Non-Traditional Input Encoding Schemes for Spiking Neuromorphic Systems

Catherine D. Schuman
Computational Data Analytics
Oak Ridge National Laboratory
Oak Ridge, TN, USA
schumancd@ornl.gov

James S. Plank
Department of EECS
University of Tennessee
Knoxville, TN, USA
jplank@utk.edu

Grant Bruer
Department of EECS
University of Tennessee
Knoxville, TN, USA
gbruer@vols.utk.edu

Jeremy Anantharaj
Department of EECS
University of Tennessee
Knoxville, TN, USA
jananth1@vols.utk.edu

***Abstract*—A key challenge for utilizing spiking neural networks or spiking neuromorphic systems for most applications is translating numerical data into spikes that are appropriate to apply as input to a spiking neural network. In this work, we present several approaches for encoding numerical values as spikes, including binning, spike-count encoding, and charge-injection encoding, and we show how these approaches can be combined hierarchically to form more complex encoding schemes. We demonstrate how these different encoding approaches perform on four different applications, running on four different neuromorphic systems that are based on spiking neural networks. We show that the input encoding method can have a significant effect on application performance and that the best input encoding method is application-specific.**

***Index Terms*—spiking neural networks, neuromorphic systems, input encoding, value-to-spike**

I. Introduction

Spiking recurrent neural networks have been shown to be a powerful computational paradigm [1], [2], and their deployment on emerging neuromorphic hardware systems can result in realizing these computational abilities with low power, energy efficient implementations [3]. However, there are a variety of hurdles to overcome when implementing real applications on spiking neuromorphic systems. One of the fundamental hurdles is how to encode input appropriately from numerical or categorical data into spikes on input neurons. Traditional artificial neural networks, which use matrix-vector operations in the transformation of inputs to outputs, can operate directly on numerical values. Spiking neural networks, on the other hand, typically cannot operate directly on numerical data; instead, those values must be converted into the form of spikes or spike trains. However, making the conversion from numerical data to spikes or spike-trains is non-trivial. The choice of encoding scheme can affect not only performance on an application in terms of accuracy, but also the rate at which data can be processed and the energy efficiency of the system, which is particularly important for neuromorphic implementations.

In this work, we present a variety of different input encoding schemes for spiking neural networks and evaluate their effectiveness on four applications, implemented on four neuromorphic implementations of spiking neural networks. These applications are: an image classification task, a time series classification task, a classic control task, and an autonomous robot control task. In particular, we discuss three types of encoding approaches and how they can be combined hierarchically to achieve complex encoding schemes and encode relatively high input resolution over short time periods. We demonstrate that the input encoding approach can have a significant effect on application performance and that often the encoding approach may be derived from the characteristics of the input that the applications produce.

II. Background and Related Work

The two most common spiking neural network input encoding approaches are both inspired by biological neural networks: rate coding and temporal coding [4]. In Maass's canonical work on spiking neural networks [1], he describes encoding schemes based on the timing of spike arrivals into spiking neural networks. Temporal coding of this form is also the basis of the spiking neural network back-propagation analog, SpikeProp [5] and has also been used more recently in new back-propagation analogs for neuromorphic systems [6]. Temporal coding has also been popular in the neuromorphic community, especially when used with learning algorithms such as spike-timing dependent plasticity (STDP) [7].

In [8], Querlioz, *et al.*, use a variety of rate encoding schemes, including in-phase fixed rate encoding, where the rate is proportional to the value being encoded, and a Poissonian rate encoding, where the time constant is inversely proportional to the value being encoded. They showed that the precise input encoding style (of these three approaches) did not affect the performance on a particular application (image recognition) using STDP as a weight training mechanism and a homeostasis rule for neuron threshold training. Rate coding has also been popularly used in neuromorphic systems [9].

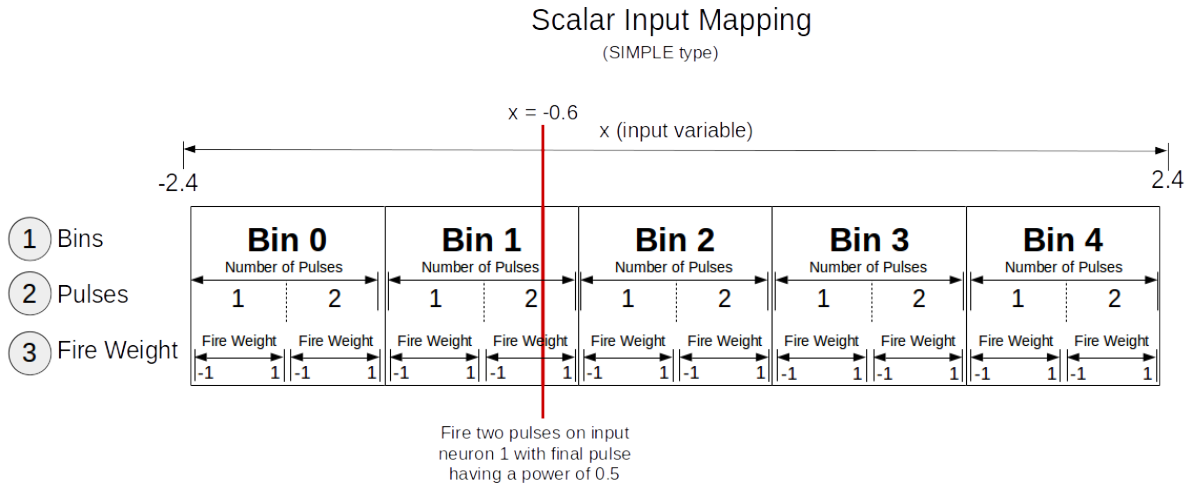

Fig. 1. Example of combining binning, spike-count encoding, and charge-injection encoding to convert a single value into spikes. In this example, we use $b_k = 5$, $p_k = 2$, $c_k = -1$, and $C_k = 1$. The value $x = -0.6 \in [-2.4, 2.4]$ is converted to two pulses on input neuron 1 (corresponding to bin 1) with the final pulse having a charge power of 0.5.

In [10], Yanguas-Gil compares the results of multiple types of encodings, including single-spike temporal encoding, temporal encoding followed by a spike-train, rate encoding, and modifying the input signal, and that for the MNIST dataset, temporal coding over a short period of time is sufficient for maintaining high accuracy. The different encoding schemes are highly affected by the neuron model's ability to process and output information. Single spike encoding schemes and the constant input signal were not able to function well on the MNIST task when restricted to binary outputs (spike or no spike). Since many neuromorphic platforms are restricted to this type of output behavior in the neurons, these encoding approaches are likely not sufficient.

A recent approach for input encoding is to binarize the input by scaling input values to the range of $[0, 1]$ and using that scaled value to represent the probability of the input neuron firing. This approach has been used for training spiking neural networks via back-propagation-style algorithms as well [11].

A more recently introduced approach to encoding visual inputs in particular for spiking neural networks is to use a neuromorphic vision sensor. Rather than capturing images frame-by-frame, neuromorphic vision sensors [12], [13] instead report event-driven information, where each pixel registers an event when a change has occurred. This type of streaming input to represent images has been used for certain image classification tasks in neuromorphic systems [14]. This approach is relatively limited, as it can only apply to certain input types and applications, in this case image processing or classification.

The encoding schemes described above are not universally applicable for all neuromorphic applications. For example, the neuromorphic vision sensor approach, or using filters to produce salient binary features only apply to image data. For real-time processing applications on neuromorphic systems, such as those requiring real-time control, temporal encoding schemes and rate encoding schemes may not be as effective or even applicable, because data may be arriving at a rate that is not capable of being encoded quickly enough to make decisions in real time. Because of this lack of general applicability of common approaches presented in the literature, we investigate three simple encoding approaches that can be hierarchically combined to produce more complex encoding schemes that (1) can be applied to a wide variety of input data types and (2) can represent single input values over a very short period of time so that they can be applied to real-time classification or control tasks.

III. Value-to-Spike Encoding

We introduce and investigate a variety of encoding schemes beyond the typical temporal or rate coding schemes, particularly those that require relatively few spikes per value to encode and that can be encoded over a short period of time. We describe three different ways to convert a value into spikes and how combinations of those approaches can be created to form more complex encoding schemes. For each of these schemes, we assume that for a single input k (where a single input may correspond to, for example, readings for an individual sensor), most values x_k that that input can take on will fall into a range $m_k \leq x_k \leq M_k$.

A. By Bin

One approach for encoding an input value in a single spike in a single time step is to create multiple input neurons corresponding to that input and then encode the value by spiking on a particular neuron. We call this approach "binning." For binning, for an input k, we define a parameter b_k to be the number of bins for input k, and split the range $[m_k, M_k]$ into b_k equal-sized bins, and extend the first and last bin to account for all possible values. For example, if $m_k = 0$,

$M_k = 1$, and $b_k = 4$, then the four bins would correspond to $(-\infty, \frac{1}{4}], (\frac{1}{4}, \frac{1}{2}], (\frac{1}{2}, \frac{3}{4}], (\frac{3}{4}, \infty)$. We then create an input neuron for each bin, and when a new value x_k is to be encoded for input k, we find the appropriate range and then spike on the corresponding input neuron for that bin.

B. By Number of Spikes

A second approach, which we call "spike-count" encoding, is to convert input values into a small number of spikes arriving at a fixed rate. We typically restrict ourselves to a small number of spikes in order to allow for streaming input. In this case, spike-count encoding has a parameter p_k, which is the maximum number of pulses or spikes to encode a single value. Similar to binning, we define a set of p_k equal-sized bins, but extending the first and last bin to account for all possible values. Here, rather than having multiple input neurons and assigning an input neuron to each bin, we instead have a single input neuron. Input values x_k that fall in the first range correspond to 1 pulse applied to the neuron, the second range to 2 pulses, and so on.

C. By Charge Value

A third approach relies on the ability not only to inject a spike into a system in order to force an input neuron to fire, but also to inject a specific charge value into a neuron, which may or may not cause that neuron to fire. This "charge-injection" encoding approach requires two parameters, c_k and C_k, which define a minimum charge value (c_k) and a maximum charge value (C_k).

Note that the function to linearly map a range $[a, b]$ to $[c, d]$ is:

$$h(x) = c + \frac{d-c}{b-a}(x-a) \tag{1}$$

We will use this mapping several times in different encoding approaches. To encode a value x_k into input for the "charge-injection" encoding scheme, we first let $x'_k = \max\{\min\{x_k, M_k\}, m_k\}$ to force is to be in the range $[m_k, M_k]$. Then, we will map the value x'_k from the range $[m_k, M_k]$ to the range $[c_k, C_k]$ (using the function $h : [c_k, C_k] -> [m_k, M_k]$ defined above) to determine the charge value to apply to the input.

D. Combining Encoding Schemes

All three of the previous encoding schemes are fairly straightforward to implement. In this work, we introduce combinations of the previously discussed encoding schemes to produce more complex encoding schemes. In fact, we assume that at all times, all three input encoding schemes are used simultaneously and that a particular encoding scheme can be "turned off" by certain parameter settings. For example, binning can be turned off by setting $b_k = 1$, spike-count coding can be turned off by setting $p_k = 1$, and charge-injection coding can be turned off by setting $c_k = C_k$. We then apply the encoding schemes hierarchically. First, for a given value x_k, we find the appropriate bin that the value should belong in, which gives us a new range $[m'_k, M'_k]$ for values that occur within that bin. This determines the neuron to which the corresponding inputs will be applied. Once the bin is defined, we then use one of the three *inter-bin encoding functions*, f, defined below (simple, flip-flop, or triangle) to map the value from $[m'_k, M'_k]$ to the range $[0, 1]$. In other words, $f(x_k) \in [0, 1]$ is determined. In order to perform spike-count encoding, we perform the following operations to determine how many pulses to apply:

$$p = \min\{\lfloor p_k * f(x_k) \rfloor + 1, p_k\} \tag{2}$$

Finally, once p (the number of pulses to apply to the corresponding bin) has been determined, we then want to map the charge value $p_k * f(x_k)$ linearly from the range $[p, p+1)$ to $[c_k, C_k)$ to find the charge value of the last pulse to be applied. An example of this approach using Equation 1 as f is given in Figure 1, where $b_k = 5$, $p_k = 2$, $c_k = -1$, and $C_k = 1$. The figure includes an encoding of the value -0.6, which results in two pulses being applied to bin 1, with charge values of 1 for the first pulse, and 0.5 for the second.

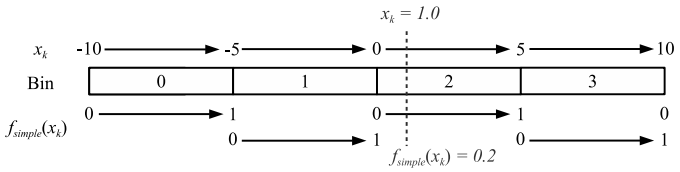

Fig. 2. Example of the simple encoding scheme for four bins. In particular, an input value of $x_k = 1.0$ is to be encoded. First, the appropriate bin is found. Since $x_k \in [0, 5)$, the correct bin is bin 2. Then $f_{simple}(x_k)$ maps x_k from $[0, 5)$ to $[0, 1)$, which converts the value to 0.2.

For **simple** inter-bin encoding (Figure 2), suppose we have applied the binning approach, and we have reduced ourselves to the case in which x_k is in the possible range of $[m''_k, M''_k]$. Simple encoding simply does the $h(x_k)$ linear mapping (as described in Equation 1) from $[m''_k, M''_k]$ to $[c_k, C_k]$.

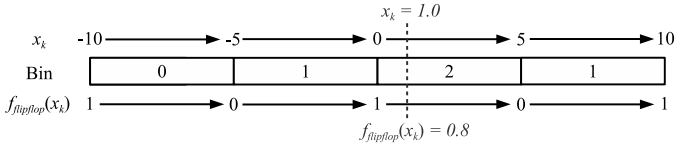

Fig. 3. Example of the flip-flop encoding scheme for four bins for the value $x_k = 1$. As in Figure 2, it will map to bin 2. However, because bin 2 is an even bin, $f_{flipflop}(x_k) = f_{evenbin}(x_k) = 0.8$, as in Equation 3.

Flip-flop encoding (Figure 3 uses a similar linear mapping function $h(x_k)$ for odd-numbered bins (i.e., bin 1, bin 3, bin 5, ...), but it flips the mapping for even-numbered bins, such that if x_k is at the max end of the bin, then the output charge is the minimum c_k. The intent of flip-flop encoding it to make the mapped values continuous when x_k crosses bin boundaries.

$$f_{evenbin}(x_k) = C_k + \frac{c_k - C_k}{M''_k - m''_k}(x_k - m''_k) \tag{3}$$

Triangle encoding has the bins overlap, so that most input values produce spikes in adjacent bins. Specifically, when b_k neurons are employed, the values between m_k and M_k are partitioned into $(b_k + 1)$ *effective* bins. Effective bins 0 through b_k produce spikes for neurons 0 through b_k, with the $f()$ mapping being like **simple**. Additionally, effective bins 1 through $b_k + 1$ produce spikes for neurons 0 through b_k, with the $f()$ mapping going from 1 to 0. Thus, most values produce spikes for adjacent pairs of input neurons.

For example, if $m_k = 0$, $M_k = 1$ and $b_k = 4$, then there are five effective bins, each with a range of 0.2. The four actual bins overlap, and correspond to $(-\infty, 0.4], [0.2, 0.6], [0.4, 0.8], [0.6, \infty)$.

$$f(x_k) = c_k + (C_k - c_k)\left(1 - \frac{1}{\frac{M_k'' - m_k''}{2}}\left|x_k - \frac{m_k'' + M_k''}{2}\right|\right) \quad (4)$$

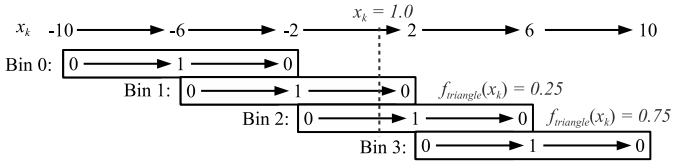

Fig. 4. Example of triangle encoding with four bins for the value $x_k = 1$. In this case, because we have overlapping bins, x_k falls both in bin 1 and bin 2. It falls in the second half of bin 1, which results in an applied pulse value of $f_{triangle}(x_k) = 0.25$ for bin 1. It falls in the first half of bin 2, which results in $f_{triangle}(x_k) = 0.75$ for bin 2.

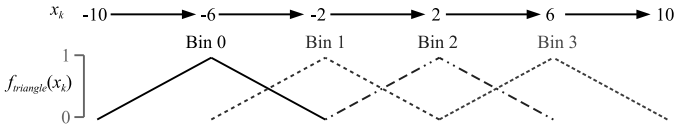

Fig. 5. Illustration of how $f_{triangle}$ encodes values across four overlapping bins.

See Figures 2 through 4 for a comparison of the same input value applied for each of the three methods when $m_k = -10$, $M_k = 10$, and $b_k = 4$. Note that simple encoding has discontinuous charge input to the network. Flip-flop encoding makes the charge of the input continuous, but moving x_k across a peak switches from a large charge applied to one bin to a large charge applied to another bin, which is highly discontinuous.

Triangle encoding smooths these bin transitions by overlapping the bins. Thus, the input pulses become fully continuous at the cost of having to use more bins to get the same sensitivity per bin (where sensitivity is how much the input charge changes with respect to the input value x_k, i.e., the slope). Figure 5 shows how the overlapping bins work with triangle encoding with four bins.

IV. Spiking Neuromorphic System Framework

In order to study the effects of these different input encoding techniques on spiking neural network and neuromorphic performance, we utilize the TENNLab's common software framework [15] and the Evolutionary Optimization for Neuromorphic Systems (EONS) training method [16], both of which are briefly described below.

The TENNLab software framework is a platform for enabling the building applications for neuromorphic systems and the study and evaluation of different neuromorphic implementations [15], [17]. The software framework implements a common interface to multiple spiking neuromorphic systems, including the three used in this work. The common interface allows for an application code to be written once and then trained on multiple neuromorphic platforms without any application software changes. The four applications in this work were all implemented once and then compiled and deployed on the different neuromorphic implementations. The software framework also implements all of the encoding schemes described in Section III.

The primary learning algorithm used in the framework is Evolutionary Optimization for Neuromorphic Systems (EONS) [18], though other learning approaches are either supported (e.g., reservoir computing) or are currently being implemented (e.g., back-propagation). For the purposes of these tests, EONS is convenient to use because it will attempt to build the best network suited for each individual coding scheme without hand-tuning and without having to change the underlying algorithm at all to accommodate for different coding schemes. The goal with using EONS rather than another algorithm is to give each encoding scheme its best chance of succeeding, rather than selecting a particular plasticity rule or gradient-descent based implementation, which are not applicable to all possible encoding schemes.

In this work, we utilize three spiking neural network or neuromorphic implementations that have been described in previous work and that we briefly summarize here:

- NIDA: A spiking neural network model with integrate-and-fire neurons and synapses with a simplified plasticity mechanism in which the spiking neural network is embedded into a three-dimensional space and the delays on synapses depend on the distance between two neurons in the 3D space [19].
- DANNA2: A fully digital programmable neuromorphic implementation with integrate-and-fire neurons, restricted synaptic connectivity, and synapses with weights and delays [20]. DANNA2 can either be implemented on an FPGA or in a custom chip implementation. We include two versions of DANNA2, one with shorter maximum synaptic delays that is more efficiently implemented on hardware (DANNA2-short) and one longer maximum synaptic delays that is better suited to certain applications (DANNA2-long).
- mrDANNA: A mixed analog-digital programmable neuromorphic implementation with integrate-and-fire neurons and synapses with a simple plasticity mechanism. In this implementation, each of the synapses are implemented using two metal-oxide memristors [21].

We utilize four applications with the neuromorphic computing framework. Two of those applications are classification tasks: one with image data (not well-suited to native spiking networks) and one with time-series data. We also include two control tasks. The four tasks are:

- Radio: A satellite radio problem signal classification task, previously described in [22].
- 3-MNIST: A restricted version of the MNIST task [23] in which the goal is to classify an handwritten digit image as either a 3 or not a 3. We also use a limited training and testing set size.
- Pole: The canonical pole balancing task, previously described in several works [24], [25]. We use the version of the pole balancing task that does not include velocities as input, only cart and pole position.
- Robonav: An autonomous robotic navigation task, previously described in detail in [26].

A. Motivation for Alternative Encoding Schemes

We use four applications here, two of which require real-time behavior. Both pole and robonav are control tasks in which the network takes input information every 0.02 seconds (20 milliseconds) and has to make a control decision before the next 0.02 seconds has elapsed. This means that the input has to be communicated from the sensors to the neuromorphic device, the network on the device has to process the input and produce output, which is then converted to the control decision to produce the next set of inputs in less than 0.02 seconds.

To reduce power consumption, many neuromorphic systems operate at a very low clock rate. For example, IBM's TrueNorth operates at a 1 kHz clock rate [27]. For both of the pole and robonav tasks, this low clock rate means that the network would have less than 20 clock cycles to make a decision based on the current set of inputs. Assuming input can be provided to the neuromorphic system only on the clock cycle, this significantly restricts the number of values that can be represented either by temporal coding and rate coding on a single neuron (at most 20 values for both). Further, this assumes that the network itself does not need all of the input in the system to be processed before needing some number of cycles to make the decision, which is likely a poor assumption. It is more likely that the network would need at least half of that time to use the input data to make the control decision, which restricts the number of possibly represented values to 10. By combining binning, spike-count encoding, and charge-injection encoding into more complex encoding schemes, we can use these encoding schemes to achieve higher resolution on the inputs over very short time windows, making them amenable to real-time control and classification tasks.

V. Results

To evaluate the different encoding schemes, we test all combinations of the parameter settings listed in Table I, eliminating those that do not make sense (e.g., triangle and flip-flop for $b_k = 1$ or $c_k = C_k$) and those that result in no information being passed into the network (e.g., $b_k = 1$, $p_k = 1$, and $c_k = C_k$). For each set of different valid parameter settings, we run 100 test runs of EONS on a particular neuromorphic implementation and application combination (i.e., 100 separate evolutions), in order to alleviate variations due to the random initialization of the populations in the genetic algorithm. The genetic algorithm parameters of EONS for these tests are given in Table II.

TABLE I
Evaluated Parameters

Parameter	Evaluated Settings
b_k (Number of Bins)	1, 2, 4, 8
p_k (Number of Spikes)	1, 2, 4, 8
$[c_k, C_k]$ (Range of Charge Values)	$[0, 0.5]$, $[0, 1]$, $[0.25, 0.5]$, $[0.25, 1]$, $[0.5, 0.5]$, $[1, 1]$
Inter-bin Encoding Type	Simple, Flip-Flop, Triangle

TABLE II
Evolutionary Optimization Parameters

EONS Parameter	Value
Population size	1000
Crossover rate	0.5
Mutation rate	0.9
Selection type	Tournament
Tournament Size	5

A. Bin Encoding

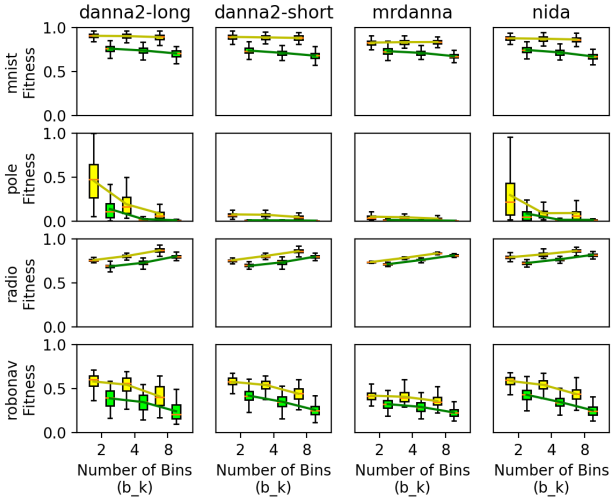

Fig. 6. Training (yellow) and testing (green) results for binning approaches alone. The mean values are connected by line plots in order to show trends when varying the number of bins.

Figure 6 gives the results for the four applications and four implementations (NIDA, mrDANNA, DANNA2-long, and DANNA2-short) when only using binning as the encoding approach. In this case, we consider the test cases in which $p_k = 1$, $c_k = C_k$, and the encoding scheme is set to simple. This figure indicates that the appropriate number of bins to choose is application specific, rather than neuromorphic implementation specific. That is, in all cases, the same general

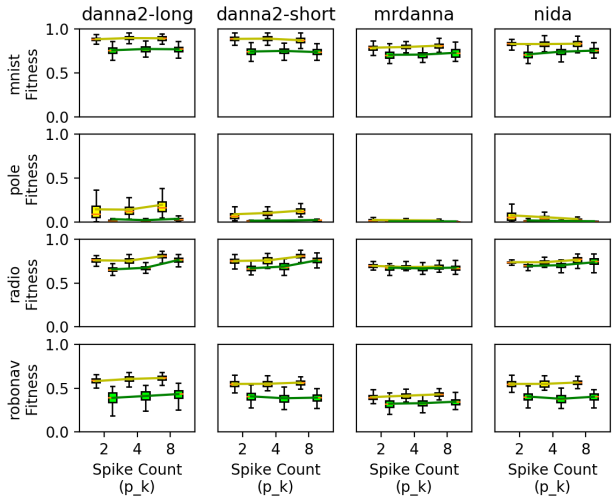

Fig. 7. Training (yellow) and testing (green) results for spike-count encoding approaches alone. The mean values are connected by line plots in order to show trends when varying the number of spikes.

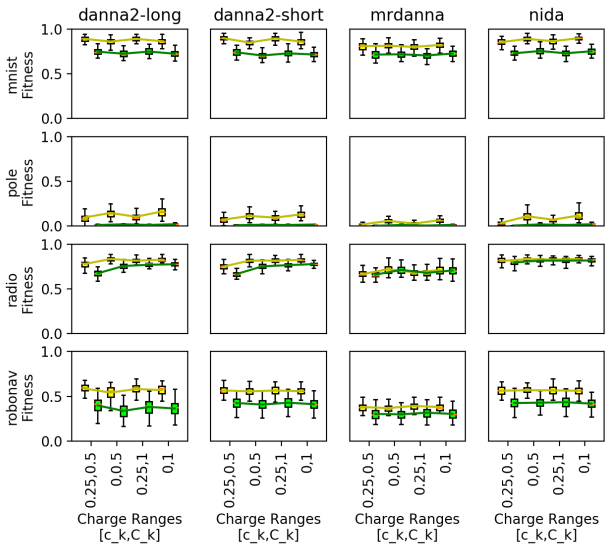

Fig. 8. Training (yellow) and testing (green) results for charge-injection encoding approaches alone. The mean values are connected by line plots in order to show trends when varying minimum and maximum values of the range.

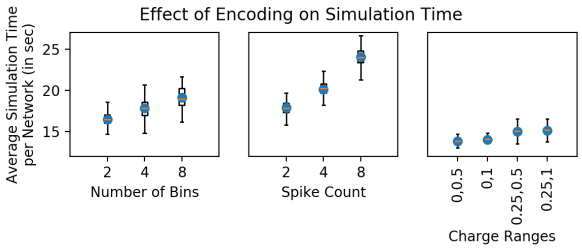

Fig. 9. Average simulation time per network for DANNA2-long on MNIST based on different encoding schemes.

trends hold across all neuromorphic implementations, while the trends vary significantly from application to application. In particular, for MNIST, the training fitness tends to remain relatively stable as the number of bins increases, but the testing fitness decreases as the bins increase. This indicates that at least for MNIST, a low-level input resolution is sufficient and that higher resolution levels can lead to overfitting. Since MNIST's images are practically binary, this is not a surprising outcome. For both pole balancing and robonav, increasing the number of bins decreases both the training and testing values significantly. Unlike the other tasks, however, radio benefits from increasing the number of bins. Since radio only has two inputs (the real and complex value of the radio signal), but since each of those values is relatively high resolution, it is not surprising that more bins gives better performance.

B. Spike Count Encoding

Figure 7 gives the results for the four applications and four implementations when only using spike-count encoding. In this case, $b_k = 1$, $c_k = C_k$, and the encoding scheme is set to simple. For spike-count encoding, the trends are much less stark and are less consistent across implementations as the binning approach. For example, on the pole balancing task, more pulses per bin improved performance for both of the DANNA2 implementations, but decreased performance for the NIDA implementation. Across the board, however, there was relatively little difference between using different numbers of spike counts in the encoding.

C. Charge-Injection Encoding

Figure 8 shows the results when using charge-injection encoding only ($b_k = 1$, $p_k = 1$, and simple encoding type). Once again, as in the spike-count encoding approach, the results are fairly consistent across different ranges. Unlike the other approaches, for the radio task, the training and testing results tracked very closely together, indicating good generalization ability for this task. The "resolution" on encoding for charge-injection is theoretically much more than for binning or spike-count in a fixed time period (i.e., it is restricted only by the resolution at which charge can be injected into the spiking neural network or spiking neuromorphic system). However, depending on the range of value encoded, it is likely that to stimulate a single fire, multiple incoming pulses would be required, which may not be desired for a given application.

D. Input Encoding Effects on Network Performance

It is worth noting that different input encoding schemes can affect the size of the networks produced, as well as the amount of activity in the networks. Both of these factors influence the amount of energy or power required for a given neuromorphic implementation. In this work, we simulate each neuromorphic implementation using discrete event simulations, which means that networks that have more activity require longer to simulate. We cap the wall-clock time for training rather than the number of generations, so we are implicitly biasing

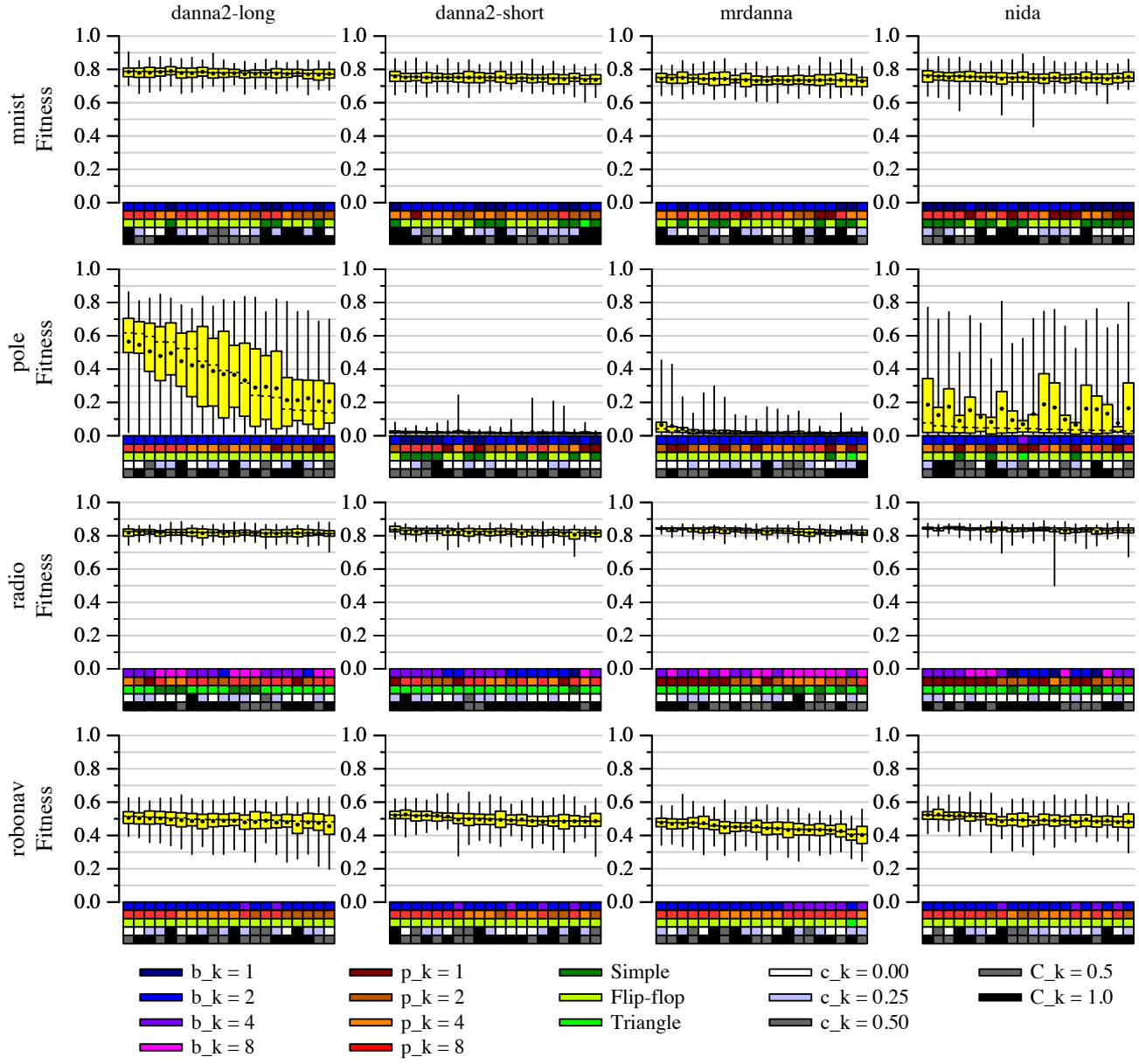

Fig. 10. Top 20 encoding schemes for each implementation/application combination based on median value of the testing performance. The encoding schemes are represented by color blocks. The top 20 encoding schemes are shown from left to right in each of the individual plots. The first row of color blocks indicates the number of bins b_k, the second row of color blocks is spike count per bin p_k, the third row is charge encoding type (simple, flip-flop or triangle), and the fourth and fifth rows represent c_k and C_k, respectively.

towards input encoding schemes that are simultaneously more accurate, have less activity, and are smaller (i.e., are more power efficient in real hardware implementations). Though we do not explicitly measure activity as part of the simulation, we can use average simulation time per network as a surrogate for how much activity there is the network. Figure 9 shows how different input encoding schemes affect network simulation time for the MNIST task for the DANNA2-long neuromorphic implementation. As can be seen in this figure, all encoding schemes have an effect on network performance. Both binning and spike-count increase the simulation time (thus, increase the activity) when increasing the number of bins/spikes. Simulation time is not an issue when moving to a physical hardware implementation of a neuromorphic system, but it correlates with the amount of activity in the network. As such, encoding schemes that use more bins or more pulses per bin will use more energy for the same task.

E. Complex Encoding Schemes

After eliminating the pathological cases for combinations of the parameters given in Table I, we are left with 230 different

valid combinations of parameters or encoding schemes. These 230 encoding schemes include the binning encoding only, spike count encoding only, and charge-injection only input schemes, which can be achieved by setting the parameters as described in each of the previous sections. We determine the best encoding schemes by ranking according to the median value across the 100 test cases in order to alleviate issues associated with random variation in the initial EONS populations. Figure 10 shows graphically the top twenty encoding schemes for each neuromorphic implementation/application combination, along with box plots on their performance. In most cases, the box plots show that there is relatively little difference in performance across the top twenty different encoding schemes, except for the pole balancing task (row 2 in Figure 10). Some clear patterns emerge in the types of encoding schemes that are successful, however.

For MNIST (the first row), a smaller number of bins ($b_k \leq 2$) and the flip-flop encoding type dominated the top 20 encoding schemes, while the number of spikes for spike-count encoding and the minimum and maximum values for charge-injection encoding have less impact on performance, which is consistent with what is seen in Figures 7 and 8.

For pole balancing (the second row of box-plots and encoding scheme color blocks in Figure 10), there is a radical difference in the performance of DANNA2-long and NIDA as compared with the other two implementations because of architectural limitations of DANNA2-short and mrDANNA (shorter realizable synaptic delays) that have more of an impact than input encoding schemes, but trends in best encoding schemes still emerge. In particular, the best performing encoding schemes had $b_k = 2$ and the encoding scheme set to flip-flop, while once again the spike-count parameter p_k and the minimum and maximum values for charge-injection encoding had less of an impact. We speculate that the two bins are chosen because they can be used to differentiate between positive and negative values of the cart's position and velocity, while the other parameters are used to indicate magnitude of the values.

Unlike all of the other applications, the radio task benefited from a larger number of bins ($b_k \geq 4$) in most cases. Requiring a larger number of bins is consistent with the results shown in Figure 6 and is consistent with the high resolution of the values represented in the radio dataset. Perhaps the more interesting result for radio is that it was the only task that consistently benefited from setting the encoding type to triangle. Triangle encoding really only makes sense when the number of bins is greater than three ($b_k \geq 3$); otherwise, a large range of the input values for triangle will spike all of the bins. Though triangle enables a higher resolution representation, because it requires a larger number of bins, this will correspond with a greater simulation time and thus, fewer trained generations. Thus, a larger number of bins and triangle encoding will only dominate the encoding schemes for an application if the gains in accuracy due to higher resolution are large enough to outweigh the implicit penalties of longer simulation time.

Similar to the pole balancing task, the Robonav task benefits

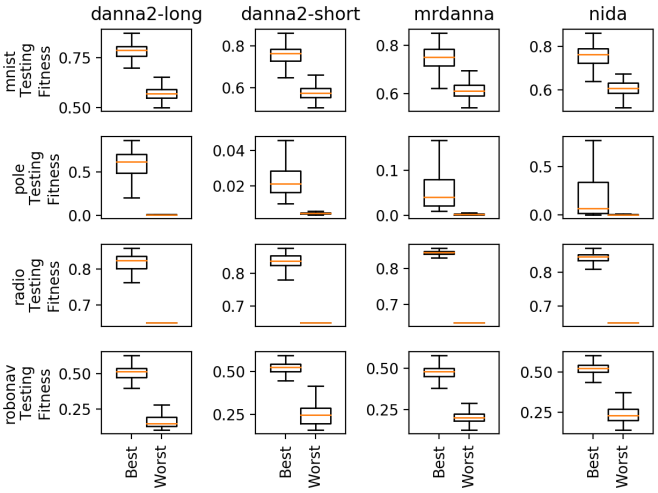

Fig. 11. Variation in performance between best and worst performing encoding schemes.

from using two bins and flip-flop encoding as well. Once again, we speculate that the two bins are used to represent positive and negative values for this control task. In general, larger numbers of spikes for spike-count encoding were also helpful for the robonav task, indicating that it benefited from higher resolution input value magnitude representation than what was generally required for the pole balancing task. Based on the results from both pole and robonav, we can conclude that $b_k = 2$ and the flip-flop encoding schemes are at least good starting points for real-time control-style tasks.

F. Key Observations

Here we summarize three of the key observations found from this comparison of encoding schemes.

Key Observation 1: The encoding scheme chosen has an impact on application performance. The variation in performance between the best and worst encoding schemes (where best and worst are determined by median value across 100 tests) is shown in Figure 11. Clearly the difference in performance is beyond the random initial population variation effects of the evolutionary process. To further illustrate the impact of the encoding scheme on performance, we show the full 230 encoding schemes performance box plots and color blocks for the DANNA2-long on the pole balancing task in Figure 12. Here, we can see the drastic difference that input encoding scheme has on performance of the evolution, with clear patterns in the parameters chosen. For example, not only do fewer bins consistently perform better (as can be seen in Figure 10), but more bins consistently perform worse. These types of patterns emerge for all of the implementation/application combinations, though we have only shown one here.

Key Observation 2: The appropriate encoding scheme depends primarily on the application, rather than the implementation. Even though the neuromorphic implementations have different levels of parameter resolution and different architectural details (e.g., neuron or synapse implementations),

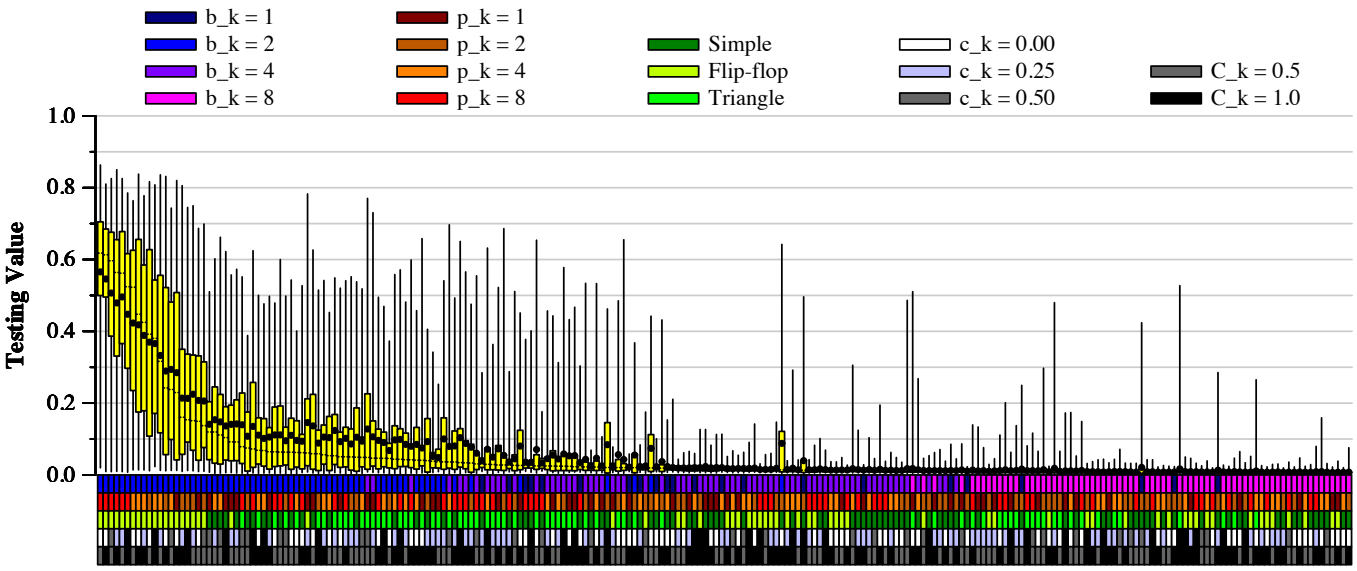

Fig. 12. All 230 encoding schemes, ranked from best to worst (left to right) on the pole balancing application, running on Danna2-long. The box plots showing the testing performance across all 100 evolutions for each encoding scheme are shown, along with the encoding scheme itself as a color block.

these architectural details did not have a major effect on which encoding schemes performed well. Trends in how different encoding schemes (more bins vs. fewer bins, etc.) were generally consistent across different implementations. For radically different architectures than those presented here, especially those utilizing different forms of plasticity or utilizing exotic device types, it is likely that different encoding schemes will have different performance effects. However, we expect that these conclusions will hold for more traditional fully digital neuromorphic implementations, such as Intel's Loihi [28] or IBM's TrueNorth [27].

Key Observation 3: All applications benefited from "complex" encoding schemes. Most of the top 20 encoding schemes for each application/implementation combination utilized some encoding combination of binning, spike-count encoding, and charge-injection encoding to achieve the higher fitness values, and the top performing encoding scheme in every case was a "complex" encoding scheme. Unlike temporal and rate coding, which both require longer intervals to encode more resolution in input information, by combining binning, spike-count, and charge-injection encoding, higher resolution can be achieved without increasing the amount of time required to encode the input. We believe that this higher resolution achieved by the complex encoding methods results in the better performance of the complex encoding scheme over the binning, spike-count, or charge-injection encoding approaches alone.

VI. Discussion and Conclusion

In this work, we present three simple approaches for encoding numerical values into input for spiking neural networks (binning, spike-count encoding, and charge-injection encoding) and demonstrate how they can be combined hierarchically to achieve more complex encoding schemes. We show how these different encoding schemes perform on four different applications and four spiking neural network or neuromorphic implementations. We show that the encoding scheme does have a significant impact on performance of the spiking neural network or neuromorphic system, and we show that the encoding scheme that is most appropriate to use is more dependent on the application than it is on the particular neuromorphic or spiking neural network implementation.

In future work, we intend to compare our new encoding schemes with rate and temporal coding approaches in order to quantify differences in application accuracy and power and energy consumption for the different encoding schemes, while also factoring in practical considerations of using rate and temporal encoding schemes for real-time applications. The goal of this work was to investigate the effect of encoding schemes; in future work, we plan to investigate the specific reasons why certain encoding schemes perform very well and why others perform very poorly. Since it may not be obvious which encoding scheme will be best for a given application, we plan to expand the EONS framework to allow for hyperparameter optimization of the input encoding schemes. That is, rather than hand-tuning the encoding scheme or performing a grid search as we have done here, we plan to use EONS to discover the appropriate input encoding parameters and schemes to use for a given application and implementation combination. Additionally, we plan to investigate how input encoding schemes affect the performance of different spiking neural network learning mechanisms, including spike-timing dependent plasticity and liquid state machines.

ACKNOWLEDGMENT

This material is based in part upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, under contract number DE-AC05-00OR22725, in part by the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory, managed by UT-Battelle, LLC, for the U. S. Department of Energy., and in part by an Air Force Research Laboratory Information Directorate grant (FA8750-16-1-0065).

REFERENCES

[1] W. Maass, "Networks of spiking neurons: the third generation of neural network models," *Neural networks*, vol. 10, no. 9, pp. 1659–1671, 1997.

[2] J. Cabessa and H. T. Siegelmann, "The super-turing computational power of plastic recurrent neural networks," *International journal of neural systems*, vol. 24, no. 08, p. 1450029, 2014.

[3] C. D. Schuman, T. E. Potok, R. M. Patton, J. D. Birdwell, M. E. Dean, G. S. Rose, and J. S. Plank, "A survey of neuromorphic computing and neural networks in hardware," *arXiv preprint arXiv:1705.06963*, 2017.

[4] Y. Yi, Y. Liao, B. Wang, X. Fu, F. Shen, H. Hou, and L. Liu, "Fpga based spike-time dependent encoder and reservoir design in neuromorphic computing processors," *Microprocessors and Microsystems*, vol. 46, pp. 175–183, 2016.

[5] S. M. Bohte, J. N. Kok, and H. La Poutre, "Error-backpropagation in temporally encoded networks of spiking neurons," *Neurocomputing*, vol. 48, no. 1-4, pp. 17–37, 2002.

[6] H. Mostafa, "Supervised learning based on temporal coding in spiking neural networks," *IEEE transactions on neural networks and learning systems*, vol. 29, no. 7, pp. 3227–3235, 2018.

[7] J. V. Arthur and K. Boahen, "Learning in silicon: Timing is everything," in *Advances in neural information processing systems*, 2006, pp. 75–82.

[8] D. Querlioz, O. Bichler, and C. Gamrat, "Simulation of a memristor-based spiking neural network immune to device variations," in *Neural Networks (IJCNN), The 2011 International Joint Conference on*. IEEE, 2011, pp. 1775–1781.

[9] C. Liu, B. Yan, C. Yang, L. Song, Z. Li, B. Liu, Y. Chen, H. Li, Q. Wu, and H. Jiang, "A spiking neuromorphic design with resistive crossbar," in *Design Automation Conference (DAC), 2015 52nd ACM/EDAC/IEEE*. IEEE, 2015, pp. 1–6.

[10] A. Yanguas-Gil, "Fast, smart neuromorphic sensors based on heterogeneous networks and mixed encodings," in *43rd Annual GOMACTech Conference*, Miami, March 2018.

[11] S. K. Esser, R. Appuswamy, P. Merolla, J. V. Arthur, and D. S. Modha, "Backpropagation for energy-efficient neuromorphic computing," in *Advances in Neural Information Processing Systems*, 2015, pp. 1117–1125.

[12] C. Posch, T. Serrano-Gotarredona, B. Linares-Barranco, and T. Delbruck, "Retinomorphic event-based vision sensors: bioinspired cameras with spiking output," *Proceedings of the IEEE*, vol. 102, no. 10, pp. 1470–1484, 2014.

[13] A. Vanarse, A. Osseiran, and A. Rassau, "A review of current neuromorphic approaches for vision, auditory, and olfactory sensors," *Frontiers in neuroscience*, vol. 10, p. 115, 2016.

[14] R. Gopalakrishnan, Y. Chua, and L. R. Iyer, "Classifying neuromorphic data using a deep learning framework for image classification," *arXiv preprint arXiv:1807.00578*, 2018.

[15] J. S. Plank, C. D. Schuman, G. Bruer, M. E. Dean, and G. S. Rose, "The TENNLab exploratory neuromorphic computing framework," http://neuromorphic.eecs.utk.edu/pages/research-publications/, August 2018.

[16] C. D. Schuman, J. S. Plank, A. Disney, and J. Reynolds, "An evolutionary optimization framework for neural networks and neuromorphic architectures," in *International Joint Conference on Neural Networks*, Vancouver, July 2016.

[17] J. S. Plank, G. S. Rose, M. E. Dean, C. D. Schuman, and N. C. Cady, "A unified hardware/software co-design framework for neuromorphic computing devices and applications," in *IEEE International Conference on Rebooting Computing (ICRC 2017)*, Washington, DC, November 2017.

[18] C. D. Schuman, J. S. Plank, A. Disney, and J. Reynolds, "An evolutionary optimization framework for neural networks and neuromorphic architectures," in *Neural Networks (IJCNN), 2016 International Joint Conference on*. IEEE, 2016, pp. 145–154.

[19] C. D. Schuman, J. D. Birdwell, and M. E. Dean, "Spatiotemporal classification using neuroscience-inspired dynamic architectures," *Procedia Computer Science*, vol. 41, pp. 89–97, 2014.

[20] J. P. Mitchell, M. E. Dean, G. R. Bruer, J. S. Plank, and G. S. Rose, "Danna 2: Dynamic adaptive neural network arrays," in *Proceedings of the International Conference on Neuromorphic Systems*. ACM, 2018, p. 10.

[21] G. Chakma, M. E. Dean, G. S. Rose, K. Beckmann, H. Manem, and N. Cady, "A hafnium-oxide memristive dynamic adaptive neural network array," in *International Workshop on Post-Moore's Era Supercomputing (PMES)*, Salt Lake City, UT, November 2016.

[22] J. J. M. Reynolds, J. S. Plank, C. D. Schuman, G. Bruer, A. W. Disney, M. Dean, and G. S. Rose, "A comparison of neuromorphic classification tasks," in *International Conference on Neuromorphic Computing Systems*. Knoxville, TN: ACM, July 2018.

[23] Y. LeCun, C. Cortes, and C. Burges, "Mnist handwritten digit database," *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, vol. 2, 2010.

[24] A. Wieland, "Evolving neural network controllers for unstable systems," in *Neural Networks, 1991., IJCNN-91-Seattle International Joint Conference on*, vol. 2. IEEE, 1991, pp. 667–673.

[25] F. Gomez, J. Schmidhuber, and R. Miikkulainen, "Efficient non-linear control through neuroevolution," in *European Conference on Machine Learning*. Springer, 2006, pp. 654–662.

[26] J. P. Mitchell, G. Bruer, M. E. Dean, J. S. Plank, G. S. Rose, and C. D. Schuman, "NeoN: Neuromorphic control for autonomous robotic navigation," in *IEEE 5th International Symposium on Robotics and Intelligent Sensors*, Ottawa, Canada, October 2017, pp. 136–142.

[27] F. Akopyan, J. Sawada, A. Cassidy, R. Alvarez-Icaza, J. Arthur, P. Merolla, N. Imam, Y. Nakamura, P. Datta, G.-J. Nam *et al.*, "Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 10, pp. 1537–1557, 2015.

[28] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.