Fast and Accurate Sparse Coding of Visual Stimuli with a Simple, Ultra-Low-Energy Spiking Architecture

Walt Woods and Christof Teuscher
Department of Electrical and Computer Engineering
Portland State University, Portland, OR, USA
{wwoods, teuscher}@pdx.edu

Abstract—**Memristive crossbars have become a popular means for realizing unsupervised and supervised learning techniques. In previous neuromorphic architectures with leaky integrate-and-fire neurons, the crossbar itself has been separated from the neuron capacitors to preserve mathematical rigor. In this work, we sought to simplify the design, creating a fast circuit that consumed significantly lower power at a minimal cost of accuracy. We also showed that connecting the neurons directly to the crossbar resulted in a more efficient sparse coding architecture, and alleviated the need to pre-normalize receptive fields. This work provides derivations for the design of such a network, named the Simple Spiking Locally Competitive Algorithm, or SSLCA, as well as CMOS designs and results on the CIFAR and MNIST datasets. Compared to a non-spiking model which scored $33\,\%$ on CIFAR-10 with a single-layer classifier, this hardware scored $32\,\%$ accuracy. When used with a state-of-the-art deep learning classifier, the non-spiking model achieved $82\,\%$ and our simplified, spiking model achieved $80\,\%$, while compressing the input data by $92\,\%$. Compared to a previously proposed spiking model, our proposed hardware consumed $99\,\%$ less energy to do the same work at $21\times$ the throughput. Accuracy held out with online learning to a write variance of $3\,\%$, suitable for the often-reported 4-bit resolution required for neuromorphic algorithms; with offline learning to a write variance of $27\,\%$; and with read variance to $40\,\%$. The proposed architecture's excellent accuracy, throughput, and significantly lower energy usage demonstrate the utility of our innovations. This work provides a means for extremely low-energy sparse coding in mobile devices, such as cellular phones, or for very sparse coding as is needed by self-driving cars or robotics that must integrate data from multiple, high-resolution sensors.**

Index Terms—**sparse coding, locally competitive algorithm, memristors, neuromorphic architecture, spiking architecture**

I. Introduction

Sparse coding, accomplished through algorithms that encode an input stimulus in a new basis with few non-zero elements, has been shown to yield excellent image classification accuracy [1]. These algorithms have also been shown to reduce the learning time required for backpropagation [2]. Since there are few non-zero elements, sparse coding also provides a means for minimizing the bandwidth required to transfer sensor data amongst multiple processors, or to store that data in long-term storage. These algorithms have gained traction in recent years as a result of these benefits coupled with biological evidence that the V1 visual layer in mammalian cortices performs similar functionality [3]–[5].

The implementation of neuromorphic algorithms on custom *Application-Specific Integrated Circuits* (ASICs) has been a wildly popular area of study largely due to the development of programmable, variable-resistance nanodevices, named memristors, that can be used to realize the synapses needed in a more compact, energy efficient form [3], [4], [6]–[11]. Sparse coding algorithms belong to the neuromorphic family, and this work set out to design an energy-efficient, sparse coding ASIC using memristors.

Our work extends from the *Locally Competitive Algorithm* (LCA) proposed by Rozell *et al.* in 2008, an optimal solver for the sparse coding problem [12], coupled with Oja's rule, used to repeatedly tune the dictionary towards an optimal solution for a set of training inputs [13]. The LCA was chosen as the most promising sparse coding algorithm due to its use of inhibitory forces to force an optimally sparse and stable solution. The sparse code from the LCA was passed to a supervised layer, which was trained to classify the stimulus either as the value of a handwritten digit or the type of object in a tiny image: MNIST and CIFAR-10, respectively [14], [15]. This approach leverages the transformation of input data from its native space to a decorrelated space via the LCA, and then uses traditional machine learning techniques to classify the stimulus based on its decorrelated representation. We have used this approach in the past [8]. In practice, there are several benefits to this approach: improved accuracy due to the stability of the decorrelated representation (an effect similar to dropout), and the ability to compress the input stimulus between the measuring device and the identification layer. A single frame of HD video data consists of approximately $6.2\,\mathrm{Mbit}$ of information, which the LCA could compress down to $1.0\,\mathrm{Mbit}$, an $84\,\%$ reduction, at a *Root-Mean-Square Error* (RMSE) of $5.8\,\%$, or down to $0.6\,\mathrm{Mbit}$, a $90\,\%$ reduction, at an RMSE of $7.0\,\%$. For video surveillance systems or autonomously driving vehicles, this means that several cameras could be wired into a single, high-speed, low-energy sparse coding device, greatly reducing the needed communication bandwidth for the system.

The LCA provides competition amongst inputs to minimize the number of active outputs and simultaneously maximize the fidelity of the reconstruction produced. The exact function minimized by the LCA, with the input stimulus as $\mathbf{s}$, the sparse code as $\mathbf{a}$, the reconstruction as $\hat{\mathbf{s}}$, and a cost function $\lambda C(\cdot)$ that expresses the trade-off between reconstruction quality and the sparsity of the final solution $\mathbf{a}$, is written as:

$$E(t) = \frac{1}{2}||\mathbf{s}(t) - \hat{\mathbf{s}}(t)||^2 + \lambda \sum_m C(\mathbf{a}_m(t)). \qquad (1)$$

$E(t)$ is minimized by descending an *Ordinary Differential Equation* (ODE) derived by Rozell *et al.* [12] to its natural steady state:

$$\dot{\mathbf{u}}_m(t) = \frac{1}{\tau}\left[\mathbf{b}_m(t) - \mathbf{u}_m(t) - \sum_{n \neq m} \mathbf{G}_{m,n}\mathbf{a}_n(t)\right], \qquad (2)$$

where $\mathbf{u_m}$ is an underlying state for the mth neuron that is thresholded to produce the resulting sparse code element $\mathbf{a_m}$, $\mathbf{b_m}$ is the inner product of the mth neuron's receptive field and the input, and $G_{m,n}$ is the inner product of the mth and nth receptive fields, minus 1 if $m = n$. Approximating the integral of Eq. (2) with sufficiently small step sizes will always end in a stable $\mathbf{u}$, corresponding to a generated sparse code $\mathbf{a}$. See the original LCA paper for more details [12].

While an effective equation, prior attempts at implementing the LCA directly in hardware have suffered from a few details which prevented an efficient implementation. For example, the LCA's matrix form can be intuitively described as reproducing the input based on a linear combination of the weight matrix columns. The most significant term limiting the LCA's efficiency comes from the inhibition term in Eq. (2): each output column's coefficient's ODE depends on all other output columns. For a naive implementation, and indeed the one chosen by Rozell's group and presented in Shapero *et al.* [16], this implies $\mathcal{O}(N^2)$ hardware scaling: doubling the number of output elements quadruples the required hardware. Additionally, a low-power implementation of the dot product in the ODE is non-trivial: it had to either be implemented digitally, or using next-generation components like memristors. While variable-resistance nanodevices like memristors can compute a dot product using little power themselves, to make the computation accurate requires either a virtual ground, which consumes significant power due not only to the matching current but also due to excessive current drained through low-resistance devices [17], [18], or requires a tuning resistance that must change based on each column's configuration [18]. The additional power consumed by these solutions motivates the exploration of techniques that do not require the calculation of an exact dot product.

In this work we set out to provide a low-power, hardware-friendly realization of an LCA-like algorithm that used a spiking framework. To maximize power savings, the model was simplified and the drawbacks of that simplification were investigated. Spikes were utilized to save power; the efficacy of spikes for power saving was explored. The proposed architecture has been named the *Simple Spiking Locally Competitive Algorithm* (SSLCA). The SSLCA was compared with the original LCA's ODE [12] as well as their later spiking work in Shapero *et al.* [16]. Both the MNIST and CIFAR-10 datasets were used for the comparison. Through these comparisons, we found that our proposal demonstrated excellent power and scaling qualities. It is our hope that this work provides the basis for efficient sparse-coding hardware in next-generation computers and robotics.

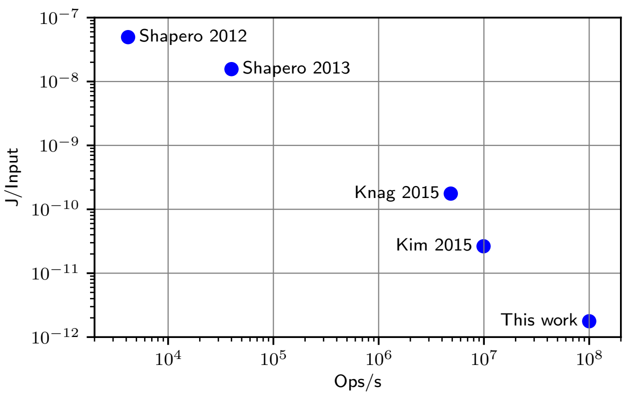


Fig. 1: Comparison of the SSLCA's energy efficiency and throughput, presented in this work, with previous state-of-the-art results. One "Op," or operation, is the complete generation of a sparse code from a single set of inputs.

II. Related Work

Due to the development of nanodevices that can function as hardware synapses, such as memristors, and the popularity of neuromorphic algorithms, there has been a lot of prior work relating to ASICs for neuromorphic sparse coding architectures. The relative performance of these in terms of energy efficiency and throughput are shown in Fig. 1. Note that the throughput axis is presented in "Ops/s" as opposed to "Inputs/s," so that the scale of an architecture has no effect on the measures reported. For example, if an architecture were built to process 64 inputs, and its performance were measured, doubling that architecture to 128 inputs would immediately double its throughput measurement if we used "Inputs/s" rather than "Ops/s." Using "Ops/s" makes the number of inputs irrelevant, which is more appropriate for architectures which have a fixed processing time regardless of the number of inputs and outputs.

Throughputs presented are inference-only; throughputs when using online training would be different in accordance with the time to set the different storage mediums used in each algorithm. As an example in this work, many memristive models can be set within 2 ns to 20 ns [17]. If every algorithm execution led to a weight update of each device in the matrix, and each update happened serially, this would lead to e.g. 392 µs being added to each loop of the matrix for an example network with 784 inputs and 50 outputs. However, this quantity may easily be amortized: either multiple devices might be updated at once, multiple runs of the algorithm could be integrated into each weight update, or only a portion of the weights would need to be updated each iteration. For all neural algorithms, disabling learning results in significantly higher throughput. Furthermore, when the problem at hand is sufficiently solved, there is no further need for the learning step, an often-used motivation for offline learning in practical applications.

The original work on LCAs was Rozell *et al.*, 2008 [12]. Rozell *et al.* sought to improve upon prior sparse coding algorithms by deriving an optimal expression to both minimize the sparse equation (Eq. (1)) and smooth the generated sparse representation when given time-varying input. Their work derived an ODE that solved both of these problems. However, a hardware implementation of this ODE would scale with $\mathcal{O}(N^2)$ operations as an artifact of its inhibitory term (Eq. (2)). Furthermore, digital implementations would require many iterations to stabilize the ODE, while analog implementations would require constant, voltage-scaled inputs, and a power-hungry virtual ground for each output neuron [8]. The LCA was implemented using analog signals and subthreshold currents on a *Field Programmable Analog Array* (FPAA) with floating gates in 2012 by Shapero *et al.*, a group including C. Rozell [19]. While it demonstrated power consumption scaling of only $\mathcal{O}(N\sqrt{N})$, the required hardware still scaled as $\mathcal{O}(N^2)$, and convergence was relatively slow, occurring after $240\,\mu\mathrm{s}$.

These drawbacks were addressed to some extent later by Shapero *et al.* in 2013 [16]. That work extended the original LCA to a spiking architecture, referred to in this work as the *Spiking Locally Competitive Algorithm* (SLCA). The motivation for spiking largely seems to have stemmed from biology: all biological systems appear to use spiking rather than constant signals [3], [4], [20]. Spiking models have also long been believed to consume less power, and to exhibit additional computational power due to their stochasticity [21]–[23]. The validity of leveraging spikes to save power is discussed further in Section IV-A2 of this work. In their work, Shapero *et al.* [16] showed that their SLCA consumed more power than their LCA at small sizes, but that their SLCA scaled only as the desirable $\mathcal{O}(N)$, and would consume less power than the LCA at large network sizes. Additionally, they reduced the convergence time to $25\,\mu\mathrm{s}$, nearly $90\,\%$ faster than their LCA with a throughput of $40\,\mathrm{kOps/s}$. However, the required hardware still scaled as $\mathcal{O}(N^2)$.

Other spiking networks optimized for sparse coding have been published, such as SAILnet, introduced by Zylberberg *et al.* in 2011 [24]. ASICs using this architecture have been studied, with a substantial reduction in power compared to the approach presented by Shapero *et al.* [16]. Knag *et al.* [25] were capable of using the SAILnet architecture to process images using only $48\,\mathrm{pJ/input}$ for their inference logic with a throughput of $0.55\,\mathrm{MOps/s}$, or using $176\,\mathrm{pJ/input}$ with a throughput of $4.8\,\mathrm{MOps/s}$, $120\,\times$ as fast as Shapero *et al.* [16]. Their design was CMOS-based, and utilized a decreased resolution for weight storage: 4 bits per excitatory or inhibitory weight. This decision has been justified in a number of prior works dealing with how much accuracy is needed for sparse coding algorithms to perform well [8], [26]. This design was later reconfigured by Kim *et al.* [27]. They tested higher clock speeds, optimized the design, and generated less detailed output, resulting in a throughput of $9.9\,\mathrm{MOps/s}$ at an energy efficiency of $26.4\,\mathrm{pJ/input}$ [27]. However, these numbers benefit from their usage of only 256 output neurons to represent 1024 inputs, whereas Knag *et al.* used 256 output neurons to represent only 256 inputs [25], [27]. Not only does using fewer outputs result in a worse encoding of the input, but due to the $\mathcal{O}(N^2)$ scaling properties of these networks, this also favors the power and throughput figures in [27]. As such, subsequent results in this work will be compared with Knag *et al.*, as their chip performs a comparable amount of work to ours [25]. Like the LCA, SAILnet uses a direct inhibitory weight between each pair of output neurons, yielding a scaling complexity of $\mathcal{O}(N^2)$.

The closest family of algorithms that does not exhibit $\mathcal{O}(N^2)$ scaling is *Spike-Timing-Dependent Plasticity* (STDP). STDP exploits what is known as "Hebbian" learning, where input spike events that occur at the same time as an output spike event become more likely to trigger that output spike event. The common idiom for this behavior is, "neurons that fire together, wire together." In effect, each output neuron learns to activate when a correlated set of inputs fires together. This is very similar to what happens in sparse coding, where a neuron responds to a specific pattern in the input. The primary differences are that STDP makes no effort to preserve the information found in the input and STDP does not implement inhibition amongst neurons. Rather, the purpose of STDP is to flag which features are present in the input and how prevalent they are, without regard for the other features present. Sparse coding, on the other hand, will suppress output of a feature that is already represented by a combination of other features. Both techniques are a form of unsupervised learning, except sparse coding requires some inhibitory terms while STDP does not. This gives STDP the desirable quality of $\mathcal{O}(N)$ scaling. Due to its excellent scaling properties, STDP was used in one of the earliest attempts to replicate the features found in mammalian visual cortices [3], has been explored as an autoencoder [28], and has been used to generate unsupervised features for digit classification on the MNIST digit database [6]. STDP is also one of the dominant architectures researched using next-generation nanodevices such as memristors [4], [6], [7], [9], [10], [20]. The downside of an STDP approach to input encoding is that more output neurons are required due to the lack of inhibition; with 50 neurons, prior research showed that STDP achieved 80% accuracy on MNIST, while a sparse coding layer using LCA achieved 85% [6], [8]. If a sparse coding algorithm were implemented with the same efficiency as STDP, it would be the preferred method of unsupervised training, as it conveys more depth of information. That is what our work set out to accomplish: to close the gap between STDP and sparse coding algorithms such that there exists a simple and efficient algorithm for sparse coding.

Recent work by Sheridan *et al.* showed that their group has manufactured memristive crossbars and applied voltage across

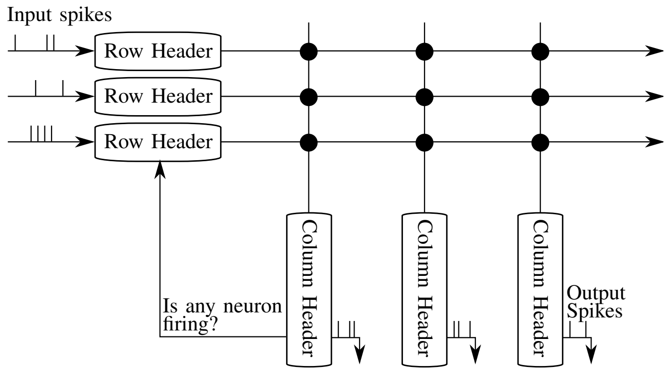


Fig. 2: High-level architecture for the SSLCA. During inference, input spikes pass through a Row Header. Voltage is forwarded from the Row Headers to a nanowire crossbar with memristors at each junction. Current is allowed to pass through each memristive junction and is used to charge or discharge an LIF neuron in each Column Header (Fig. 4). When any LIF neuron spikes, an output spike is propagated and inhibitory forces are passed back through the crossbar to the Row Headers. Only a single shared bit ("Is any neuron firing¿') is required. The count of output spikes across any given time window describes the sparse code for the input pattern seen during that time window.

the network to calculate the similarity coefficients in the LCA equations [29], [30]. Sheridan *et al.* used a microprocessor to implement the majority of the LCA, and did not include comprehensive throughput and power information [29], [30]. However, a fundamental result of their work is that memristive devices have sufficient resolution and accuracy to implement LCAs on real devices [29]. Our work extends their work by proposing a means of implementing the entirety of the LCA on the same chip as the memristive crossbar with few additional components. The Sheridan *et al.* work also advocated using a *Winner-Take-All* (WTA) approach to training the weight matrix: rather than updating the weights for all columns participating in a reconstruction, they only updated the largest contributor [29]. While effective, using WTA was motivated largely by the supposition that a single neuron's firing would dominate the response to most stimuli; however, with inhibition and larger, more complicated inputs, this is not the case.

III. Model

In light of issues with previous hardware implementations and the potential benefits of sparse coding algorithms discussed in Sections I and II, we set out to develop the simplest architecture for sparse coding that would exhibit $\mathcal{O}(N)$ scaling, utilize inhibition, and emphasize low-power operation. While any device whose resistance can be modified in-situ would suffice, memristors from Lu *et al.*'s group were chosen due to their nanoscale form factor and ability to be fabricated in tight crossbars [29]. These devices additionally exhibit a low on:off ratio, which has been associated with devices that possess better long-term storage and analog qualities [17], [29]. It is also worth noting that while the internal state of memristive devices could change at any voltage, it changes very slowly at the $0.7\,\mathrm{V}$ used during inference for the SSLCA. Further discussion of the time-varying qualities of memristive devices can be found in [17]; for this architecture, we generally assume that the memristive devices used are static during the inference step due to the low voltages used, and are adjusted in a separate training step that implements Oja's rule, or in a loading process that configures the chip with weights learned during offline training.

As an initial step, we established that the chosen architecture should fit the form shown in Fig. 2. Assuming good accuracy could be derived, such an architecture would be sufficient for implementing sparse coding with the desired traits. Such an architecture would clearly exhibit $\mathcal{O}(N)$ scaling. Inhibition could be implemented with a backwards-pass through the same crossbar used to charge the output neurons. Low-power operation would stem from the simplicity of the architecture, its good scaling properties, and an innovation on the way the neurons were integrated into the architecture.

Neurons in this architecture differ from previously proposed architectures. Like prior work, the Column Headers implement *Leaky-Integrate-and-Fire* (LIF) neurons [9], [16]. They consist of a capacitor that charges (integrates) input events as they are active, and discharges (leaks) when input events are inactive. In contrast to those architectures, which use a separate resistor to control the leak rate, this architecture's LIF neurons both accrue and dissipate charge via the memristive crossbar. In addition to requiring fewer components, this configuration has proven more tolerant of un-normalized receptive fields, a phenomenon discussed in Section III-A.

Inference with this architecture begins with input spikes reaching the Row Headers. Input spikes were chosen due to biological inspiration, the promise of lower power consumption, and also partially because memristors exhibit vastly different resistances at different voltages; using spikes rather than voltage-scaled inputs helps to avoid this situation [17]. Additionally, keeping both the input and output to the algorithm as spiking helps increase the homogeneity of the system. This enables the architecture to encode not only an input stimulus but also e.g. the output of an STDP algorithm. Upon reaching the Row Headers, spikes are converted into voltages - one of V_{cc} or $0\,\mathrm{V}$ - which are applied to the nanowire crossbar. At each crossbar junction, a memristive device provides resistance between the Row Headers and the Column Headers. The resistance of each device is set such that, at V_{cc}, it is between R_{min} and R_{max}. The pattern of conductances formed by these memristive devices in each column are the *Receptive Field* (RF) of the corresponding neuron. An input pattern aligning with this field will trigger an output spike in this column before any other column. Current flows through these memristive devices into or out of the capacitor in each Column Header. When one of the Column Header capacitors reaches a given threshold, that column generates an output spike, and current flows from that column back through the corresponding memristive devices into the Row Headers to re-charge inhibitory forces. Once the output spike event finishes, the process repeats.

The derivation of necessary parameters was broken down into two stages: calculations without inhibition, and an extension of those calculations to incorporate inhibition. This divide was necessary to ensure the solution was tractable, and had the added benefit of deriving two versions of the architecture which were used to demonstrate the benefits of inhibition.

A. Uninhibited SSLCA

To begin the derivation for the Uninhibited SSLCA, we start with the equation for Rozell *et al.*'s LCA, Eq. (2), and remove the inhibitory term. What remains is a leaky dot product, with no $\mathcal{O}(N^2)$ scaling problem. However, the resulting equation also no longer possesses optimality guarantees.

Oja's rule, used in this work to train each neuron's RF, can be used to somewhat remedy the missing inhibitory term by adjusting multiple RFs to work together to reconstruct the input without inhibition [13]. Briefly, Oja's rule is that a neuron's receptive field will change proportionally to its output activity multiplied by the difference between the original input and the LCA's reconstruction: $\Delta w_{i,j} = \eta a_j r_i$, where $r_i = x_i - \sum_j w_{i,j} a_j$. This is identical to gradient descent with a loss function of r_i^2. When both $w_{i,j}$ and y_j are finite and bounded, as in a spiking network using memristors, this equation naturally reduces the weight for inputs that were over-represented, and increases the weight for inputs that were under-represented. Furthermore, increasing a weight results in an increase of the corresponding spike count. As a result, any given input produces a natural equilibrium of weight and spike count values. Either might saturate, but that is not a problem aside from the loss of some of the input's magnitude.

Even with this compensation, using a leaky dot product for sparse coding would be difficult in hardware due to the angle property of the dot product between two vectors $\mathbf{X}$ and $\mathbf{Y}$:

$$\mathbf{X} \cdot \mathbf{Y} = |\mathbf{X}||\mathbf{Y}|cos(\theta). \tag{3}$$

From Eq. (3), a larger magnitude in either vector could be used to compensate for a larger difference in angle. In other words, a maximally-conductive RF would generate more current than an RF that is a better match. Prior works have solved this issue by normalizing each RF [29]. Instead, we decided to solve this problem by creating a negative stimulus via inactive input channels, achieved by grounding them rather than using high impedance. The current through these channels would be proportional to the RF, meaning that missing activity where the RF is conductive would lead to a higher penalty. As a result, a maximally conductive RF is perfectly fine within the system. It will respond to inputs that are maximally valued, and not respond to inputs with a shape better matched by a different RF. Coupled with Oja's rule, a maximally conductive RF will adapt to not be maximally conductive if it could better represent a wider range of inputs with a more specialized shape. This is the reason that capacitors in this network are placed directly on the crossbar, rather than behind a diode or equivalent. Placing them directly on the crossbar allows accrued charge to dissipate when an input row is inactive.

Using the layout from Fig. 2, the Row Headers for the uninhibited SSLCA are simple passthroughs (directly connected to the crossbar), and the Column Headers are simply a capacitor and a Schmitt trigger that drains all capacitors once any one neuron's voltage exceeds V_{fire} volts. The partial derivative of any neuron's voltage is therefore:

$$C\frac{\partial V_{neuron}}{\partial t} = \sum_i (V_i(t) - V_{neuron})G_i, \tag{4}$$

where C is the capacitance of the capacitor, V_{neuron} is the current voltage of that capacitor, $V_i(t)$ is the ith input's voltage at time t (one of V_{cc} or $0\,\mathrm{V}$ depending on whether it is currently spiking or not), and G_i is the conductance of the memristive device connecting the nanowires of the ith Row Header and the neuron in question's Column Header.

This equation can be better reasoned about by assuming an input row's voltage V_i spikes to voltage V_{cc} with a mean activity of K_i, meaning that at any point in time the voltage is V_{cc} with probability K_i and grounded with probability $1-K_i$. Generally, we assume that analog inputs would be bounded on $[0, 1]$. These analog values would be converted to input spikes by taking a maximum spike duty cycle, K_{max}, and multiplying it by the input's intensity to produce K_i. Equation (4) can then be reduced via the Laplace transform to:

$$\begin{aligned} Q_1 &= \sum_i G_i, \\ Q_2 &= V_{cc}\sum_i K_i G_i, \\ C\frac{\partial V_{neuron}}{\partial t} &= Q_2 - Q_1 V_{neuron}, \\ V_{neuron}(t) &= \frac{Q_2}{Q_1}(1 - e^{\frac{-tQ_1}{C}}) + V_{neuron,t=0}e^{\frac{-tQ_1}{C}}, \end{aligned} \tag{5}$$

where $V_{neuron,t=0}$ is the neuron's voltage at $t = 0$. Q_1, the column's total conductance, and Q_2, a matching metric between the stored RF and the input pattern, arise as intuitive factors that affect the neuron's state. To establish the necessary values for C and V_{fire}, Q_1 and Q_2 need to be derived in a way that produces good results for the network's "average case." Empirically, we found that assuming both the input and stored RF have binary elements (even for analog problems) produced the best results: the K values are either 1 or 0, while the G values are either the minimum or maximum conductance of our memristive devices.

The resulting calculation for Q_1 and Q_2, required to determine both the network's trigger voltage V_{fire} and neuron capacitance C, is described in Algorithm 1. Though these calculations are based on a single sample of Q_1 and Q_2, our results showed that the network still worked well outside of these "average cases" (Section IV).

Sparse coding being the goal of this architecture, we also make the assumption that any spike event will reset all neuron charges to $0\,\mathrm{V}$, implying that each output spike only encodes input activity seen since the end of the previous output spike. Any input spikes in this reset window will be ignored by the system. Since the SSLCA is designed to produce multiple spikes for a single input stimulus, this is not a problem as the statistics of the input spikes give equal likelihood of a spike during the reset as during any other period of operation. Real-time (non-episodic) uses of the SSLCA would also not suffer from this implementation detail, assuming all changes to the input stimulus happen at a significantly lower frequency than the spiking frequency of the system. This phenomenon could be compared to standard sampling theory: one spike is a sample of the input leading up to that spike, and changing frequencies in the input that exceed some fraction of the frequency of the sample rate cannot be deduced.

The downside to this assumption is that the architecture becomes a one-hot system: a pattern of simultaneously-firing output spikes becomes impossible. Superficially, this is in contrast to some other work on stochastic computation with spiking neurons [21]–[23]. The network still encodes stochastic information in a single output spike: the input pattern represented is stochastic due to the phase of input spikes' duty cycles, and as such the corresponding output is stochastically selected. By not allowing a pattern of simultaneous output spikes, the number of representable input patterns in a single event is reduced. However, due to this stochasticity, we have found that collecting multiple output spikes over a period of time results in a stochastic pattern of output activity that accurately represents the input. This is functionally identical to the trade-off of memory for time in computation: we are reducing the memory of the momentary output of our architecture in exchange for longer runtime. For sparse coding, where the resulting code often needs to be stored or otherwise buffered, this is not an issue.

With the above assumption, all $V_{neuron,t=0} = 0$, and Eq. (5) can be rearranged to calculate C based on some Q_1, Q_2, V_{fire}, and t_{fire}, where V_{fire} and t_{fire} are the desired voltage and time at which an output spiking event should occur given the input and stored RF parameters that produce Q_1 and Q_2:

$$C = \frac{-t_{fire} Q_1}{\ln\left(1 - V_{fire}\frac{Q_1}{Q_2}\right)}. \tag{6}$$

As t_{fire} can be calculated from the desired hardware clock rate and number of spikes per patch, the remaining parameters needed to fully specify the uninhibited SSLCA are V_{fire}, Q_1, and Q_2. Our experiments produced the lowest reconstruction RMSEs when V_{fire} is calculated based on a thresholded max voltage from Eq. (5) with a Q_1 and Q_2 calculated for the desired minimum RF that the resulting sparse code can represent, and when the Q_1 and Q_2 for the calculation of C come from an average case of the data set used with the network. The exact procedure followed to calculate these values is described in Algorithm 2. The algorithm requires knowledge of the expected average value of a stored receptive field, Rf_{avg}, as well as an idea of the minimum input intensity that should trigger an output spike, Rf_{least}. After scanning across many different combinations of Rf_{avg} and Rf_{least}, we discovered that setting $Rf_{least} = (1 - e^{-1})Rf_{avg}$ typically yielded optimal results with regards to classification accuracy, as can be seen in Fig. 3; this relation was used throughout this work. Though Fig. 3 displays results only on CIFAR, we found a similar result shape with MNIST when trying different values of Rf_{avg} and Rf_{least}.

Following Algorithm 2, and substituting the resulting values into Eq. (6), all parameters for constructing the uninhibited network are defined, and the network might be built. Applying

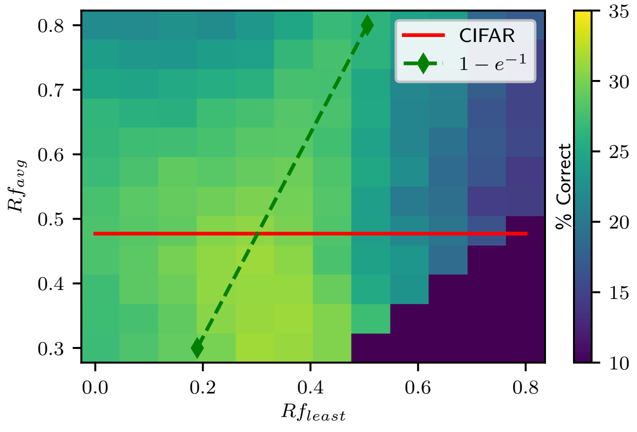


Fig. 3: Accuracy for the inhibited network on CIFAR-10 across different values of Rf_{least} and Rf_{avg}. Generally, setting $Rf_{least} = (1-e^{-1})Rf_{avg}$ was found to be a safe choice.

Algorithm 1: Process used to determine Q_1 and Q_2 given stored RF of average, relative conductance Rf_{stored} and a matching input of average, relative intensity Rf_{input}.

```
Input: $Rf_{stored}$, the average, relative conductance of the stored RF.
          This value must be on the interval $(\frac{G_{min}}{G_{max}}, 1]$;
       $Rf_{input}$, the average, relative intensity of all inputs;
       $G_{min}$, the minimum conductance of a crossbar device;
       $G_{max}$, the maximum conductance of a crossbar device;
       $K_{max}$, the proportion of time spent at $V_{cc}$ for an input signal
          spiking at its maximum rate;
       $N$, the number of inputs to the network.
Output: $Q_1, Q_2$
begin
    // Assumes that the stored RF consists
       entirely of elements at $G_{max}$ or $G_{min}$, and
       that the input pattern matches, but with a
       scaled intensity of $\frac{Rf_{input}}{Rf_{stored}}$. This
       simplification helps the network perform
       well with high-contrast RFs.
    $g_{min} \leftarrow \frac{G_{min}}{G_{max}}$;
    $I_h \leftarrow \frac{Rf_{stored} - g_{min}}{1 - g_{min}}$;          // Portion of inputs
                                                 at max-intensity.
    $I_l \leftarrow 1 - I_h$;
    $Q_1 \leftarrow N G_{max} Rf_{stored}$;
    $Q_2 \leftarrow N V_{cc} G_{max} K_{max} \frac{Rf_{input}}{Rf_{stored}} (I_h + I_l g^2_{min})$.
                        // Note that the $g^2_{min}$ term comes
                           from one $g_{min}$ multiplication
                           for $G_{max}$ and another for
                           $K_{max}$. This is the "match"
                           term between the RF and the
                           input.
end
```

Algorithm 2: Recommended process for selecting V_{fire}, Q_1, and Q_2, required for the calculation of C from Eq. (6).

```
Input: $Rf_{avg}$, the desired average, relative conductance of a stored
         RF. This value must be on the interval $(\frac{G_{min}}{G_{max}}, 1]$;
       $Rf_{least}$, the smallest average, relative input intensity is
         expected to trigger an output spike;
       $G_{min}$, the minimum conductance of a crossbar device;
       $G_{max}$, the maximum conductance of a crossbar device;
       $K_{max}$, the proportion of time spent at $V_{cc}$ for an input signal
         spiking at its maximum rate;
       $N$, the number of inputs to the network.
Output: $Q_1, Q_2, V_{fire}$
begin
    $V_{fire} \leftarrow (1 - e^{-1})\frac{Q_2}{Q_1}$, with $Q_1, Q_2$ from Algorithm 1 applied
      to $Rf_{stored} = Rf_{avg}$, $Rf_{input} = Rf_{least}$, other parameters
      matching;
    $Q_1, Q_2 \leftarrow Q_1, Q_2$ from Algorithm 1 applied to
      $Rf_{stored} = Rf_{avg}$, $Rf_{input} = Rf_{avg}$.
end
```

voltage spikes to the input lines of magnitude V_{cc} with a maximum duty cycle of K_{max} will cause the best-matching column to spike for t_{spike} seconds; collecting these spikes across a window of time (e.g. $10(t_{fire} + t_{spike})$ for an average of 10 spikes) will produce a reasonable reconstruction of the input based on the network's receptive fields. Results with the uninhibited SSLCA can be found in Section IV.

B. Adding Inhibition to the SSLCA

One of the original requirements deduced at the beginning of Section III was the need for inhibition. Prior works have shown the need for inhibition in an effective sparse coding system [6], [8], and Section IV-A3 of this work demonstrates this as well. While works such as that of Shapero *et al.* implemented inhibition by using additional hardware between each pair of neurons [16], leading to $\mathcal{O}(N^2)$ scaling, the SSLCA is designed in a way that allows for $\mathcal{O}(N)$ scaling.

Instead of additional hardware, a percentage of the SSLCA's running time is dedicated to calculating inhibitory forces. Whenever an output spike is generated, the duration of the output spike is used to pass current from the corresponding column back through the SSLCA's crossbar, charging capacitors in the Row Headers. Intuitively, the charges on these capacitors indicate how well represented the corresponding input signal is in the current reconstruction; overrepresented input signals will be suppressed. This is implemented through the Row and Column Headers shown in Fig. 4.

The Column Header for the Inhibited SSLCA is almost identical to that of the Uninhibited SSLCA: a standard LIF neuron setup, with the state capacitor connected directly (through a transmission gate) to the nanowire crossbar rather than being buffered. A crude schmitt trigger setup ensures that all output capacitors drain sufficiently when any neuron fires (firingAny), resetting the spike potentials. Additionally, if the current column is firing (fireSelf), the crossbar column is pulled up through a transistor. As a result, during each output spike, an inhibition current will flow back through the memristive crossbar into the Row Headers. The current

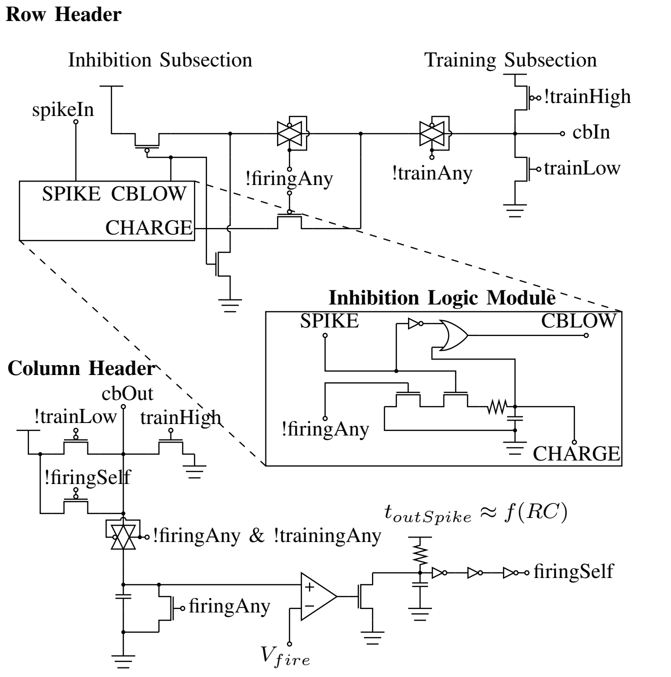


Fig. 4: The Row and Column headers needed to add inhibition to the SSLCA. The Row Header's responsibilities are to stop input spikes from reaching the crossbar when they are inhibited, and to keep track of the current state of the inhibitory forces. An Inhibition Logic Module is diagrammed as broken out from the main circuit for space reasons. The CHARGE port is responsible for sinking current from the crossbar when an output spike has occurred, and in turn charges the capacitor in the Inhibition Logic Module which prevents subsequent spikes from applying a voltage on the crossbar. After enough input spikes occur, the capacitor becomes sufficiently drained to apply voltage to the crossbar once more. The Column Header is much simpler and uses a transmission gate to direct current to and from the neuron's state capacitor. When any neuron fires, the capacitor is drained, and in the same column, V_{cc} is applied to the crossbar. A simple RC circuit cleaned up by several NOT gates is responsible for the output spike.

flowing into each Row Header will be proportional to the receptive field of the firing column.

The Row Header is more complicated, but the important aspect is that a capacitor storing the inhibition state discharges whenever an input spike arrives, and charges whenever an output spike occurs. The capacitor is charged through the crossbar junctions; the resistor for discharging the capacitor, referred to as R_{inhib}, is the one in the labeled Inhibition Logic Module. The stored inhibition state, when above $\frac{V_{cc}}{2}$, prevents input spikes from reaching the crossbar. $\frac{V_{cc}}{2}$ is chosen as it maximizes the linearity of the inhibitory response, since both charging and discharging occur at the same point on the exponential function (Eq. (7)).

Calibrating this architecture requires specifying both the capacitor, C_{inhib}, and the resistor, R_{inhib}, in the Inhibition Logic Module (Fig. 4). Ideally, a sparse coding algorithm should produce a stable, one-hot response to an input that exactly matches any of the stored RFs, and should combine several outputs when representing inputs that do not match a stored RF exactly. For simplicity, we focused on tuning the inhibitory components of the network to an input that matches the stored conductance of an RF, similarly to Algorithms 1 and 2. Additionally, we make room for inhibition in the spike cycle by using a neuron capacitance of $C_{cb} = f(C)$, where C is from Eq. (6) and $f(C)$ is an arbitrary function with value $0 < f(C) < C$. Using R_{cb} as the equivalent resistance of the memristive device used to charge the inhibitory force, and R_{inhib} as the resistance in the Inhibition Logic Module, we can write a few equations to describe the inhibition voltage V_i for a specific input i both before an output spiking event ($V_{i,pre}$) and after an output spiking event ($V_{i,post}$):

$$\begin{aligned} A &= \frac{1}{R_{cb}C_{inhib}}, \\ B &= \frac{K_i}{R_{inhib}C_{inhib}}, \\ V_{i,pre} &= V_{i,0}e^{-t_{fire}B}, \\ V_{i,post} &= V_{cc} + (V_{i,pre} - V_{cc})e^{-t_{spike}A}, \end{aligned} \tag{7}$$

where t_{spike} is the duration of an output spike, K_i is the portion of the time that the input being tracked is active, and $V_{i,0}$ is the voltage after an output spike. For a stable system with a uniform firing rate, $V_{i,0} = V_{i,post}$, and we are left with:

$$\begin{aligned} V_{i,0} &= V_{cc} + \left(V_{i,0}e^{-t_{fire}B} - V_{cc}\right)e^{-t_{spike}A} \\ &= \frac{V_{cc}\left(1 - e^{-t_{spike}A}\right)}{1 - e^{-t_{fire}B - t_{spike}A}}. \end{aligned} \tag{8}$$

With the inhibition voltage after a spike defined, one issue remains: so long as $V_{i,0} > \frac{V_{cc}}{2}$, the desired t_{fire} will no longer match t_{fire} without inhibition. There is always a period of time during which input spikes are inhibited, inflating t_{fire}. We label this period of time as t_{inhib}, and rewrite t_{fire} as $t_{inhib} + t_{collect}$, allowing Eq. (8) to be rewritten and a second equation for $V_{i,0}$ to be written by integrating backwards to $V_{i,0}$ from $\frac{V_{cc}}{2}$. These two equations are then combined to make a single equality, the solution of which indicates adequate values for R_{inhib} and C_{inhib}:

$$\begin{aligned} V_{i,0} &= \frac{V_{cc}\left(1 - e^{-t_{spike}A}\right)}{1 - e^{-t_{collect}B - t_{spike}A}}, \\ V_{i,0} &= \frac{V_{cc}}{2}e^{t_{inhib}B}, \\ \frac{V_{cc}}{2}e^{t_{inhib}B} &= \frac{V_{cc}\left(1 - e^{-t_{spike}A}\right)}{1 - e^{-t_{collect}B - t_{spike}A}}. \end{aligned} \tag{9}$$

Unfortunately, this formulation leaves two new variables, t_{inhib} and $t_{collect}$. Additionally, were we to use the original capacitance calculated in Eq. (6), we would miss the desired t_{fire} due to the added time for inhibition. To solve all of these problems, we set $C_{cb} = f(C) = \frac{C}{2}$. $t_{collect}$ is then solved for using the new neuron capacitance C_{cb} and Q_1, Q_2 from Algorithm 1 using $Rf_{stored} = Rf_{input} = Rf_{avg}$. t_{inhib} is then solved by subtracting $t_{collect}$ from t_{fire}. The remaining variable, R_{inhib}, is solved for by taking the log of both sides of the above equality (Eq. (9)), squaring the result, and minimizing the resulting function via Python's *scipy.optimize.minimize*, ensuring a near-zero result [31].

TABLE I: Example Parameters for Inhibited Network

Row	N	Rf_{avg}	G_{min} (µS)	G_{max} (µS)	V_{fire} (mV)	C_{cb} (fF)
1	192	0.40	4.8	19	87	1200
2	192	0.40	0.48	1.9	87	120
3	192	0.40	0.048	0.19	87	12
4	192	0.40	0.048	1.9	130	120
5	192	0.40	0.048	19	140	1200
6	48	0.40	4.8	19	87	290
7	48	0.60	4.8	19	120	430
8	48	0.80	4.8	19	130	580

Examples of the results from Algorithm 2 plus the inhibition transformations ($C_{cb} = f(C)$) can be seen in Table I. Notably, $N = 192$ corresponds to an input image of dimension $8 \times 8 \times 3$, while $N = 48$ corresponds to an input dimension of $4 \times 4 \times 3$. Row 1 is similar to the settings used in most of our experiments. While $C_{cb} = 1200$fF is significant, this number could be greatly reduced by future memristive technologies with greater resistance (rows 2 and 3). Rows 4 and 5 highlight that a higher ratio of G_{max} to G_{min} results in a higher V_{fire}, which would be helpful to overcome the comparator's input offset voltage and would allow the algorithm to better represent zero weights in each neuron's RF; both of these would increase the algorithm's effectiveness. A lower G_{max} may be artificially imposed on the network if the circuit designer wants less capacitance and is willing to sacrifice some of the accuracy that comes from a high G_{max} to G_{min} ratio. Rows 6 through 8 demonstrate the effects of fewer inputs, and of varying Rf_{avg}, the expected average stored RF in the network. As written, Rf_{avg} is also treated as the average input to the network; for very low G_{max} to G_{min} ratios, this might not make sense, and the actual average input value should be added as a separate input to Algorithm 2 and used in the final calculation of Q_1 and Q_2 to correct for the difference.

To validate this network design, we investigated different parameters other than Rf_{avg} for Rf_{stored} and Rf_{input} when optimizing the inhibitory response. Figure 5 demonstrates the results of this: while different combinations require different values of R_{inhib} to be completely accurate, choosing a single, median value works well in practice.

C. Training

The networks we used were trained using the ADADELTA algorithm in tandem with Oja's rule across two epochs of the training data, an identical approach to our prior work [8], [13], [32]. When considering the reconstructions for Oja's rule, we used the ratio of the conductance of each memristive device to G_{max}, the maximum expected conductance of a crossbar device. This approach limited the minimum representation of each input element in an RF to the inverse of the conductance on/off ratio of the memristive device. For our experiments, we used the Yang *et al.* device which featured a conductive on/off ratio of around 4 at 0.7 V [17]. We also tried training without this limitation (allowing the learned weight to drop all the way to 0, even though the device conductance would be set to 0.25), but did not find such a change to impact accuracy, although it

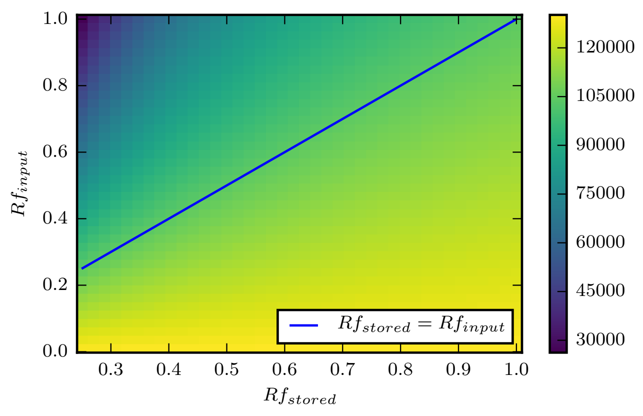


Fig. 5: Values of R_{inhib} needed to achieve the desired spike rate with different Rf_{stored} and Rf_{input} values (the spike rate scales with the ratio of Rf_{input} over Rf_{stored}). Ideally, the resulting plot would be flat, indicating a single value of R_{inhib} is sufficient for all cases. Since it is not flat, areas with larger than the chosen R_{inhib} will spike slower than expected, and areas with a smaller value will spike faster. In practice, we use Rf_{avg} for both (the blue line); receptive fields with a high stored value and a low input will under-spike, which should not be an issue as those regions should be better-covered by another neuron.

did affect the RMSE between the input and the reconstruction. Since the logical minimum does not affect the programmed conductance, this makes sense: the resulting sparse code is unchanged. The benefit of training with a non-zero minimum representable value is that the training could be done using only the memristive crossbar, without supplemental memory.

Homeostasis was used during training to encourage the network to use all available neurons, similar to prior work by Querlioz *et al.* [33]. If a neuron had not produced an output spike after several patches, V_{fire} was lowered for that neuron to encourage it to spike. This behavior was disabled for evaluating accuracy and RMSE.

For this work, all conductances were represented as analog values. We have conducted prior research that assumed a lower resolution of conductances would be achievable [8]. Currently available literature has shown that memristors might be trained within 1% of a target resistance [34], which is much better than the 4-bit resolution needed for good performance with neuromorphic algorithms [8], [26]. Note that a 4-bit resolution corresponds to $\pm 3.1\%$ write accuracy.

D. Models Used for Power and Accuracy Comparisons

The SSLCA was simulated algorithmically based on the above equations and algorithms. The simulator was written as a hybrid event/time-based simulator based on the maximum of the next predicted spiking event and a small window of time (2×10^{-18} s). Unless otherwise specified, our algorithm was configured to collect an average of 10 spikes per input, based on Fig. 6. Accuracy was computed with a *Single-Layer Perceptron* (SLP) network that was trained to associate resulting sparse codes with the category that generated them. This setup is efficient to compute, but does not rival the accuracy of a state-of-the-art deep learning architecture. A deep learning classifier was investigated in Section IV-A3.

While crossbar and capacitor power were calculated through these simulations, comparator power for the column headers was derived by simulating the 5 GHz comparator from Xu *et al.* 2011 at 4 GHz using a 0.7 V power supply, implemented with 45 nm CMOS transistors using the Predictive Technology Model published by Zhao *et al.* in 2006 [35], [36]. It was found that, per column, this setup added 2.2 µW.

Since we used Xu *et al.*'s comparator at 4 GHz [35], we configured the networks for an average spike accumulation period (t_{fire}) of 0.8 ns and an output spike duration (t_{spike}) of 0.2 ns. Input spikes were considered with a minimum period of 0.4 ns and an active duty cycle of $K_{max} = 0.5$ unless otherwise noted.

E. Example Code Availability

The simulation implementation used in this work was made available on Github at https://github.com/wwoods/tlab_sslca.

IV. Results

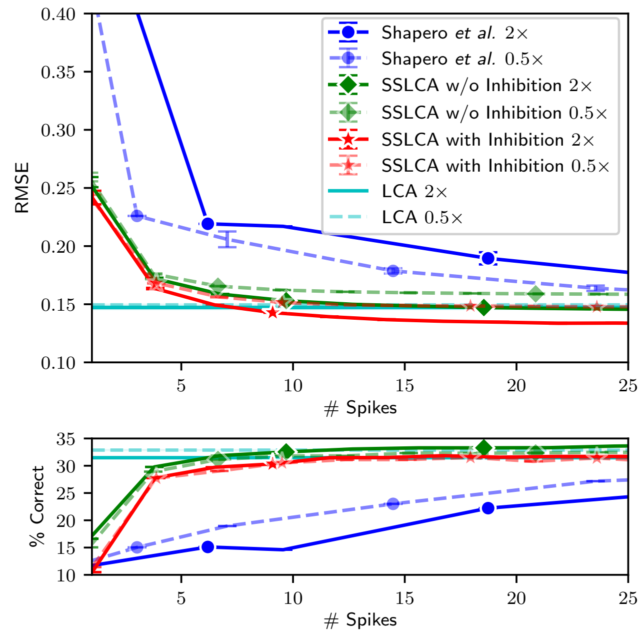
The SSLCA, SLCA, and LCA were tested with two different data sets to demonstrate the relative performance of the SSLCA. Reported RMSE values were generated as though zero were representable, and accuracies were from an SLP (discussed in Sections III-C and III-D). Experiments were run either 12 times, or until $\pm 10\,\%$ accuracy was achieved with $95\,\%$ confidence as per [37].

To show that our assumptions and simplifications did not result in significantly worse accuracy than the algorithms from which the SSLCA was derived, all results were compared with both the LCA and Shapero *et al.* [16]'s SLCA. An accuracy comparison across different numbers of spikes can be seen in Fig. 6. The LCA implementation is from equation (3.1) of Rozell *et al.*'s paper [12]; the SLCA implementation consisted of equations (5)-(7) in Shapero *et al.*'s paper [16]. Note that only the outputs of the SLCA network are spiking, while its inputs are constant voltages. Our work dealt with both spiking inputs and outputs.

Power numbers for the LCA come from (13) of Shapero *et al.*'s 2012 work [19] and scale as $\mathcal{O}(N\sqrt{N})$. Power numbers for the SLCA come from Shapero *et al.*'s 2013 work [16], and as that work included no built-in Vector Matrix Multiplier (VMM) as our algorithm does, we added the power from a memristor-based VMM to its figures. The throughputs of each of those architectures were several orders of magnitude lower than the SSLCA's (Fig. 1).

A. CIFAR-10

The first dataset, CIFAR-10, consisted of 60 000 32×32 RGB images, each containing one of 10 classes of objects [15]. For faster simulation and to demonstrate the scalability of each algorithm, these were scaled down to both 3×3 and 8×8. As the CIFAR-10 dataset contains equal numbers of each class, a simple accuracy was used to evaluate each algorithm's abilities.



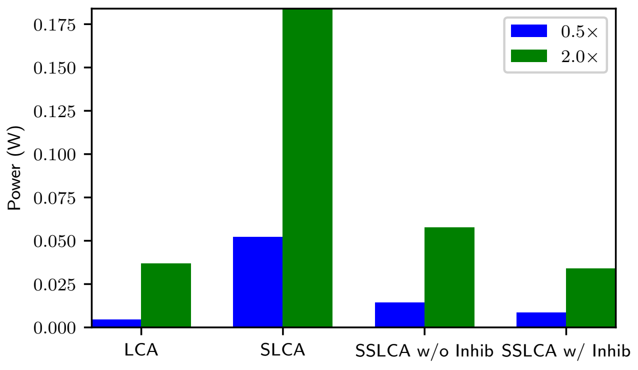


Fig. 6: Comparison of the LCA [12], SLCA [16], and the SSLCA on CIFAR-10 scaled to 8×8; suffixes indicate completeness ($2\times$ indicates 384 neurons, while $0.5\times$ indicates 96 neurons). While the SLCA achieves lower RMSE with significantly more spikes (around 100), for practical numbers of spikes the SSLCA produced much better results. The LCA performed better classification with fewer output neurons because it had slightly less output activity, which with a shallow classifier is more effective. A lower RMSE is more important for deep learning, seen in Fig. 13. While the LCA displayed promising power statistics for this problem, its throughput was four orders of magnitude smaller.

1) Accuracy: Compared to an optimal, analog implementation of the LCA, the SSLCA with inhibition matched performance on the 8×8 rescale of CIFAR-10 with a $3\,\%$ relative loss in accuracy ($33\,\%$ vs $32\,\%$; Fig. 7). The uninhibited SSLCA always produced a worse reconstruction than its inhibited counterpart, although for low Rf_{avg} (and correspondingly a higher number of spikes per patch) its classification accuracy was better with the simple SLP classifier. The trained network had an average spike count of 8 even though the architecture was configured for 10 spikes. The spike duty cycles were $K_{in} = 0.5$ and $K_{out} = 0.2$, where K_{in} is the maximum duty cycle of input spikes (and will be scaled by each input's

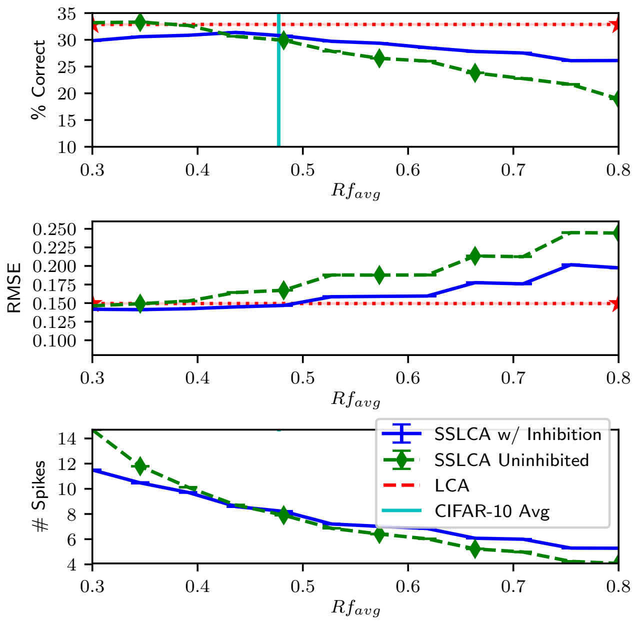


Fig. 7: SSLCA accuracy targeting 10 spikes on CIFAR-10 rescaled to 8×8 with and without inhibition, compared to LCA. Lower Rf_{avg} tended to produce lower RMSE due to increased activity in the resulting sparse code, and increased spike count (since the input intensity is greater than the target, spikes happen more frequently than calibrated). Inhibition ubiquitously reduced the RMSE, although with an SLP, its classification accuracy was less than the uninhibited version for darker receptive fields. See Section IV-A3 for the impact of RMSE when using a deep classifier.

intensity), and K_{out} is the expected duty cycle of the output spikes. That is, an output spike spikes for $K_{out}(t_{fire} + t_{spike})$.

The performance seen on the 3×3 and 8×8 rescales are compared in Fig. 8. In both instances, the performance of the LCA is approached by the SSLCA. However, the value of Rf_{avg} that optimizes accuracy is not obvious based on the problem's statistics; using the dataset average works well for the 3×3 case, but the 8×8 case requires a smaller Rf_{avg} in order to encourage more output activity, which translates into higher accuracy. At values of Rf_{avg} approaching the memristive device's minimum, the algorithm breaks down, as seen by the decreasing accuracy. This result can be explained through Eq. (6) and Algorithm 2: small Rf_{avg} results in a lower Q_1 and thus a smaller C, reducing the smoothing of input spike activity and in turn producing less consistent patterns of output spikes.

Another facet investigated was how different K factors (spike duty cycles) affected the overall classification accuracy of the system. The result is shown in Fig. 9; generally, a higher input duty cycle K_{in} performed better, and a lower output duty cycle K_{out} performed better. Intuitively this makes sense: larger duty cycles for input spikes means that more spikes are expected to work together when forming a single output spike; smaller duty cycles for output spikes means more time spent collecting input spikes, and thus each output spike represents a better average of the input spikes triggering it.

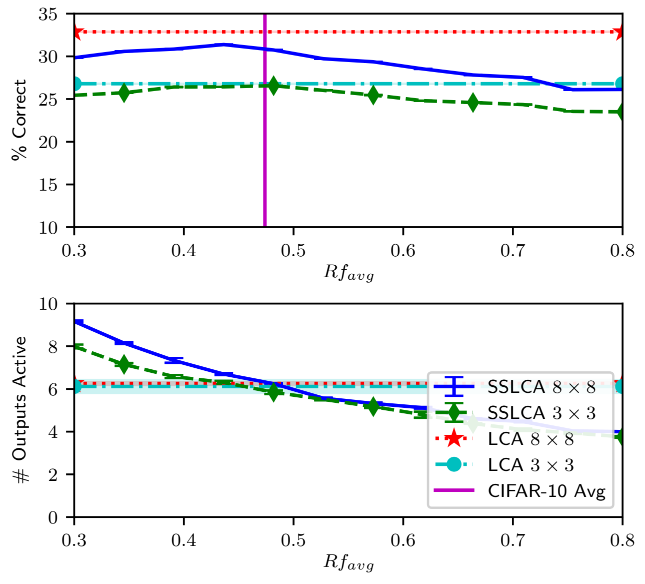


Fig. 8: A look at the difference between CIFAR-10 scaled to 3×3 and 8×8. In both cases, the SSLCA approached the LCA's accuracy. Unfortunately, the required setting of Rf_{avg} to maximize accuracy is not intuitive. The lower plot shows the sparsity of the output for each configuration; like the original LCA's λ threshold, a combination of the Rf_{avg} parameter of the SSLCA and the number of spikes collected may be used to control the sparsity of the output.

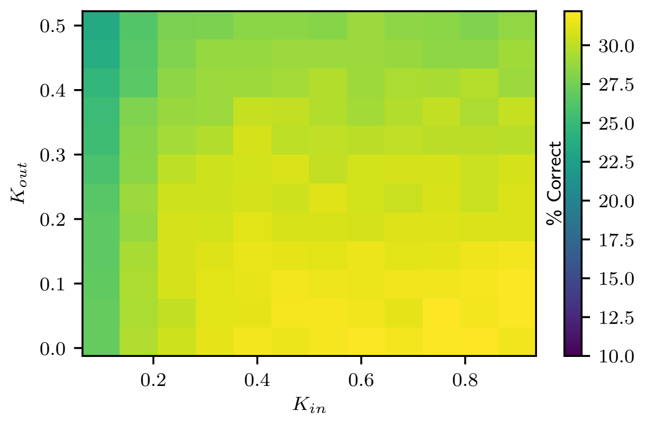


Fig. 9: CIFAR-10 spike accuracy with different spike duty cycles. High duty cycles for input spikes and low duty cycles for output spikes are the most accurate, but require more power (Fig. 12).

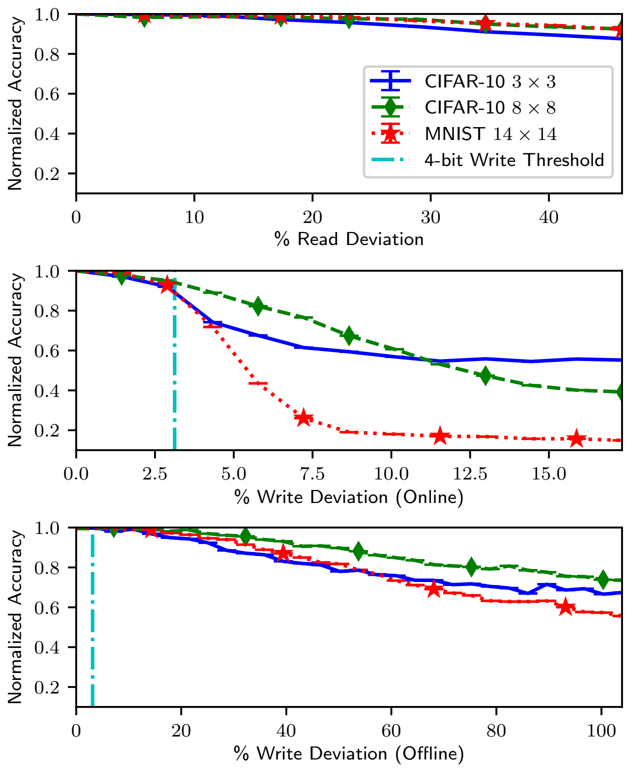


Fig. 10: The effects on the SSLCA of conductance variability during each read cycle (the period of time between two output spikes), during each write cycle (online training), or when a weight matrix learned offline is written to the memristive crossbar (offline training). Our results showed that unmitigated write deviations become serious for online algorithm stability after 3 %. However, using offline training or modifying the training approach as previously reported helps significantly [8]. For CIFAR-10 3×3, 8×8, and MNIST 14×14, $Rf_{avg} = 0.46, 0.425, 0.35$, respectively. Accuracies were normalized based on performance without deviations.

Device variability was also considered. Previous work has demonstrated significant variance from one read to the next [38]. To test how the SSLCA performed with imperfect hardware, we implemented three types of conductance deviations: read deviation, write deviation using offline training, and write deviation with online training. Read deviation was recalculated after every output spike to better simulate the time-varying nature of read randomization, and varied the effective conductance of a device uniformly by $\pm 0\,\%$ to $80\,\%$ (a standard deviation of $0\,\%$ to $46\,\%$). Write deviation with offline training consisted of training the model without variance, and then varying the conductance uniformly by $\pm 0\,\%$ to $180\,\%$ (not allowed to drop below 0 S; a standard deviation of $0\,\%$ to $104\,\%$). Write deviation with online training was applied after each application of Oja's rule, and modified the target conductances uniformly by $\pm 0\,\%$ to $30\,\%$ (a standard deviation of $0\,\%$ to $17\,\%$).

These results can be seen in Fig. 10. Neither read variability nor offline-trained write variability were found to have a significant impact. For online training, write variability could be tolerated up to $3\,\%$. This result is satisfactory for 4-bit learning, as described in Section III-C. Should better variability resistance be required, prior work on imperfect weight updates has indicated that sensitivity to these deviations might be further mitigated with a more aggressive training regimen that deliberately changes the magnitude of weight updates for greater effect [8].

2) Power: The inhibited SSLCA exhibited extremely low power consumption on the CIFAR-10 task scaled to 8×8 with 128 neurons ($2\times$ completion); at the optimal $Rf_{avg} =$

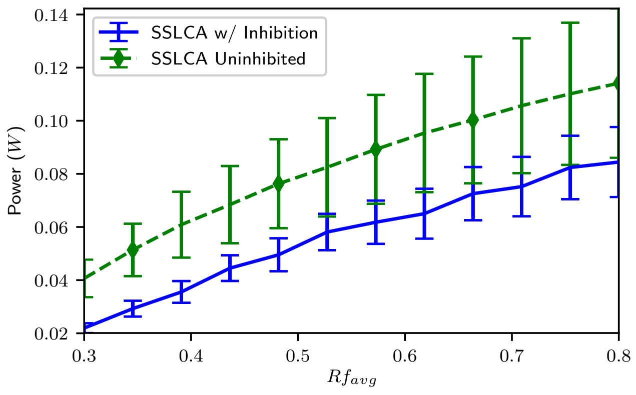


Fig. 11: Power for 8×8 CIFAR-10. Inhibition produced lower, more consistent power consumption due to its suppression of input spikes. Higher values of Rf_{avg}, which encourage the network to learn more conductive RFs, consumed more power accordingly.

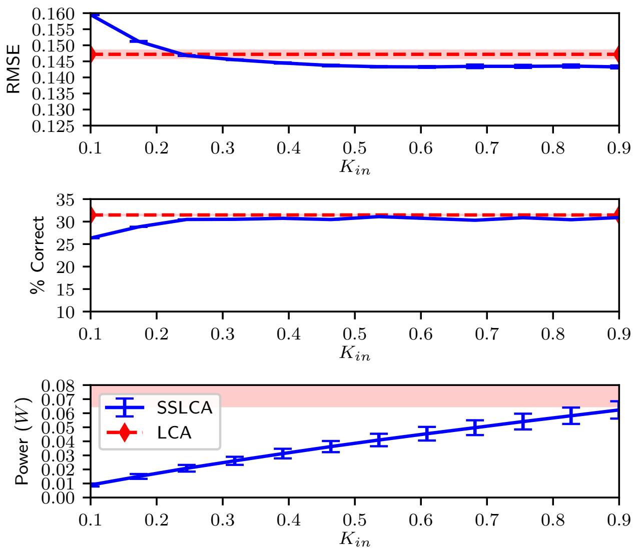


Fig. 12: Power and accuracy trade-offs on CIFAR-10 with varied K_{in}. LCA power shown represents only a voltage-scaled crossbar, to directly compare spiking and non-spiking approaches. Spiking always consumed less power than a voltage-scaling approach, a combination of the dataset having a high average input and the spiking algorithm utilizing inhibition of input signals (see Fig. 11 for the effects of inhibition on power consumption).

0.43, the consumption was just $1.77\,\mathrm{pJ/input}$ (Fig. 11) with a throughput of $100\,\mathrm{MOps/s}$. Compared with prior work such as Knag *et al.* [25], whose lowest energy consumption was $48\,\mathrm{pJ/input}$, this was a $96\,\%$ reduction in energy consumption for $180\times$ the throughput during inference [25]. At their high throughput ($310\,\mathrm{MHz}$), the SSLCA exhibited a $99\,\%$ reduction in energy consumption with a still substantially improved $21\times$ throughput.

Spiking architectures are often considered to produce power savings, though the extent of these savings has been a topic of discussion for some time [21]; we investigated that claim in Fig. 12. Except for very large duty cycles, the spiking architecture's crossbar used less power than the non-spiking, voltage-scaled crossbar. With a spiking implementation like the SSLCA, where input spikes are suppressed during an output spike, we would have expected the spiking to consume less power so long as $(1-K_{out})K_{in} < Rf_{input}$. This is a result of average power scaling with the square of voltage versus linearly with a duty cycle. The SSLCA surpasses this expectation due to the additional input spike suppression implemented through the inhibition mechanism. Interestingly, the standard deviation for the SSLCA's power was also substantially lower, probably as a result of columns in the SSLCA not being grounded, unlike the LCA, which sinks all current into a virtual ground [8], [17].

3) Information Retention for Deep Learning: While an SLP might be used in practice due to the simplicity of its implementation, it does not adequately express the depth of the information contained in the input dataset. To determine how much useful information was retained by both the LCA and SSLCA encodings, we used these architectures to encode augmented, full-size 32×32 CIFAR-10 input images using convolutions of different sizes. The convolved, sparse coded input images were then passed as input to a state-of-the-art deep learning architecture, the DenseNet-BC, presented by Huang *et al.* in 2016 [39]. This network architecture consists of a number of dense blocks that each halve the scale of the input data; within each dense block are many more layers, each accepting as input all previous layers within the dense block. Using this setup with 3 dense blocks and parameters $L=190, k=40$, Huang *et al.* achieved $96.54\,\%$ accuracy on CIFAR-10 (with data augmentation) [39]. See Huang *et al.* for more further details on these parameters.

We tested our architecture by dividing the input CIFAR-10 image into non-overlapping patches of $S\times S$, and encoding each patch using either the LCA or the SSLCA. For example, $S=4$ implies that the 32×32 CIFAR-10 image was broken into 8×8 non-overlapping regions of 4×4; each region was then sparse coded, and the resulting "image" consisting of all such encodings was passed to the DenseNet. For the $2\times$ networks with $S=4$, this means that rather than receiving each image as a $32\times32\times3$ spatial array, we passed in an $8\times8\times96$ spatial array. For the $0.5\times$ networks, the spatial array passed would only have a depth of 24. The SSLCA was configured with $Rf_{avg}=0.45$.

In order to allow each DenseNet a similar amount of expression for its classification, we parametrized the DenseNet so that the final dense block would output a 4×4 spatial array; the original paper's final block output an 8×8 array. To accomplish this, each DenseNet had a number of dense blocks $B=-1+\log_2\frac{32}{S}$. To hold the number of computations that each DenseNet performed roughly equivalent, we chose $L=40$, $k=12$, and the number of filters on the initial convolution before the first dense block was $k_0=6S^2B$ rather than 16. The limitation of this approach is that the number of tunable parameters becomes significantly larger with larger values of S, creating a greater potential for overfitting.

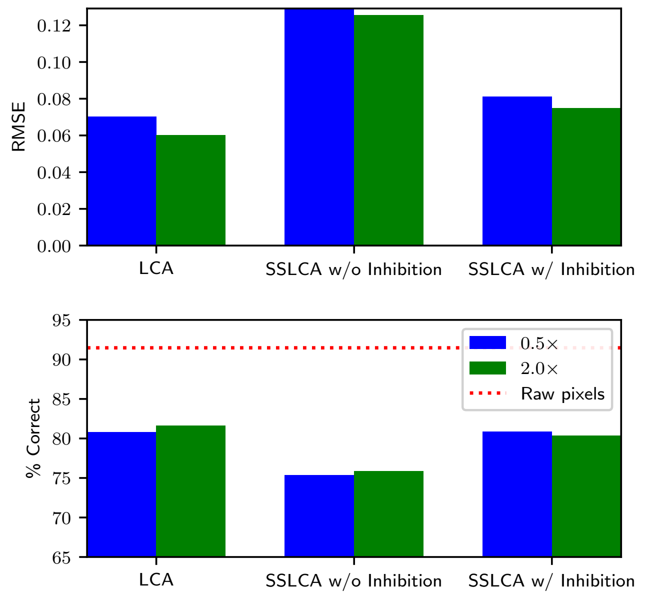


Fig. 13: Comparison of different algorithms and different completenesses at $S = 4$. The SSLCA is capable of matching the accuracy of the LCA when a deeper classifier is used.

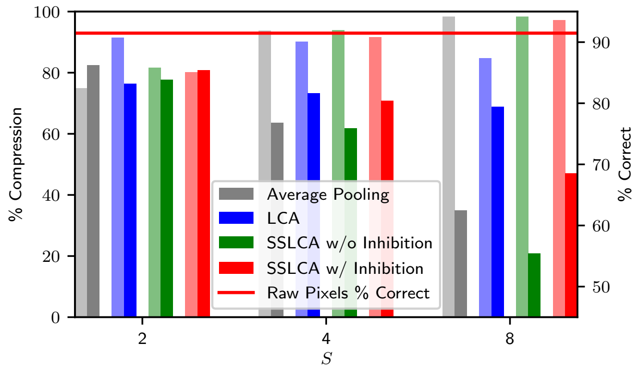


Fig. 14: Comparison of each algorithm at different encoding scales S. The left (lighter) of each pair of bars is the “% Compression;” the right (solid) bar of each pair is the “% Correct.” Inhibition always improved performance when using the deep classifier, without significantly affecting compression. The LCA does not change significantly due to a fixed λ threshold parameter; the SSLCA might achieve a similar effect by collecting more spikes, but this would slow the algorithm’s throughput.

Rather than 300 epochs with mini-batches of 64 samples, we used 150 epochs and mini-batches of 32 samples to train these networks. We trained using stochastic gradient descent, with an initial learning rate of 0.1; after 75 epochs this was reduced to 0.01, and after 112 epochs this was further reduced to 0.001. Simulations were done with keras; the DenseNet implementation can be found at https://github.com/titu1994/DenseNet, and keras can be found at https://github.com/fchollet/keras. Each accuracy measurement was the result of a single trial, so some stochasticity is embedded in the reported results. They are nonetheless internally consistent. Results are shown in Figs. 13 and 14.

On the raw CIFAR-10 data, these conditions produced a classification accuracy of 92 %. With the analog LCA $S = 4$, the DenseNet achieved a classification accuracy of 82 %, compressing the data down by 90 %. The SSLCA produced an accuracy of 80 % with 92 % compression. In the context of the deep classifier, we found a direct and inverse relationship between the LCA’s RMSE and the classifier’s accuracy (Fig. 13). Compression was calculated as $1 - \#\ active\ neurons \times (\log_2(\#\ neurons) + 4)$ divided by the number of bits in the image ($8 \times W \times H$). This represents the minimum number of bits to send a neuron index and its 4-bit spike count per active neuron.

For different values of S, the LCA maintained similar compression factors, due to the threshold λ being held constant with an increasing number of inputs, leading to more active outputs (Fig. 14). In contrast, the SSLCA’s sparsity comes from the number of spikes collected, which was fixed at 10 for all experiments. Thus, for larger patch sizes, more and more sparse representations were created, resulting in lower accuracy but higher compression. These parameters are all configurable and could be used to trade off between accuracy and compression, but these values were chosen as they produced roughly equivalent compression at $S = 4$. If the input dataset has greater covariance, then the accuracy loss would be lower for higher compression rates. Inhibition was always beneficial with a deeper classifier, and each increase in accuracy aligned with lower RMSE without exception. Figure 14 also includes compression factors and accuracies for downsampling the input image using average pooling across groups of $S \times S$ pixels. This demonstrates a baseline image compression technique against the sparse coding achieved by the LCA family. Generally, the LCA methods produced greater accuracies at comparable, high levels of compression than the average pooling method.

B. MNIST

The second dataset, MNIST, consisted of 70 000 28×28 grayscale images, each containing a single, centered, handwritten digit [14]. Again for faster simulations, this dataset was scaled down to 14×14. The test set contains an equal number of each digit class, so a simple accuracy was tabulated for each algorithm.

1) Accuracy: The SSLCA as defined up to this point performed notably worse on MNIST than the non-spiking LCA (Fig. 15). Unlike CIFAR, which has an average input value of 0.47, MNIST has an average input value of only 0.13. Since the SSLCA was designed deliberately to include a leak current through the crossbar, this lower input intensity could not sustain neuron charge reliably, leading to more random patterns of input events being encoded in each output spike. We found that applying a bias signal, by redefining the duty cycle of each input signal from $K_{max}k_{input}$ to $K_{max}(bias+(1-bias)k_{input})$, we could remedy this problem while preserving the power gains of the SSLCA architecture.

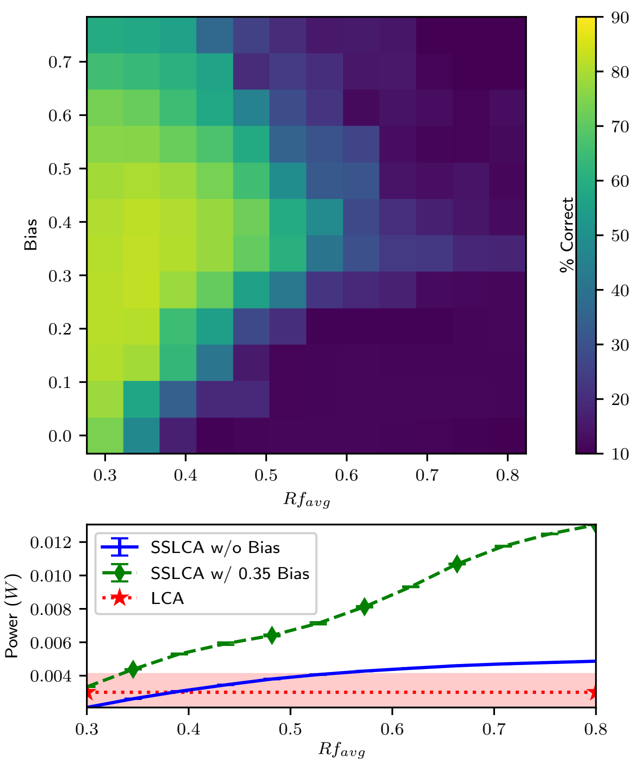


Fig. 15: Results of MNIST scaled to 14×14. Since MNIST has a much lower average input value than CIFAR, a bias needed to be applied to compensate for the additional current lost from the neurons back into the crossbar (Eq. (5)). Since a bias also increased the duty cycle of each input signal, more power was consumed.

For MNIST, we found that a bias of 0.35 boosted performance from $77\,\%$ correct classification up to $84\,\%$, versus a performance of $88\,\%$ by an optimal, analog LCA. The MNIST experiments' responses to write deviations were not found to be significantly different than the CIFAR experiments', shown in Fig. 10.

2) Power: The power savings on MNIST, even with the bias, were in-line with those found for CIFAR: $0.26\,\mathrm{pJ/input}$ for $100\,\mathrm{MOps/s}$. Note that the increased power savings compared to CIFAR (which consumed $1.77\,\mathrm{pJ/input}$) were due to the lower relative number of outputs to inputs: additional neurons are more expensive than additional input lines (partially due to the comparator, though mostly due to the crossbar). While the cost of additional inputs is different from the cost of additional columns, the SSLCA still demonstrates $\mathcal{O}(N)$ scaling in both dimensions.

As seen in Fig. 15, a non-spiking approach might consume less power on MNIST due to the low Rf_{avg} of the dataset. On the other hand, the power presented for the LCA does not include inhibitory logic, unlike the SSLCA: it would be difficult to include inhibition logic without closing the already-narrow margin.

V. Conclusion

Our work demonstrated that memristive devices with a low conductance ratio could be used for fast, low-power sparse coding, with in-situ learning, as long as their conductances could be set within $\pm 3\,\%$. This requirement matches the 4-bit resolution required by other neuromorphic architectures. We improved upon a previously published all-CMOS ASIC with $21\,\times$ the throughput using $99\,\%$ less energy per input. The quality of the resulting sparse codes were also evaluated with a state-of-the-art deep learning network, and were shown to reduce relative accuracy by only $2.4\,\%$ while compressing the input data by $92\,\%$. These figures could be adjusted for higher accuracy and lower compression. We showed that even datasets with low input activity, such as MNIST, could be properly represented through the use of a bias. The proposed SSLCA architecture was demonstrated to be very resistant to device variations, particularly when used with offline training. Sparse coding algorithms such as the SSLCA could be used to greatly reduce communication bandwidth between visual sensors and other processing algorithms, such as deep-learning networks. This architecture has applications in robotics and self-driving cars as well as surveillance and next-generation computers.

Acknowledgement

This work was supported by the National Science Foundation under award # 1028378 and by DARPA under award # HR0011-13-2-0015. The views expressed are those of the author(s) and do not reflect the official policy or position of the Department of Defense or the U.S. Government. Approved for public release, distribution is unlimited.

References

[1] A. Coates, A. Y. Ng, and H. Lee, "An analysis of single-layer networks in unsupervised feature learning," *International Conference on Artificial Intelligence and Statistics*, pp. 215–223, 2011.

[2] H. B. Ammar, K. Tuyls, M. E. Taylor, K. Driessens, and G. Weiss, "Reinforcement learning transfer via sparse coding," *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems*, no. Aamas, pp. 4–8, 2012.

[3] T. Masquelier and S. J. Thorpe, "Unsupervised learning of visual features through spike timing dependent plasticity," *PLoS Computational Biology*, vol. 3, no. 2, pp. 0247–0257, 2007.

[4] C. Zamarreño-Ramos, L. A. Camuñas-Mesa, J. A. Pérez-Carrasco, T. Masquelier, T. Serrano-Gotarredona, and B. Linares-Barranco, "On Spike-Timing-Dependent-Plasticity, Memristive Devices, and Building a Self-Learning Visual Cortex," *Frontiers in Neuroscience*, vol. 5, no. MAR, pp. 1–22, 2011.

[5] M. Zhu and C. J. Rozell, "Modeling Inhibitory Interneurons in Efficient Sensory Coding Models," *PLOS Computational Biology*, vol. 11, no. 7, p. e1004353, 2015.

[6] D. Querlioz, W. S. Zhao, P. Dollfus, J.-O. Klein, O. Bichler, and C. Gamrat, "Bioinspired networks with nanoscale memristive devices that combine the unsupervised and supervised learning approaches," in *Proceedings of the 2012 IEEE/ACM International Symposium on Nanoscale Architectures - NANOARCH '12*, pp. 203–210. New York, New York, USA: ACM Press, 2012.

[7] S. H. Jo, T. Chang, I. Ebong, B. B. Bhadviya, P. Mazumder, and W. Lu, "Nanoscale Memristor Device as Synapse in Neuromorphic Systems," *Nano Letters*, vol. 10, no. 4, pp. 1297–1301, 2010.

[8] W. Woods, J. Bürger, and C. Teuscher, "Synaptic Weight States in a Locally Competitive Algorithm for Neuromorphic Memristive Hardware," *IEEE Transactions on Nanotechnology*, vol. 14, no. 6, pp. 945–953, 2015.

[9] M. Payvand and L. Theogarajan, “Exploiting local connectivity of CMOL architecture for highly parallel orientation selective neuromorphic chips,” *Proceedings of the 2015 IEEE/ACM International Symposium on Nanoscale Architectures, NANOARCH 2015*, pp. 187–192, 2015.

[10] C. H. Bennett, D. Chabi, T. Cabaret, B. Jousselme, V. Derycke, D. Querlioz, and J. O. Klein, “Supervised learning with organic memristor devices and prospects for neural crossbar arrays,” *Proceedings of the 2015 IEEE/ACM International Symposium on Nanoscale Architectures, NANOARCH 2015*, pp. 181–186, 2015.

[11] B. A. Olshausen and C. J. Rozell, “Sparse codes from memristor grids,” *Nature Publishing Group*, vol. 12, no. 8, pp. 722–723, 2017.

[12] 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. 2526–63, 2008.

[13] E. Oja, “Simplified neuron model as a principal component analyzer,” *Journal of Mathematical Biology*, vol. 15, no. 3, pp. 267–273, 1982.

[14] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[15] A. Krizhevsky, “Learning Multiple Layers of Features from Tiny Images,” *... Science Department, University of Toronto, Tech. ...*, vol. 44, no. 8, pp. 1–60, 2009.

[16] S. Shapero, C. Rozell, and P. Hasler, “Configurable hardware integrate and fire neurons for sparse approximation,” *Neural Networks*, vol. 45, pp. 134–143, 2013.

[17] W. Woods, M. M. A. Taha, S. J. Dat Tran, J. Burger, and C. Teuscher, “Memristor panic - A survey of different device models in crossbar architectures,” *Proceedings of the 2015 IEEE/ACM International Symposium on Nanoscale Architectures, NANOARCH 2015*, pp. 106–111, 2015.

[18] W. Woods and C. Teuscher, “Approximate vector matrix multiplication implementations for neuromorphic applications using memristive crossbars,” in *Proceedings of the 2017 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, pp. 103–108. IEEE, 2017.

[19] S. Shapero, A. S. Charles, C. J. Rozell, and P. Hasler, “Low Power Sparse Approximation on Reconfigurable Analog Hardware,” *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 2, no. 3, pp. 530–541, 2012.

[20] T. Serrano-Gotarredona, T. Masquelier, T. Prodromakis, G. Indiveri, and B. Linares-Barranco, “STDP and STDP variations with memristors for spiking neuromorphic learning systems.” *Frontiers in neuroscience*, vol. 7, no. February, p. 2, 2013.

[21] W. Maass, “To Spike or Not to Spike: That Is the Question,” *Proceedings of the IEEE*, vol. 103, no. 12, pp. 2219–2224, 2015.

[22] S. Habenschuss, Z. Jonke, and W. Maass, “Stochastic Computations in Cortical Microcircuit Models,” *PLoS Computational Biology*, vol. 9, no. 11, p. e1003311, 2013.

[23] T. J. Hamilton, S. Afshar, A. van Schaik, and J. Tapson, “Stochastic Electronics: A Neuro-Inspired Design Paradigm for Integrated Circuits,” *Proceedings of the IEEE*, vol. 102, no. 5, pp. 843–859, 2014.

[24] J. Zylberberg, J. T. Murphy, and M. R. DeWeese, “A Sparse Coding Model with Synaptically Local Plasticity and Spiking Neurons Can Account for the Diverse Shapes of V1 Simple Cell Receptive Fields,” *PLoS Computational Biology*, vol. 7, no. 10, p. e1002250, 2011.

[25] P. Knag, J. K. Kim, T. Chen, and Z. Zhang, “A Sparse Coding Neural Network ASIC With On-Chip Learning for Feature Extraction and Encoding,” *IEEE Journal of Solid-State Circuits*, vol. 50, no. 4, pp. 1070–1079, 2015.

[26] T. Pfeil, T. C. Potjans, S. Schrader, W. Potjans, J. Schemmel, M. Diesmann, and K. Meier, “Is a 4-Bit Synaptic Weight Resolution Enough? Constraints on Enabling Spike-Timing Dependent Plasticity in Neuromorphic Hardware,” *Frontiers in Neuroscience*, vol. 6, no. JULY, pp. 1–19, 2012.

[27] 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 C50 C51,” no. 4, pp. 50–51, 2015.

[28] K. S. Burbank, “Mirrored STDP Implements Autoencoder Learning in a Network of Spiking Neurons,” *PLOS Computational Biology*, vol. 11, no. 12, p. e1004566, 2015.

[29] P. M. Sheridan, C. Du, and W. D. Lu, “Feature Extraction Using Memristor Networks,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 27, no. 11, pp. 2327–2336, 2016.

[30] P. M. Sheridan, F. Cai, C. Du, W. Ma, Z. Zhang, and W. D. Lu, “Sparse coding with memristor networks,” *Nature nanotechnology*, vol. 12, no. 8, pp. 784–790, 2017.

[31] E. Jones, T. Oliphant, P. Peterson *et al.*, “SciPy: Open source scientific tools for Python,” 2001–, [Online; accessed 2016-08-17].

[32] M. D. Zeiler, “ADADELTA: An Adaptive Learning Rate Method,” *arXiv*, p. 6, 2012.

[33] D. Querlioz, O. Bichler, P. Dollfus, and C. Gamrat, “Immunity to device variations in a spiking neural network with memristive nanodevices,” *IEEE Transactions on Nanotechnology*, vol. 12, no. 3, pp. 288–295, 2013.

[34] F. Alibart, L. Gao, B. D. Hoskins, and D. B. Strukov, “High precision tuning of state for memristive devices by adaptable variation-tolerant algorithm.” *Nanotechnology*, vol. 23, no. 7, p. 075201, 2012.

[35] Y. Xu, L. Belostotski, and J. W. Haslett, “Offset-corrected 5GHz CMOS dynamic comparator using bulk voltage trimming: Design and analysis,” in *2011 IEEE 9th International New Circuits and systems conference*, pp. 277–280. IEEE, 2011.

[36] W. Zhao and Y. Cao, “New Generation of Predictive Technology Model for Sub-45nm Design Exploration,” in *7th International Symposium on Quality Electronic Design (ISQED’06)*, pp. 585–590. IEEE, 2006.

[37] M. R. Driels and Y. S. Shin, *Determining the Number of Iterations for Monte Carlo Simulations of Weapon Effectiveness*. Naval Postgraduate School, 2004.

[38] R. Degraeve, A. Fantini, N. Raghavan, L. Goux, S. Clima, B. Govoreanu, A. Belmonte, D. Linten, and M. Jurczak, “Causes and consequences of the stochastic aspect of filamentary RRAM,” *Microelectronic Engineering*, vol. 147, pp. 171–175, 2015.

[39] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten, “Densely Connected Convolutional Networks,” *arXiv preprint*, pp. 1–12, 2016.