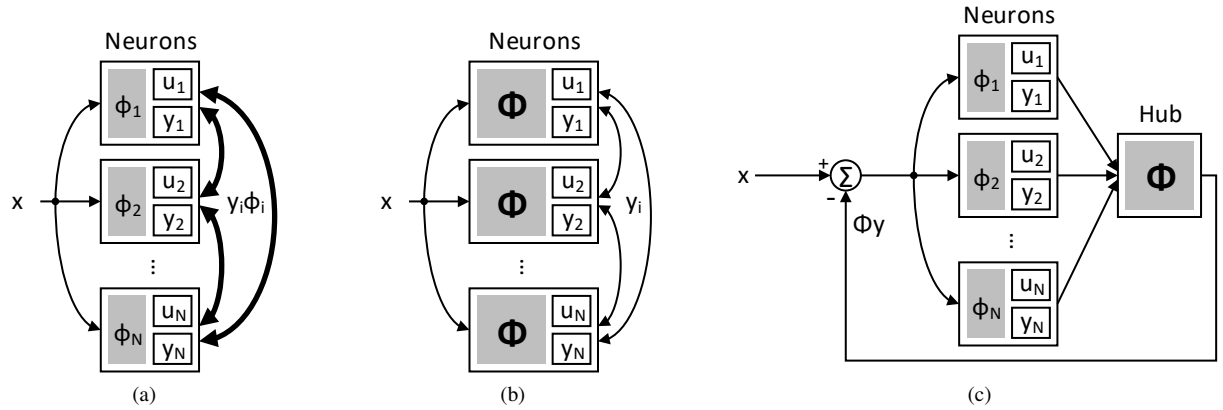

Fig. 2. Three accelerator architectures for LCA inference operation: (a) Neuron features are broadcast for a distributed feedback computation. (b) Neuron output coefficients are broadcast for a distributed feedback computation. (c) Neuron output coefficients are sent to the hub for a centralized feedback computation.

TABLE I
COMPARISON OF MEMORY AND COMMUNICATION REQUIREMENT

Architecture	Memory	Feedforward connection	Feedback connection
DFFA	$O(NM)$	$O(NM)$	$O(N^2M)$
DCFA	$O(N^2M)$	$O(NM)$	$O(N^2)$
CRFA	$O(NM)$	$O(NM)$	$O(NM)$

(6a). But unlike DFFA, in which each neuron only retains its own feature, neurons in this architecture retain the entire feature set. When a neuron's potential exceeds the threshold, the neuron broadcasts its output, which will be used by other neurons to lookup the feature set in computing the feedback inhibition.

This architecture achieves the same level of modularity and parallelism as in DFFA. Compared to DFFA, this architecture requires less feedback connections, i.e., $O(N^2)$, as features are not broadcast from neurons. As a trade-off, the total memory requirement for features is increased to $O(N^2M)$ in order to store the feature set in every neuron. Implementations based on this architecture typically lead to a better hardware utilization than DFFA. An ASIC design based on this architecture has demonstrated very high throughput for a network of 256 neurons [7]. This architecture, however, also has limited scalability since the memory requirement grows quadratically with the neuron number.

C. Centralized Reconstruction Feedback Architecture (CRFA)

Neurons in this architecture retain their own feature. A centralized hub which retains the entire feature set is added and connected to all neurons, as illustrated in Fig. 2c. The potential update is calculated using (6b), of which the feedforward and the feedback computation are carried out by the neurons and the hub, respectively. In performing the feedforward computation, the coding error, i.e., the difference between the original input and the reconstructed input, is sent to all neurons. Unlike the two distributed architectures, in which neurons are fully-connected with each other, neurons in this architecture are only connected to the hub and there is no inter-neuron connection. When a neuron's potential exceeds the threshold, the neuron sends its output to the hub. In performing the feedback computation, the hub collects output from neurons and reconstructs the input, which will be used in the next iteration.

Each neuron and the hub require $O(M)$ and $O(NM)$ memory storage for the feature and the feature set, respectively, requiring in total $O(NM)$ feature memory storage. The number of feedforward and feedback connection in this architecture are both $O(NM)$. This architecture achieves the best scalability by combining the advantage of the two distributed architectures: (1) storing individual feature in neurons for the feedforward computation, and (2) having another copy of the feature set to reduce the number of feedback connections. Compared to the distributed architectures, which can be made fully parallel and each neuron has similar workload, the hub in this architecture could become the performance bottleneck as the workload of the hub is significantly higher than the neurons. In preventing the hub from becoming the bottleneck, [5] designed an event-driven reconstruction to reduce the computation carried out at the hub by taking advantage of the sparsity. To further improve the throughput, a globally-asynchronous locally-synchronous structure is deployed in [5] so as to balance the throughput between neurons and the hub by adjusting their clock frequency.

IV. PROPOSED HARDWARE SYSTEM

A hardware system consists of an accelerator and a general purpose processor, as illustrated in Fig. 3, is proposed to accelerate the LCA inference and learning operation. Inference operation is carried out by the accelerator implemented using CRFA, which is the most scalable architecture discussed in Section III. The processor incorporated in the system can be programmed to control the accelerator and perform learning operation that follows (7). The inference and learning efficiency is further improved by two optimizations discussed in the following subsections.

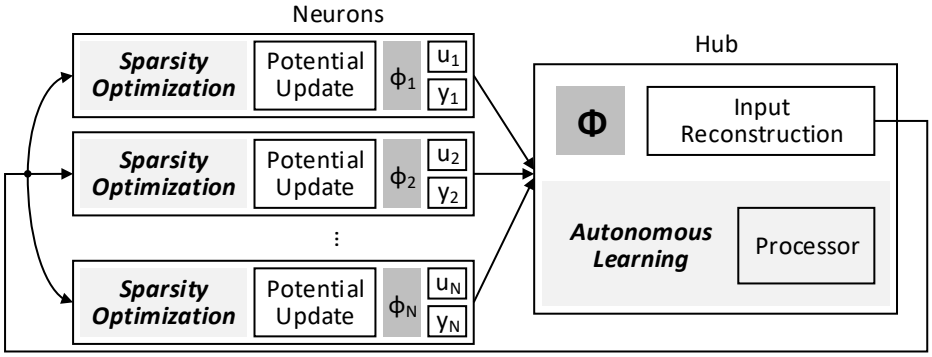

Fig. 3. The proposed hardware system consists of a CRFA accelerator for the inference operation and a general purpose processor for the learning operation.

A. Sparsity Optimization for Inference

In performing inference, typically the excitation to most of the neurons is substantially lower than the threshold, leading to a high sparsity in neuron output, i.e., only a few neurons will have nonzero output. Prior work has demonstrated that sparsity can be exploited to improve computation efficiency by time-sharing critical hardware resource [7] or by skipping redundant computations [5], [8]. However, they only focused on improving the feedback computation efficiency. Here we show that sparsity can also be exploited to improve feedforward computation efficiency for CRFA.

In the first iteration, all neuron outputs are zero, i.e., $y_i = 0$; therefore, (6b) is reduced to $\Delta u_i = \boldsymbol{\phi}_i^T \mathbf{x}$, which is defined as the excitation. If the excitation to a neuron is r times smaller than the threshold, where r is the number of iterations, then the neuron potential will not accumulate above the threshold so the neuron can be disabled to reduce power consumption. A user-specified sparsity can be enforced by disabling more neurons. For example, if the target sparsity is s, then neurons with excitation smaller than $s\lambda/N$ are to be disabled. Disabling extra neurons helps further reduce the power consumption at the cost of suboptimal inference result.

B. Autonomous Learning

Conventionally learning is viewed as a separate process from inference. For example, a convolution neural network (CNN) based design typically does not support on-chip learning as the computation of learning is too costly and is drastically different from inference. Learning for LCA can be readily supported by the inference accelerator based on CRFA as the reconstructed input, which requires the most computation in the learning operation, is already computed by the hub during the inference operation. In performing the learning operation, the processor is programmed to compute the feature update for each neuron by multiplying the reconstructed input with the neuron output. The feature update is scaled and accumulated to the current feature by the processor. The hub in CRFA performs reconstruction in every iteration, meaning that, compared to [7] which is implemented using DCFA and an explicit reconstruction step is needed in performing learning, the added hardware for learning in the proposed system is minimal.

Interestingly, the LCA learning operation can be made autonomous by adding a simple MAC that calculates the reconstruction error as the hub performs reconstruction. Learning can be automatically triggered when the reconstruction error exceeds a user-defined threshold value. And if the reconstruction error is sufficiently larger than the threshold, then a different set of features may need to be loaded.

V. Conclusions

In this paper, we compare three accelerator architectures for LCA inference operation. A hardware system consists of an accelerator and a general purpose processor for LCA is presented. Two optimization techniques, i.e., feedforward sparsity optimization and autonomous learning, are proposed to further improve the system performance.

Acknowledgment

This work was supported in part by SONIC, DARPA UPSIDE and Intel Corporation.

References

[1] B. A. Olshausen and D. J. Field, "Emergence of simple-cell receptive field properties by learning a sparse code for natural images," *Nature*, vol. 381, no. 6583, pp. 607–609, June 1996.

[2] R. Grosse, R. Raina, H. Kwong, and A. Y. Ng, "Shift-invariant sparse coding for audio classification," in *Twenty-Third Conference on Uncertainty in Artificial Intelligence*, July 2007, pp. 149–158.

[3] C.-E. Lee, T. Chen, and Z. Zhang, "A 127mW 1.63TOPS sparse spatio-temporal cognitive SoC for action classification and motion tracking in videos," in *IEEE Symposium VLSI Circuits*, August 2017, pp. 226–227.

[4] J. K. Kim, P. Knag, T. Chen, and Z. Zhang, "Efficient hardware architecture for sparse coding," *IEEE Transactions on Signal Processing*, vol. 62, no. 16, pp. 4173–4186, August 2014.

[5] C. Liu, S.-G. Cho, and Z. Zhang, "A 2.56mm^2 718GOPS configurable spiking convolutional sparse coding processor in 40nm CMOS," in *IEEE Asian Solid-State Circuits Conference (A-SSCC)*, November 2017, pp. 233–236.

[6] C. J. Rozell, D. H. Johnson, R. G. Baraniuk, and B. A. Olshausen, "Sparse coding via thresholding and local competition in neural circuits," *Neural Computation*, vol. 20, no. 10, pp. 2526–2563, October 2008.

[7] J. K. Kim, P. Knag, T. Chen, and Z. Zhang, "A 640M pixel/s 3.65mW sparse event-driven neuromorphic object recognition processor with on-chip learning," in *IEEE Symposium VLSI Circuits*, June 2015, pp. 50–51.

[8] P. Knag, C. Liu, and Z. Zhang, "A 1.40mm^2 141mW 898GOPS sparse neuromorphic processor in 40nm CMOS," in *IEEE Symposium VLSI Circuits*, June 2016, pp. 180–181.