Synaptic Weight States in a Locally Competitive Algorithm for Neuromorphic Memristive Hardware

Walt Woods, Jens Bürger, and Christof Teuscher
Department of Electrical and Computer Engineering
Portland State University, Portland, OR, USA
Email: {wwoods, jb22, teuscher}@pdx.edu

***Abstract*—Memristors promise a means for high density neuromorphic nanoscale architectures that leverage in-situ learning algorithms. While traditional learning algorithms commonly assume analog values for synaptic weights, actual physical memristors may have a finite set of achievable states during online learning. In this paper we simulate a learning algorithm with limitations on both the resolution of its weights and the means of switching between them to explore how these properties affect classification performance. For our experiments we use the *Locally Competitive Algorithm* (LCA) by Rozell *et al.* in conjunction with the MNIST dataset and a set of natural images. We investigate the effects of both linear and non-linear distributions of weight states. Our results show that as long as the weights are distributed roughly close to linear, the algorithm is still effective for classifying digits, while reconstructing images benefits from non-linearity. Further, the resolution required from a device depends on its transition function between states; for transitions akin to round-to-nearest, synaptic weights should have around 16 possible states (4-bit resolution) to obtain optimal results. We find that lowering the threshold required to change states or adding stochasticity to the system can reduce that requirement down to 4 states (2-bit resolution). The outcomes of our research are relevant for building effective neuromorphic hardware with state-of-the art memristive devices.**

***Keywords*—memristive devices, neuromorphic computing, MNIST, LCA, image reconstruction**

I. Introduction

The discovery of memristors has led to many publications detailing their unique ability to function similarly to a biological synapse [1]–[3]. Recently proposed neuromorphic architectures employ memristive crossbars for storing synaptic weights and use *Spike-Timing-Dependent Plasticity* (STDP) to train those weights [1]–[4]. The architectures that we have seen do not discuss the issue of memristors only being tunable to an imprecise number of states. Analog control over these devices' internal states is not necessarily feasible as some devices display stochastic resistance to changes in their internal states [4], [5]. Online learning algorithms used for real-time video processing may not afford time to read and subsequently correct a memristor's internal state several times as proposed in [5]. In other terms, while learning the synaptic weights, we cannot assume high precision (or analog) weight tuning, but are likely to face devices that offer only a few bit resolution.

Rubin and Fusi have discussed synaptic state distributions in the context of memory retention [6]. Their comparison included bistable, multi-state, and cascaded synapses. It was shown that cascaded synapses, with state-dependent degrees of plasticity, improve memory retention and robustness toward noise. With different memristive device types showing different state distributions and switching probabilities, Rubin's results become very relevant to memristor-based neuromorphic architectures.

In [5] a tuning algorithm for memristive devices was shown that allows to adjust the memristive state within a 1% precision. The presented algorithm targets applications that require precise tuning of memristors, but does not fit well to online learning architectures as the tuning depends on a feedback loop and successive application of voltage pulses that vary in amplitude and/or width. STDP, as used for memristive weight training, does not provide such a feedback loop with precise state tuning. It assumes a fixed amplitude and a range of pulse width that depends on the overlap of the forward and backward pulses. Over many of these pulse interactions, some level of fine-tuning occurs, but not to the extent induced by the mentioned algorithm.

In this paper we model memristive characteristics that can influence overall classification or reconstruction performance. We use the algorithm presented by Rozell *et al.* [7] to show how many states are required to achieve a certain performance. We also look into how the distribution of weight states and the behaviors for switching between them affects learning. To investigate this topic with adequate breadth, we present the algorithm with two distinct tasks: classifying digits from the MNIST dataset, and reconstructing image patches from a small set of whitened natural images. We expand upon our prior work [8] by adding the θ family of state distributions and a new dataset of natural images, investigating the effects of synaptic state switching with both a near-binary dataset, MNIST, and a dataset with many intermediate values.

II. Memristive Switching Properties

As was briefly introduced, the importance of this work stems from the physical limitations when modifying the state of memristive devices, but that are rarely modeled in presented neuromorphic architectures based on those devices. In this section we will discuss several aspects of memristive switching that can have an impact on the learning ability of a neuromorphic design. The modeling of switching properties throughout this paper are simplified abstractions of the experimental data reported on memristive devices. This is done as we aim to provide general conclusions for a wide range of memristive devices, rather than focusing on a specific one.

In [5] switching characteristics for a titanium-dioxide memristor were presented. The state change of the presented memristor is a function of the applied pulse amplitude and the pulse width. The choice of pulse amplitude and width determines the granularity with which a device state can be altered. One implication of this is that the distribution of memristive states is not linear. For this particular device the data implies a denser weight distribution in its high resistive range. A second implication is that, depending on the applied pulse width, not every pulse will actually trigger a state update.

With regard to the weight distribution, the memristive device presented in [9] exhibits a higher density of states

in its low-resistive range. The general conclusion we derive, independent of any specific device, is that one can face various types of weight distributions and the choice of either memristor type has to consider the impact of the state distribution on the chosen architecture, algorithm, and application.

Similar to the device presented in [5], the issue of stochastic switching is discussed in detail in [4]. It is stated that the switching probability depends on the applied voltage amplitude and pulse width. The authors provide a detailed description of the switching probability distribution, and while individual switching events cannot be controlled, the switching distribution can be controlled well. The effects of stochastic switching on learning performance is another subject that will be studied in Section IV-B.

III. Model

The network structure and learning algorithm used in this work is the *Locally Competitive Algorithm* (LCA) introduced by Rozell *et al.* in 2008 [7]. Motivated by the sparse neuronal activity found in biological brains, the LCA employs a sparse coding principle to minimize a given cost function. Sparseness within the algorithm is achieved by modeling local inhibitory connections across neurons. This locality well resembles biological brains and makes it a suitable algorithm for hardware implementations. The LCA that we use in our experiments is an unsupervised learning approach that calculates an output neuron's excitation by integrating an *ordinary differential equation* (ODE) as shown in [7]:

$$\dot{u}_m(t) = \frac{1}{\tau}\left[b_m(t) - u_m(t) - \textstyle\sum_{n\neq m} G_{m,n}a_n(t)\right], \qquad (1)$$

where u_m is the m^{th} output neuron's membrane potential, b_m represents a neuron's excitatory inputs and is equal to the dot-product of the neuron's input vector $\vec{s}$ and the corresponding weight vector $\vec{w}_m$. With the LCA employing leaky-integrator neurons, $-u_m$ models the leak rate as a function of time and the current membrane potential. The above-mentioned sparse activity within the network is achieved by inhibitory connections across all output neurons. $G_{m,n} = w_m^t \times w_n$ describes the similarity of neurons m and n's receptive fields. The inhibition that a neuron m experiences is the sum over the receptive field correlation terms multiplied with the corresponding neuron output values. In other words, if neurons m and n have similar receptive fields, neuron m will have strong inhibition of its membrane potential when neuron n is active. To enforce sparseness and only apply inhibition amongst active neurons, a thresholding function is applied to the membrane potential to yield a_n, which describes neuron n's output signal.

In hardware, this ODE would be implemented directly, with the time for the system to reach a stable state proportional to the τ parameter corresponding to the physical system. For our software simulations, the ODE is implemented purely mathematically, with its integral calculated by evaluating the derivative and summing 21 times, with a dynamic integration width for each step. Each step thus corresponds to an amount of physical time proportional to some constant times the width of the step. We minimize the difference between a real-time hardware implementation of the ODE and our step-based software simulation by calculating the width of each step dynamically as $\alpha/\|\dot{u}\|$, where α is a constant used to balance the total time simulated against the number of steps required. In our simulations, this allowed us to reduce the number of steps for integration from 1,000 down to 21, yielding a 50x speedup.

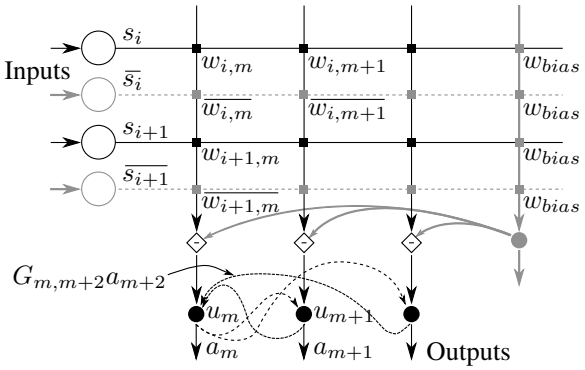


Fig. 1: Example LCA network displaying relation of input currents $\vec{s}$, receptive field weights $W = (\vec{w}_0, \vec{w}_1, ...)$, internal states $\vec{u}$, thresholded outputs $\vec{a}$, and inhibitory connections of strength $G_{m,n}a_n, m \neq n$. The field weights $w_{i,m}$ are realized as memristive devices at the crossbar junctions. Illustrated in solid gray is the bias column, populated with memristive devices set to weight w_{bias}. The bias column corrects for leak current that would otherwise skew the calculated dot products. For systems needing to calculate dot products between inputs and weights both spanning positive and negative values, such as our reconstruction task, three modifications are required: w_{bias} is set somewhere between 0 and 1, such as 0.4; the illustrated dotted gray rows must be added with complementary weights ($w = 0.7 \implies \overline{w} = 2 * w_{bias} - w = 0.1$); and only one of s_i or $\overline{s_i}$ must be set to a positive voltage dependent on the sign of the original input. These three modifications modify the dot product calculation to account for negative and positive weights and inputs.

As a modification to Rozell *et al.*'s original ODE, we additionally apply a small homeostasis term by dampening $b_m(t)$ by a multiplier μ_t, which is decreased towards a minimum of 0.4 according to each neuron's activation after a training pulse. It then slowly charges back up to 1.0. Homeostasis counter-acts inhibition early in the learning process, preventing a neuron from learning an average representation of all digits. Our natural image patches are evaluated similarly, except homeostasis is disabled as there is sufficient diversity in input patterns to render it unnecessary: the multiplier μ_t is always 1.0. After integration, we apply a training pulse to update weights according to the formula derived in [10]:

$$\dot{\vec{w}}_m = (\vec{s} - \vec{a}\phi^t)(a_m * \alpha), \qquad (2)$$

where α is a parameter to adjust the learning rate, and weights are clamped post-update to $[0, 1]$. We use an initial α of 0.811 for the MNIST task and 0.611 for image reconstruction. These were optimized for classification accuracy and reconstruction error, respectively. Even though these α values seem large, as the LCA learns and reconstruction error is minimized, the corresponding residual also decreases in magnitude

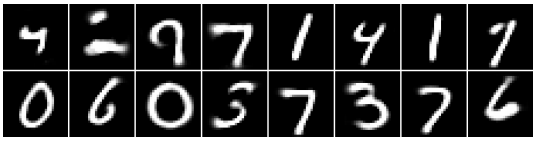


Fig. 2: Subset of learned 28×28 receptive fields in a matrix with 50 output neurons, with white representing maximally conductive devices.

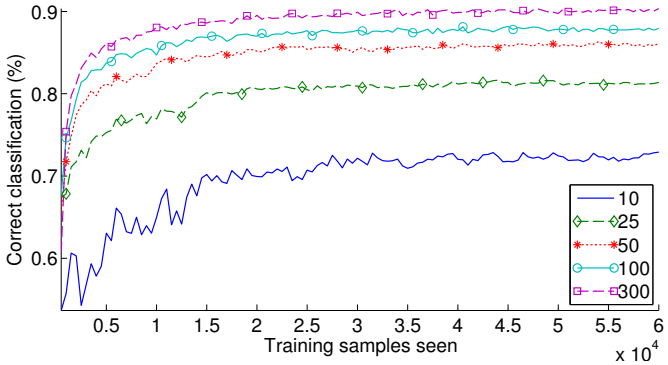


Fig. 3: Performance on MNIST test set with analog weights throughout training for different output neuron counts. For each number of neurons, our classification accuracies are comparable to or better than those presented for the STDP algorithm in [1], allowing that our networks trained across only one iteration of the training set instead of three. Improved performance with the same number of output neurons results from inhibition in LCA vs the lack of inhibition in STDP.

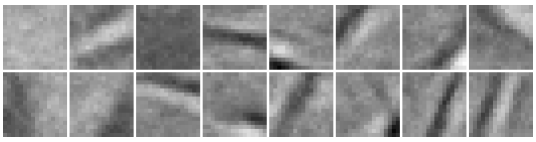


Fig. 4: Learned 16×16 receptive fields for the reconstruction task. Note that these weights represent both positive and negative values, a requirement for gabor filters.

leading to smaller updates each training cycle. Additionally, after every weight update, we multiply α by a constant such that every 1,000 updates, $\alpha_{t+1000} = \alpha_t * 0.92$. This helps our system to converge, as it prevents us from repeatedly stepping between opposite sides of an optimal solution. The reconstruction task benefits less from this repeated downstepping; there, we use a constant such that $\alpha_{t+1000} = \alpha_t * 0.99$.

The net effect of integrating Rozell's ODE under these conditions and adjusting the weights via equation 2 is that neurons respond to their receptive field while simultaneously inhibiting neurons responding to the same stimuli, encouraging the various output neurons to diversify a bit. This diversity may be seen in the receptive fields of a trained network's output neurons (see Fig. 2). As can be seen, some receptive fields do not learn complete digits, but just parts of them.

For classification tasks such as MNIST, one may interpret the LCA's purpose as feature extraction via sparse encoding. To translate from the sparse encoding, we use a supervised *single layer perceptron network* (SLP) trained via back-propagation in order to infer a digit classification from the output neurons from the unsupervised layer. The SLP works only to determine which neurons from the LCA correspond to which digits; if there are 50 output neurons in the LCA, then the SLP will consist of $50 \times 10 = 500$ weights: one set for each digit classification. As the SLP performs no calculation other than interpreting the active set of LCA features, our experiments do not modify the resolution of the SLP's weights. These are always stored as analog values in our simulations, unlike the weights used within the LCA. We then apply images from the MNIST database [11] as inputs into our system, training the supervised layer to learn the labels of each digit based on the activations of the output neurons from the unsupervised layer. Our baseline classification percentages with traditional analog weights can be seen in Fig. 3.

All of our MNIST networks used to investigate weight states consist of 784 input rows and 50 output neurons. They are trained with one round of the full 60,000 MNIST training set, then tested on the 10,000 MNIST test set. The entire experiment of training and subsequent testing is repeated either 7 or 12 times depending on if the estimated error of the sample's mean is less than 5% after 7 trials, as per the method outlined in [12]. The initial weights used in our networks are uniformly distributed along $[0, 1]$.

For our reconstruction task, we train networks with 256 input rows and 50 output neurons on 114,921 grayscale patches of 16x16 pixels each from various whitened natural images. We measure the average *root-mean-square error* (RMSE) on a separate set of 12,769 patches by calculating the average squared difference for each number in the residual: $\frac{\|\vec{s}-W\vec{a}\|}{16}$. After whitening, which intrinsically sets the mean of the image data to 0, our pixel values are scaled to $[-0.5, 0.5]$ for an overall domain of 1 so that the RMSE is a number roughly correlating to an average pixel's percent error from the reconstruction to the original. If our network is given a perfectly black image and reconstructs it as a completely white image, the RMSE will be 1.0. If a perfect gray image is reconstructed as white, then the RMSE will be 0.5. Note that we use a scale with 0 centered so that the learned patches are nice gabor-esque features that correlate with contrast edges, as seen in other reconstruction tasks [10], [13], [14]. These can be seen in Fig. 4.

Since the input values for image reconstruction include negative values, the weights represented by our LCA for this task must span negative numbers rather than the strictly positive $[0, 1]$ interval that we used for MNIST. The output for each neuron, a_m, is still strictly non-negative, as dictated by the thresholding function. In hardware, this arrangement would be realized by having a crossbar with two rows for each input (Fig. 1). If the input value is positive, then the driver for one row is set to a positive voltage and the other row's driver is set as high impedance. If the input value is negative, then the other row is driven by a positive voltage, and the first row is given high impedance. The memristive devices connecting each row to each column have complementary memristive conductances. Each crossbar would additionally have a bias column, with the memristive conductance fixed to a value that represents the device's zero weight. By subtracting the current through a fixed-weight device from the current generated by the learned weight, a single memristor's state represents both negative and

positive weights. The usage of complementary weights allows representation for both positive and negative inputs. Note that depending on the zero point set by the fixed-weight devices, the weight scale may be asymmetric.

For our software simulations, we use two simulated memristive devices with a weight range of $[0, 1]$ for each $w_{i,m}$: one that is used for a positive input polarity, and the complementary device that is used for a negative input. w_{bias} is set differently for each experiment, and is addressed in section V-A. Each patch is evaluated by assembling a vector composed of the weights representing polarities that match the input's polarity, and calculating $b_m = \sum_i 2|s_i|(w_{i,m} - w_{bias})$. To update each weight, a target value is calculated as $w_{target} = w_{i,m} + \dot{w_{i,m}}$, where $w_{i,m}$ is the weight matching the input sample's polarity, and the derivative $\dot{w_{i,m}}$ is calculated as in equation 2. The weight representing the same polarity as the input receives a standard training pulse, while the weight representing the opposite polarity receives an update pulse of magnitude $2w_{bias} - w_{target} - \overline{w_{i,m}}$, where $\overline{w_{i,m}}$ is the complementary weight (as in Fig. 1). We skip this update calculation for columns whose output a_m is very close to zero, which implies a $\dot{w_{i,m}}$ close to zero.

To compare our approach with other image reconstruction publications, multiply the reported RMSE by a scalar that normalizes the given pixel space to a domain of width 1, and be aware how completeness affects reconstruction fidelity (Kim *et al.* has a number of plots demonstrating this [15]). As an example for the scalar, since Kim *et al.* scale their pixels from $[-2, 2]$, the reported RMSE must be multiplied by $\frac{1}{4}$ to make a valid comparison. Our analog reconstruction results are comparable to others in the field: with only 50 output neurons on 16x16 image patches, or with only 19% completeness, the LCA algorithm as implemented for this paper reconstructs natural images with a 10% RMSE (Fig. 16). The original results presented by Rozell *et al.* in their 2008 paper presents about a 16.7% RMSE with a network with sufficient neurons for 400% completeness, but their network uses a static dictionary and not one specifically generated to minimize RMSE [7]. Zylberberg *et al.*'s SAILnet algorithm, which is comparable to LCA in that it demonstrates inhibition, is explored by Kim *et al.* with a network exhibiting 100% completeness and reports an RMSE of 4.9% [14], [15].

IV. MNIST Results

A. Limited State Model

The first experiment we will discuss is motivated by the inability for infinitely small pulse widths used for weight updates in online learning situations. In [5] a tuning algorithm was able to adjust memristive states to a roughly 7-bit resolution. However, without feedback loops it is reasonable to expect a lower state resolution for individual updates.

Our limited state model is similar to a step function: there is a finite set of obtainable values w'_u for u of $[0, k-1]$ for some k number of states. For instance, if $k = 3$, then the uniform distribution would yield 0, 0.5, and 1 as potential states for weights post-update. This is a simplified approach for modeling coarse-grained state updates, which can be a result of too large pulse widths or highly non-linear (e.g., binary) state switching characteristics. As was discussed in section II, memristive devices show different state distributions which can be very roughly approximated by a power law distribution.

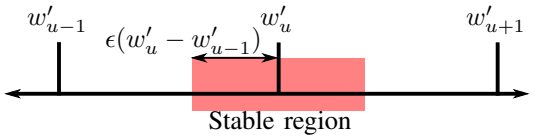


Fig. 5: Illustration of state switching considering a threshold region of stimulus where the weight's value does not ultimately change. If a weight would move outside of the stable region, it will be advanced to the adjacent state.

Introducing a parameter ω describing the power law, we use Eq. 3 to describe the state distributions:

$$\begin{aligned} w_u &= \tfrac{u}{k-1} \text{ for } u \in [0, k-1], \\ w'_u &= w_u^{\omega}. \end{aligned} \tag{3}$$

A special case of the power law is the linear distribution, where $\omega = 1$. It is worth noting that $\omega > 1$ implies a higher resolution in the lower weights, whereas $\omega < 1$ implies a higher resolution in larger weights.

In addition to these basic power law distributions, which lead to a high state density at either the low end or the high end, but not both, we also investigate distributions created by stacking two of these next to each other with multiplicatively inverted powers. This leads to distributions with good representation at either the extremes or the medial region. θ is used to denote this type of distribution and its associated parameter. That is, a value of $\theta > 1$ corresponds to a distribution with a high density at either extreme, while $\theta > 1$ describes a distribution with increased density in the medial weight region.

To fully specify state transitions based on a limited number of states, we need to make an assumption about switching between states. We define a threshold for the minimum weight delta calculated in Eq. 2 in order for the weight to jump to the next state. This threshold is defined as $\epsilon(w'_u - w'_{u-1})$, illustrated in Fig. 5; for now we assume a round-to-nearest approach ($\epsilon = 0.5$) where the final state is set to the stable state closest to the adjusted weight as calculated by Eq. 2.

We will start by analyzing the linear case, $\omega = 1$. Fig. 6 shows that performance is poor until a specific threshold: at $k = 7$ states, the mean performance is 25%. At $k = 8$, performance jumps to a much improved 70%. To see why this happens with round-to-nearest, we will take a look at the weight distributions of the trained networks with varying numbers of states, in Fig. 8. Looking at the "Analog" data set, we can see that the optimal weight distribution has significantly more weights close to 0 than 1, but also has a significant number of high values. Very few finalized weights end up in the middle range. This makes sense with the MNIST data set, which is largely pure-white digits on a black background. With round-to-nearest behavior, $\epsilon = 0.5$, the effects of initializing the weights with a uniform distribution can clearly be seen in, e.g., the case of $k = 4$. Since we are using round-to-nearest, and the states at 0 and 1 are special in that they are approached from only one side rather than two sides, we end up with half as many weights initially ending up in states 0 and 1 as in 0.33 or 0.67. Since the final distribution is identical to this initial distribution, we can conclude that

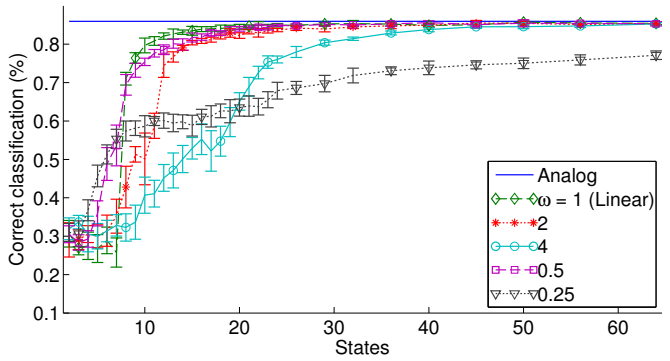


Fig. 6: MNIST performance given varying ω and a finite number of states k; state changes are performed by round-to-nearest, $\epsilon = 0.5$. For near-linear cases, we can see that around 16 states are required for performance to surpass 82% when a round-to-nearest model is used.

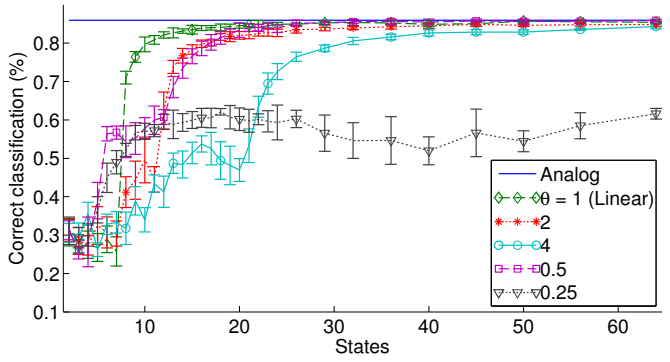


Fig. 7: MNIST performance with θ distributions. While slightly worse at classifying digits than when a simpler ω distribution is used, convergence with linear performance happens with only a few more states, around $k = 20$. Decreased performance is due to an inability to pass through the medial region when $\theta > 1$, and an inability to reach the ideal solution (more weights in the extremes) when $\theta < 1$.

almost all of the weight deltas generated by our algorithm are insufficient to shift a weight into a neighboring state. This might be addressed by either increasing the width of the training pulse, effectively modifying the α parameter in equation 2, or by decreasing ϵ, which would require different memristive switching characteristics. A lower ϵ value is desired in practice as it translates to lower energy requirements for updating the device. However, for training purposes, increasing α or decreasing ϵ are identical.

Networks with $\omega \neq 1$ have lower thresholds in their highly represented regions due to the tightened proximity of states at one end of the spectrum or the other. Lower thresholds mean that smaller training pulses effect a weight change. Even a slight variation in output weights can translate to a large difference in classification as it enables the supervised SLP to differentiate the varying receptive levels. However, the benefit is small, indicating that optimal performance with round-to-nearest behavior requires around 16 states for near-linear distributions. The further the weight distribution gets

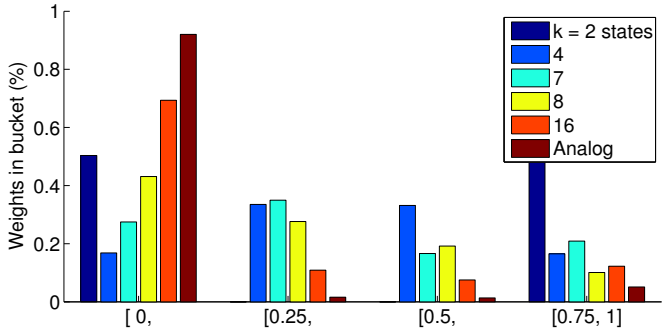
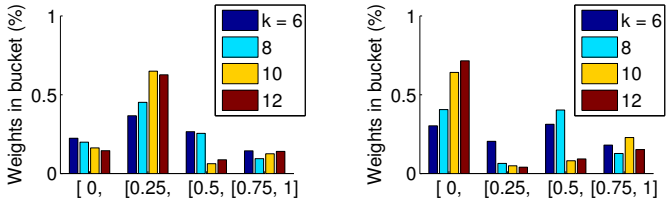


Fig. 8: Comparative learned weight distributions for $\omega = 1$ linear distribution of weights and varying n states and $\epsilon = 0.5$. We show that the main issue with a large ϵ is the difficulty for the system to overcome each stable state's inertia. Most weights are adjusted within the stable region, keeping them in their initial state.



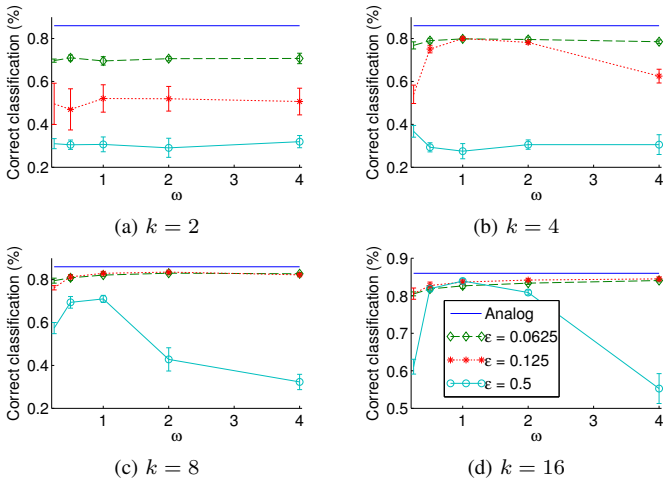
(a) $\omega = 0.5$ (b) $\omega = 2.0$

Fig. 9: Comparative learned weight distributions for (a) $\omega = 0.5$ and (b) $\omega = 2.0$. We can see that these distributions have issues moving between states in their low-density regions when ϵ is large (0.5).

from linear, the more states required in order to reach the same asymptotic performance of a linear distribution.

To overcome the algorithm's inability to move a weight between states when there is a large threshold, one might either increase the strength of the training stimulus (to overcome the large threshold) or lower the threshold (this would require different device characteristics). The results of varied ϵ can be seen in Fig. 10. Between the binary case of $k = 2$ and the next step of $k = 4$, we can see that the case of $\epsilon = 0.125$ catches up to $\epsilon = 0.0625$. This trend continues as k increases; a lower threshold promises high initial results, but becomes limiting as the representable resolution increases. The effect trickles out with a more gradual effect into non-linear states. It is easiest to see with $\epsilon = 0.5$: the linear case converges to optimal performance first, followed by $\omega = 0.5$. This confirms that the largest benefit for a linear representation with a thresholded algorithm is the uniformity of the gap between states. The constant gap allows a thresholded algorithm to traverse up to 1 (full conductance) as easily as it can work down to 0 (low conductance). While using a lower threshold allows the algorithm to compensate somewhat, performance above 80% is not achieved until minimally $k = 4$ states.

We also investigate the special case of $\epsilon = 0$, which we term "always switch." This case would be relevant for memristive devices with no threshold behavior or when applying voltage pulses that always cross the switching threshold.



(a) $k = 2$ (b) $k = 4$ (c) $k = 8$ (d) $k = 16$

Fig. 10: Change in classification performance for various values of ϵ as k is increased. When k is low, it is beneficial to have a lower ϵ. However, as k increases, rounding becomes more expressive and it becomes beneficial to have a higher ϵ. As discussed earlier in Section IV-A, ϵ and α are interchangeable, inversely proportional variables.

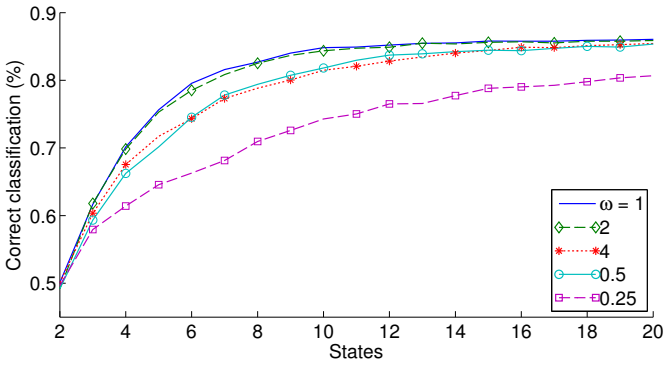


Fig. 11: Linear stochastic switching between weight states. A linear weight distribution ($\omega = 1$) is most capable of learning the MNIST digits, as it was with thresholding, but values of ω around 1 exhibit similar performance.

Networks exhibiting "always switch" are explored in Figs. 12, 13, and 14. These networks perform significantly worse than round-to-nearest or networks exhibiting a low ϵ. The reasoning for this is illustrated in Fig. 14. Since a tiny update results in an entire state change, networks with "always switch" tend to generate receptive fields that oscillate between two or more digits. Since each subsequent update makes the output neuron represent a different digit, the SLP that we use to classify our output neurons can not keep up and labels the digits poorly. This could be remedied by training the SLP on frozen output neurons, but that increases the required training time and is not suitable for online learning.

B. Stochastic State Switching

In addition to thresholded switching, we have taken the first step towards exploring stochastic switching. Stochastic switching was discussed in [4] and is closely related to cycle-to-cycle variation of memristive devices. While [4] addresses stochasticity in a strictly binary capacity, the effect has a non-binary component visible in Figs. 4 and 5 of [4] and is also prominent in the analog device behavior presented in [5], where near-identical pulses produce drastically different changes in device state. We investigate linear switching, which is, on average, ideal for a set of weights. That is, if a weight update would move by 30% toward the next state, then the probability of the weight changing to that next state is set to 0.3. For the same case, with a probability of 0.7, the weight will remain unchanged. This switching model compensates for the lack in state resolution by storing additional information probabilistically. Overall, the average weight for a given subset of weights should closely approximate the desired average weight. Looking at Fig. 11, we can see that ω near 1 converges to optimal performance around $k = 10$ states with a slight bias towards $\omega > 1$. For comparison with our prior thresholded transitions, we show a close-up of this in Fig. 12. While networks exhibiting linearly stochastic state transitions outperform round-to-nearest, they perform markedly worse than transitions utilizing a low ϵ (high state density or weak non-linear switching characteristics).

To see why stochasticity performs worse than thresholding with low ϵ, we investigated both the weight histograms in Fig. 13 as well as the generated receptive fields in Figs. 15. We show the weight histogram to demonstrate that the algorithm avoids the issue experienced in round-to-nearest threshold, where partial updates are lost since they are smaller than the threshold. The weight distributions for both $\epsilon = 0$ and linear stochasticity are much closer to the analog weight distribution than the distribution generated by the round-to-nearest network. However, the generated receptive fields illustrate the issue: since our LCA uses inhibitory forces to discourage two output neurons from being simultaneously active to represent the same input, the stochastic network learns non-continuous parts of a digit, and then begins inhibiting other neurons from learning to represent those pixels. While stochasticity would guarantee that the average for a given set of weights will approach the ideal weight, the effect is that inhibition favors early learning and "blurs" the resulting receptive fields. Since these receptive fields react to a wider spread of pixels than they do in the analog network, the corresponding output neurons respond to more digits, preventing the SLP from being able to adequately label input digits.

V. Reconstruction Results

A. Limited State Model

In Section III, we established that the reconstruction task needs each memristor's logical $[0, 1]$ range to be partitioned into negative and positive weights by a point representing zero, w_{bias}. While w_{bias} could be set to the device state closest to 0.5, achieving a symmetric weight range, we find that for ω distributions, using the device state closest to 0.5^{ω} produces much better results, as this generates a w_{bias} with equal numbers of device states above and below the zero point. For θ distributions, we take the device state closest to 0.5, for the same reason.

Observing the quality of image reconstruction under a limited state restriction in Fig. 16, it can be seen that greater resolution in the input signal does not significantly change the algorithmic requirements for the linear case. The algorithm

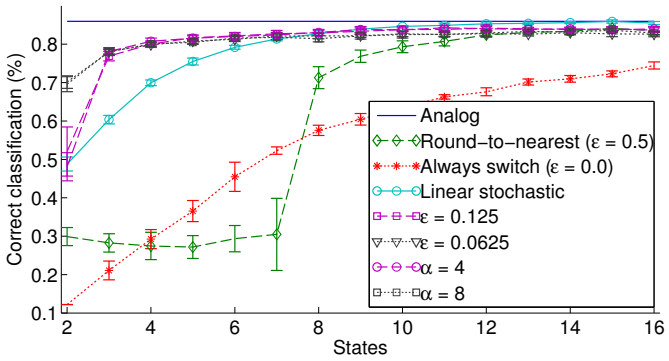


Fig. 12: Switching comparison with $\omega = 1$. Linear stochasticity performs worse when $k < 8$ than thresholded switching with a low ϵ, which is explained in Fig. 15. Bringing ϵ to its limit of 0 also yields undesirable results. The "always switch" line explores the issues with setting ϵ too low, which are illustrated in Fig. 14. Lines with higher α are also shown (with $\epsilon = 0.5$), to demonstrate the equivalence relationship between higher α and lower ϵ. Note that all graphs have an asymptotic value identical to the analog line. As the distance between states becomes less and less, all of these methods converge to a real-valued approximation.

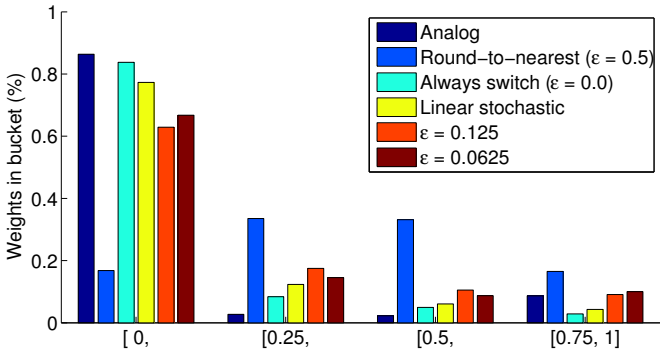


Fig. 13: Weight distributions for $k = 4$ states with $\omega = 1$ linearly distributed states across various state switching schemes. From Fig. 12, we know that "always switch" and round-to-nearest both exhibit poor classification performance. From the weight distributions alone, the reason for this is impossible to determine: "always switch" avoids the issue of losing partial updates. Linear stochastic transitions also perform worse than expected. Figs. 14 and 15 explain why these switching schemes exhibit suboptimal performance. Some stability (small, non-zero ϵ) helps to alleviate these issues.

converges with a minimum of $k = 20$ states, only a few more than the linear case for MNIST. This slight increase fits with the slight decrease in standard deviation for this dataset compared to the polarized MNIST dataset. In other words, we expect shorter training pulses to be generated as the residual will, on average, be smaller. However, the core issue is the same, where the maximum gap between two states determines the ability of the LCA to learn features from the input data. Once update pulses reach sufficient width to trigger changes in the device's internal state, the algorithm has little issue

Fig. 14: Sample set of learned receptive fields with $k = 4$ for "Always switch" thresholding ($\epsilon = 0$). The performance for this network is abysmal (30%), not due to an inability to represent individual digits, but rather because many output neurons oscillate between representing several different digits. Since ϵ is so low that any potential change leads to a state change, the digits do switch, confusing the supervised SLP layer on top. Here, the first receptive field could easy switch between an 8, 5, and 3. This is in accordance with [6] where it was shown that cascaded synapses with different switching thresholds are required to achieve good memory retention.

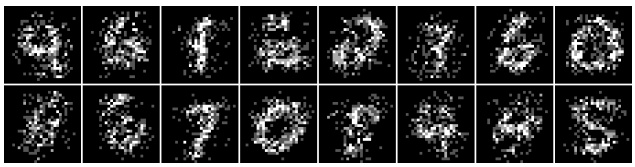


Fig. 15: Sample set of learned receptive fields for $k = 4$ with "Linear stochastic" state transitions. The spread in this network indicates that inhibitory forces prevent it from converging to a more representative solution. Since only a subset of the weights that could change state actually effect a state change, those weights that do change state inhibit themselves changing in other output neurons during subsequent training.

converging to a good approximation of the analog solution using the available device states.

For ω distributions, any non-linear configuration will create a high-density region of device states around w_{bias}, lowering the required number of states to achieve a given level of performance (Fig. 18). This adds a little insight to our prior findings from Section IV-A: since the final analog solution for reconstruction only uses weights around w_{bias}, this is the only region in which the size of the gap between states is of particular importance. The only exception to this is value initialization - as discussed in Section IV-A, weights have a high probability of being initially assigned to states with large gaps between them in our experiments. This can be seen in the results for θ distributions in Fig. 17. Since θ distributions have a symmetric distribution of states around 0.5, it is expected that w_{bias} is close to 0.5. $\theta < 1$ has a high state density in the right region, but too many weights are stuck at the extremes during initialization: compare Fig. 17 to stochastic switching in Fig. 21.

Experiments showing the results of varying ϵ on the reconstruction task can be seen in Fig. 19. A low number of states k is complemented by a small value of ϵ, exactly the same as with the MNIST dataset. Using memristive hardware or update voltages consistent with a lower value of ϵ helps to prevent weight stagnation, allowing the algorithm to continue learning.

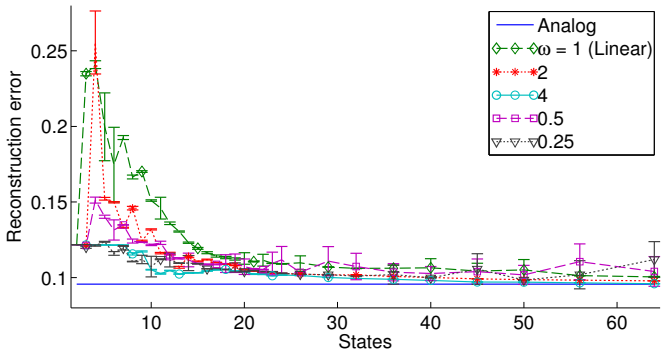


Fig. 16: Quality of reconstruction for varied ω and a finite number of states k; state changes are performed by round-to-nearest, $\epsilon = 0.5$. For the linear case, we can see that performance approaches that of the analog algorithm when $k \geq 20$. Distributions far away from linear display an improvement for reconstruction, converging as early as $k \geq 10$. This is because of our choice of w_{bias}, which makes logical zero a memristive state with neighbors closer than they would be in the linear case. This is further explored in Fig. 18.

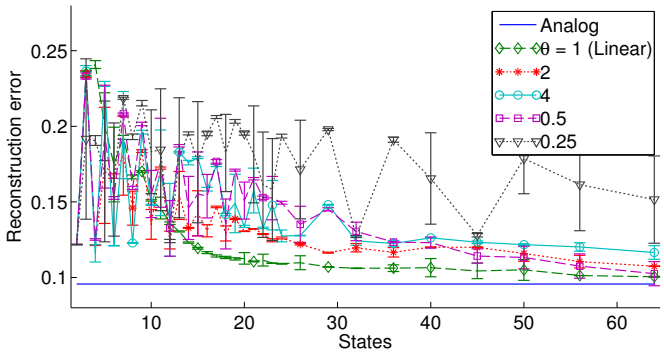


Fig. 17: Reconstruction performance for θ distributions with round-to-nearest switching. As discussed in Fig. 18, the ideal solution has a greater density of weights around w_{bias}, which for θ distributions is always near 0.5. Even though $\theta < 1$ is expected to outperform the linear case due to better representation of the ideal result range, it does not since so many weights are stuck at the extremes during initialization. $\theta = 0.25$ is particularly susceptible to this effect; stochastic switching, Fig. 21, or reducing ϵ may be used to resolve the issue. The oscillation for low values of k between high and low performance is explored in Fig. 21.

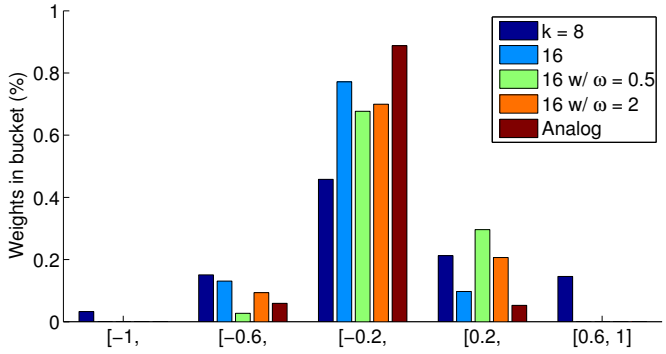
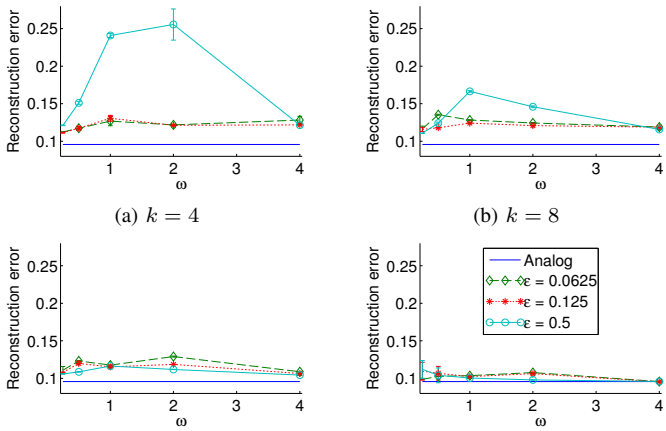


Fig. 18: Weight distributions for various states and ω distributions, with w_{bias} mapped to 0 and scaled by 2. Non-linearity performs significantly better due to the mass of values around w_{bias}. Whereas the MNIST results consist of weights primarily at the extremes of possible values, the focus on medial values in a reconstruction task means that medial values must be well-represented. Distributions further away from linear create a region where smaller updates will not be discarded as they were for MNIST (Fig. 8). Placing w_{bias} in this high-density region results in improved performance.



(a) $k = 4$ (b) $k = 8$

(c) $k = 16$ (d) $k = 64$

Fig. 19: Similarly to the results from the MNIST experiments from Fig. 10, a low value of ϵ (or a high training α) allows the algorithm to enact more successful weight updates. This is beneficial for low values of k or for escaping the high-inertia states introduced by $\omega \neq 1$. As k increases, this benefit is reduced.

The overall algorithm functions very similarly with an analog dataset as it does with a highly polarized one, such as MNIST. However, achieving near-analog weight performance on a reconstruction task requires a few additional states and a device with switching distribution that is favorable to representing weights in the selected w_{bias} region.

B. Stochastic State Switching

As shown in Fig. 20, stochastic performance in an analog dataset shares the most important property found with the MNIST dataset (Section IV-B). Overcoming large gaps created by non-linearity in state distribution is not an issue as it is with a rounding model. This greatly helps for $\theta < 1$. All other distributions perform similarly to their corresponding experiments with ϵ-based rounding models.

VI. Conclusion

In this paper, we investigated the effect of a limited number of stable states on a hardware-friendly LCA, which might be realized in a memristor network. While previous work has simulated learning algorithms using analog storage for the weights, we have studied a learning algorithm in the contexts of different state transitions that different types of memristors possess. We have also proposed an architecture suitable for memristive networks needing to evaluate both

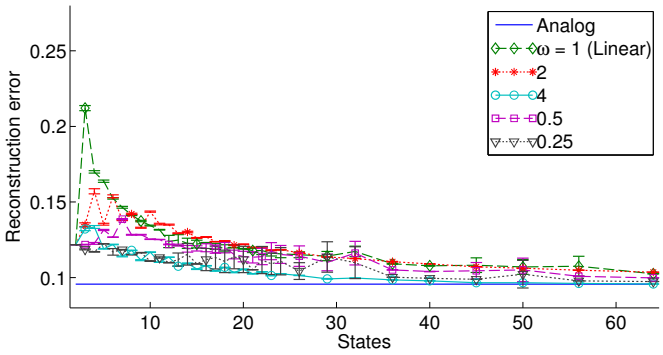


Fig. 20: Quality of reconstruction for varied ω and a finite number of states k with stochastic state switching. These results are virtually identical to those from round-to-nearest, Fig. 16. Since the ideal (analog) solution consists mainly of low-valued weights (Fig. 18), the benefits of a decreased gap between states in the region around w_{bias} quickly overcomes the benefits of stochasticity.

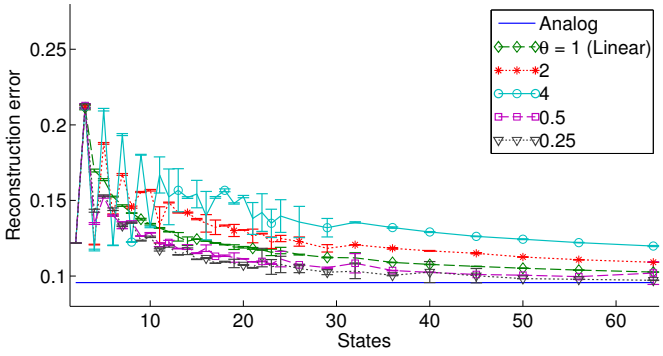


Fig. 21: Reconstruction performance of a θ distribution with stochastic switching. Note that, since the solution space exists solely in the medial region, and stochastic switching provides enough switching capability to escape the extreme states on either end of the spectrum, any distribution with $\theta < 1$ will outperform other distributions (even linear). Performance oscillates for low k with $\theta > 1$ due to the alternating presence and absence of the value 0.5 as an available device state. When 0.5 is not present, w_{bias} gets set to the closest value greater than 0.5. Since $\theta > 1$ has high state density at the extremes, the distance from w_{bias} to the first state representing a positive (excitatory) weight will always be less than if 0.5 is present.

positive and negative inputs and weights, as in an image reconstruction task. It is our hope that our results provide targets for memristive devices. Regardless of state switching characteristics, given a distribution that stays within a power law of $\omega = [0.5, 2.0]$, 16 stable states, or 4-bit resolution, is sufficient for a polarized dataset, such as MNIST. This number may be decreased greatly, even down to 4 stable states, if state switching is favored over stability at a low threshold. To a lesser extent, the number of required states may be decreased if there is an amount of stochasticity to compensate for partial updates lost as a result of thresholding behavior. A dataset with values distributed throughout the input space, such as natural image patches, requires a slightly higher number of stable states, with 20 stable states performing similarly to analog. Unlike a polarized dataset, the reconstruction task benefitted greatly from non-linear distributions. This is due to the w_{bias} parameter in the architecture, responsible for denoting positive and negative regions of the device's possible states. For problems like this, any distribution with a high-density region of weights near w_{bias}, i.e. anything except $\theta > 1$, is preferable. Our experiments have shed light on the conditions in which learning algorithms will break down, and what is required of the materials used for learning. Without regarding these limitations, hardware implementations might perform worse than expected.

Acknowledgment

This work was supported by the National Science Foundation under award # 1028378 and by the Defence Advanced Research Projects Agency 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] D. Querlioz *et al.*, "Bioinspired networks with nanoscale memristive devices that combine the unsupervised and supervised learning approaches," in *Nanoscale Architectures (NANOARCH), 2012 IEEE/ACM International Symposium on*, July 2012, pp. 203–210.

[2] C. Zamarreño-Ramos *et al.*, "On spike-timing-dependent-plasticity, memristive devices, and building a self-learning visual cortex," *Frontiers in Neuroscience*, vol. 5, pp. 1–22, 2011.

[3] S. H. Jo *et al.*, "Nanoscale memristor device as synapse in neuromorphic systems," *Nano letters*, vol. 10, pp. 1297–1301, 2010.

[4] S. Gaba *et al.*, "Stochastic memristive devices for computing and neuromorphic applications," *Nanoscale*, vol. 5, pp. 5872–5878, 2013.

[5] F. Alibart *et al.*, "High precision tuning of state for memristive devices by adaptable variation-tolerant algorithm," *Nanotechnology*, vol. 23, p. 075201, 2012.

[6] B. Rubin *et al.*, "Long memory lifetimes require complex synapses and limited sparseness," vol. 1, pp. 1–14, 2007.

[7] C. J. Rozell *et al.*, "Sparse coding via thresholding and local competition in neural circuits," *Neural computation*, vol. 20, pp. 2526–2563, 2008.

[8] W. Woods *et al.*, "On the influence of synaptic weight states in a locally competitive algorithm for memristive hardware," in *Proceedings of the 2014 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH 2014)*. IEEE Press, July 2014, pp. 19–24.

[9] R. Berdan *et al.*, "Memristive devices as parameter setting elements in programmable gain amplifiers," *Appl. Phys. Lett.*, vol. 101, p. 243502, 2012.

[10] P. Schultz *et al.*, "Replicating kernels with a short stride allows sparse reconstructions with fewer independent kernels," 2014. [Online]. Available: http://arxiv.org/abs/1406.4205v1

[11] Y. LeCun *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, pp. 2278–2324, 1998.

[12] M. Driels *et al.*, *Determining the Number of Iterations for Monte Carlo Simulations of Weapon Effectiveness*. Naval Postgraduate School, 2004.

[13] T. Masquelier *et al.*, "Unsupervised learning of visual features through spike timing dependent plasticity," *PLoS Comput Biol*, vol. 3, p. e31, 2007.

[14] J. Zylberberg *et al.*, "A sparse coding model with synaptically local plasticity and spiking neurons can account for the diverse shapes of v1 simple cell receptive fields," *PLoS Comput Biol*, vol. 7, p. e1002250, 2011.

[15] J. K. Kim *et al.*, "Efficient Hardware Architecture for Sparse Coding," *IEEE Transactions on Signal Processing*, vol. 62, pp. 4173–4186, 2014.