From Memristors to Compositional Machine Learning

Exploring Neuromemristive Algorithmic Abstractions with the Knowm API

Alex Nugent
Knowm Inc
Santa Fe, New Mexico
alex@knowm.org

Tim Molter
Knowm Inc
Santa Fe, New Mexico
tim@knowm.org

ACM Reference Format:

Alex Nugent and Tim Molter. 2017. From Memristors to Compositional Machine Learning: Exploring Neuromemristive Algorithmic Abstractions with the Knowm API. In *NCS '17: Neuromorphic Computing Symposium, July 17–19, 2017, Knoxville, TN, USA*. ACM, New York, NY, USA, 8 pages. <https://doi.org/10.1145/3183584.3183615>

1 INTRODUCTION

Due to the recent congruence of big data and compute acceleration, the field of machine learning (ML) has made significant advancements, driven primarily by deep neural networks trained with the stochastic gradient descent algorithm (DNN). Error rates on challenging learning benchmarks have fallen precipitously. The algorithmic success of DNNs combined with the significant architectural mismatch between modern processors and neural architectures has resulted in a race to build DNN processors. The mismatch is demonstrated by analyzing the dynamic power consumption due to capacitive loss associated with synaptic access:

$$N \frac{\text{synapse}}{\text{brain}} \cdot b \frac{\text{bit}}{\text{synapse}} \cdot r \frac{\text{access}}{\text{sec}} \cdot \sigma d V^2 \frac{J}{\text{bit} \cdot \text{access}} = N b r \sigma d V^2 \frac{W}{\text{brain}} \quad (1)$$

Where σ is the linear metal wire capacitance, d is average signaling distance per bit access and V is the operating voltage. A cursory look at this equation reveals power consumption is minimized when synapses are stored with lower precision or analog ($b = 1$), synaptic activation is sparse, input activation is binary or spiking, operating voltage is low and average signaling distance small. None of these conditions match the operating regime of DNN on modern digital compute cores, where synaptic precision is high, input activation is continuous, synaptic access high, memory and processing are separated (and therefore d is high) and operating voltages are high. Optimization over these variables can reduce the dynamic power consumption by over nine orders of magnitude, as seen in Table 1.

A quick survey of the deep learning landscape in Table 2 reveals what end of the scale modern GPU computing falls, and how far we have to go before reaching the power and space efficiency of biology. While emerging deep learning processors claim gains of 10-100x over GPUs [Jouppi et al. 2017], these optimizations represent

b	r	d	V	Watts
64	100000	.1	3	86,400,000,000
32	10000	.01	2	192,00,00
16	1000	.001	1	240,000
8	100	.001	.5	3,000
4	10	.001	.25	37.5
1	1	.001	.1	.15

Table 1: Signaling power consumption for hypothetical synaptic processors as given by equation 1, where we have set $\sigma = 10\text{E-}10 \frac{F}{m}$ and $N = 1.5\text{E}14$

Paper	Year	Synapses	Power [Watt]	Brain-Scale Power [Watt]
<i>Building high-level features using large scale unsupervised learning.</i> [Le 2013]	2012	10E9	.16E6	2.4E9
<i>Imagenet classification with deep convolutional neural networks.</i> [Krizhevsky et al. 2012]	2012	60E6	788	1.9E9
<i>Deep image: Scaling up image recognition.</i> [Wu et al. 2015]	2015	213E6	53E3	3.8E10
Human Brain	NA	150E12	15	15

Table 2: Power consumption of deep learning systems extrapolated to human brain-scale ($N = 1.5\text{E}14$) under perfect scaling.

the first steps in a much larger algorithm-hardware co-optimization landscape.

Memristors offer a path for extreme power reduction by reducing memory-processor separation, either through layering of digital memory on top of processors or via direct analog synaptic emulation. In the latter approach, synaptic weights can be stored as nonvolatile conductances in dense arrays such as crossbars, while dot-product operations can be reduced to an analog summation-of-currents with low operating voltages. Challenges of this approach entail finding solutions for (a) reliable circuit operation in the face of fabrication and runtime variations and (b) achieving robust and algorithmically useful local learning in circuitry. While some groups ignore learning altogether and focus on inference-only processors

[Esser et al. 2013; Miyashita et al. 2016], this is a short-term solution. A learning processor does not imply a *continuously* learning processor, so arguments against online learning in mission-critical applications such as self-driving cars do not necessarily apply. Furthermore, given two synaptic processors with equivalent power and space metrics, where one processor can learn and the other cannot, the learning processor will prevail if the environment or goals change. This condition describes most if not all competitive environments. Additionally, recent attempts at analog neural network accelerators demonstrate that considerable chip real-estate is required for calibration to counteract circuit fabrication variations. [Miyashita et al. 2016] We point out that learning is calibration and that a solution to on-chip analog learning may also be a solution to robust self-calibrating analog programmable synaptic inference.

While robust on-chip analog memristive synaptic learning processors sound promising, it is a difficult technology to develop. Two major solution approaches can be summarized as "make useful digital algorithms analog" or "design useful analog-compatible algorithms". In the first approach, groups attempt to implement successful algorithms (primarily DNN trained with stochastic gradient descent) within the constraints of hardware which include limited weight precision, memristor decay, circuit noise, fabrication variations, and non-symmetric and/or stochastic conductance incrementation. [Bayat et al. 2016; Negrov et al. 2017] The second approach is to develop new or modify existing ML algorithms to map more naturally to the constraints of hardware. In this paper we report on some of our explorations of the second approach.

2 BACKGROUND

The kT-RAM technology specification [Nugent and Molter 2017] describes multiple levels of abstraction and specialization needed to implement an Anti-Hebbian and Hebbian (AHaH) computing [Nugent and W 2014] compliant synaptic processor and integrate it into digital computing systems. kT-RAM adds a general-purpose adaptive synaptic resource to existing digital computing platforms. Like transistors and logic gates, multiple levels of computational abstraction can be defined on top of kT-RAM's differential-memristor synaptic primitive. To explore these levels of abstraction we created the KnownAPI, a Java library built on top of a kT-RAM core emulator.

While ML and DNN are computationally challenging problems in themselves, the introduction of circuit simulation makes the problem intractable on a desktop computer. To address the computational intractability of exploring ML algorithms with a circuit simulator, we created a kT-RAM emulator with interchangeable core types. Each core type emulates synapses and circuit physics at various levels of fidelity. This strategy allows us to explore the space of kT-RAM instruction routines ("AHaH Algorithms") with efficient cores and, once candidate solutions are found, to replace them with more accurate cores that trade simulation efficiency with circuit realism. To better appreciate this approach, consider that a Spice-level circuit simulation of a 256×256 kT-RAM core on a desktop computer can take a day to complete, while the same simulation on an efficient circuit abstraction core takes less than a second.

Figure 1: A spike-encoding framework defines spike encoders (sensors), spike streams (wire bundles), spike channels (a wire), spike space (number of wires), spike patterns (active spike channels) and finally spikes (the state of being active).

2.1 Spike Coding

A collection of n synapses belong to a neuron (AHaH node), each with an associated weight: $\{w_0, w_1, \dots, w_n\}$. A subset of the synapses in a node can be co-activated by a spike pattern, which define integers in a spike space. A good way to picture it is as a big bundle of wires, where the total number of wires is the spike space and the set of wires active at any given time is the spike pattern. We call this bundle of wires and the information contained in it the 'spike stream'. The algorithms or hardware that convert data into a spiking representation are called spike encoders. A visual representation of this can be seen in Figure 1.

2.2 Memristor Types

Due to our focus on learning hardware, we have found it useful to categorize memristors into categories based on incrementation response:

Non-Polar: Application of both positive and negative voltage bias induces only increase or only decrease in conductance. Thermodynamic decay is used to change the conductance in the other direction. Examples include the dielectrophoretic aggregation of conductive nanoparticles in colloidal suspension [Wissner-Gross 2009].

Polar: Application of voltage bias enables incremental conductance in one direction, but all-or-nothing change in the opposite direction. An example of this includes phase-change memory [Burr et al. 2017].

Bi-Polar: Application of positive and negative voltage bias enables incremental conductance increase and decrease. An example of this include Self Directed Channel (SDC) memristors [Campbell 2016].

2.3 Linear Specialist Classifier

We have previously reported a simple routine of kT-RAM instructions for multiple-label classification with confidence estimation [Nugent and Molter 2017]. This classifier offers good general-purpose classification and is utilized in the work presented in this paper. To overcome loss of classification accuracy due to lower precision synapses, we created the Linear Specialist Classifier. A specialist classifier utilizes more than one classifier to provide better accuracy at the expense of more synapses. The operation utilizes a base or 'root' classifier to perform an initial classification. If the root classifier returns more than one possible label, indicating confusion, a new prediction is generated that is the average over multiple specialist classifiers, where each specialist is associated with the predicted labels from the root classifier. Given a predicted set of labels generated by the root classifier, the specialist classifiers focus learning on their assigned labels and the patterns that are mistaken for those labels. A specialist classifier could be constructed as a decision tree and extended to arbitrary depths. While what we show below contains one root node that branches to a number of specialists, each specialist could, in turn, have child specialists and so on.

3 COMPETITIVE GROWTH FRAMEWORK

Based on our work with the KnowmAPI, we present a series of abstraction layers that take us from low level memristor-based synaptic primitives to compositional supervised and unsupervised machine learning functions. We call this collection of abstractions the Competitive Growth Framework (CGF). The CGF points to a general methodology for learning across multiple compositional modules within a spiking communication framework. Unlike DNN, the CGF is more amenable to neuromorphic hardware implementation as it does not require the computation of derivatives, backward propagation of information, or high-precision synaptic weights¹.

3.1 Collections of Meta-Stable Switches Form Memristors

The term *memristor* encompasses a wide range of resistance-changing electronic devices and many devices have been reported in the literature.[Campbell 2016; Hasegawa et al. 2011; Jackson et al. 2013; Oblea et al. 2010; Valov and Kozicki 2013; Yang et al. 2012] Leon Chua, the memristor's theoretical inventor, has stated that "if its pinched it's a memristor" [Chua 2014]. This is in reference to the shape of the IV trace during application of a periodic signal. An IV trace with a hysteresis response that goes through the origin (ie, it's pinched) implies a memory response from non-energy storing mechanisms. All devices which demonstrate such a response are termed memristors. For our purposes of exploring machine learning function, we require memristor models that can be tuned to the large variety of behaviors observed in the literature and, importantly, capture non-ideal effects such as low precision and stochastic switching. We have achieved this with the generalized meta-stable switch model [Nugent and W 2014], which treats a memristor as a collection of meta-stable switches (Figure 2).

¹In one limit, where node in the decision tree are random hyperplanes, learning is purely a function of tree growth and synapses require zero weight precision.

Figure 2: The Metastable Switch model considers a memristor (C) as a collection of metastable switches (B). A metastable switch (A) is an idealized mechanism that switches probabilistically between two conduction states of differing conductances G_A and G_B

3.2 Competing Memristors Form Complex Weights

A differential-pair synapse representation of a synapse has a number of advantages over single memristor. While it is highly desirable for a memristive material to possess high-resolution non-volatile state retention, this is not generally the case in practice. Decay of state is typical, as is variation of device properties during fabrication and operation. Tolerance to decay and material variations can be achieved with a differential-pair representation. Rather than pinning logic levels to absolute conductances and relying on the intrinsic stability of the materials and repeatability of the fabrication process, a differential pair can create its own reference: zero is defined when the conductance of each memristor is equal. Through anti-Hebbian operations, the state of a differential-pair synapse can be driven toward zero, providing a local passive mechanism to adapt to fabrication variations. Since conductance decay is (to a first approximation) a common-mode signal, differential representation provides a degree of isolation. Finally, most ML models including DNN require the concept of multiplicative synaptic state, i.e. positive and negative synaptic weights. It is not physically possible to

Figure 3: We define four synapse configurations formed of differential pairs of bi-polar memristors.

have a negative conductance, which means that a single memristor cannot represent a negative weight without a reference. On the other hand, the conductance differential, $G_a - G_b$, naturally forms a multiplicative state and can be defined as positive when $G_a > G_b$ and negative when $G_a < G_b$.

There are four ways to construct differential pair-synapses with polar memristors, which we call the 1-2, 1-2i, 2-1, and 2-1i configurations (Figure 3). In our notation, the polarity of a memristor is defined such that the bar points to the lower potential end when the conductance of the device increases. For the remainder of this paper we will assume the 2-1 configuration, but it is straightforward to translate what we show to other configurations.

While differential pairs are necessary for implementation of multiplicative state they offer additional features not found in the mathematical abstractions of real-valued weights. For example, consider evidence accumulation and learning rate adaptation.

3.2.1 Evidence Accumulation. While the conductance differential encodes state, the conjugate $G_a + G_b$ can be used to encode evidence accumulation. Consider a case where the conductance of each memristor varies by integer multiples between 1 and 256 and is initialized into a low conducting state: $G_a = 1, G_b = 1$. Presume that the differential pair is used to measure the co-occurrence of events E_i with a target binary state A or B . We can associate each event with a synapse, $E_i = [G_a^i, G_b^i]$ and each memristor of the pair with the target binary state: $E_A \rightarrow G_a, E_B \rightarrow G_b$. For each occurrence of target event A , we increment G_a , and for each occurrence of target event B we increment G_b . At some time in the future, after accumulating evidence over multiple events, the value of the differential pair synapses are read and the output voltage V_y is given by:

$$V_y = \frac{\sum_i G_A^i - G_B^i}{\sum_i G_A^i + G_B^i} \quad (2)$$

Let us presume event E_1 occurred 150 times and that $E_1 = [100, 50]$ while event E_2 occurred only 3 times and that $E_2 = [1, 2]$. While each synapse has measured equal and opposite probabilities or weighting factors, the synapse for E_1 has gathered much more evidence. When accessed independently, each synapse will return a value reflective of its equal and opposite probability:

$$V_y^A = \frac{100 - 50}{100 + 50} = 1/3, V_y^B = \frac{1 - 2}{1 + 2} = -1/3 \quad (3)$$

On the other hand, when both synapses are accessed together, the output is dominated by the evidence gathered by E_1 :

$$V_y^{A+B} = \frac{50 - 1}{150 + 3} = \frac{49}{153} \approx 1/3, \quad (4)$$

3.2.2 Learning Rate Adaptation. A common practice in learning algorithms concerns decreasing the adaptation rate of weights over time, a process that improves learning by enabling large changes in the beginning of the learning process that reduce over time until the final modifications are small refinements. Adaptive learning rates can be achieved with a differential synapse representation through its initialization state. As the conjugate grows and evidence accumulates, the same change in the differential causes smaller changes in the synaptic output. The result is that memristor-pairs initialized into low conductive states will undergo higher adaption that reduces over time as the conjugate grows, while the opposite occurs for memristor pairs initialized into high conductive states.

3.3 Collections of Synapses form AHaH nodes

An AHaH node is formed when a collection of differential-pair synapses are coupled together and provides a simple but computationally universal synaptic adaptation and inference resource. Thermodynamic RAM is a proposed synaptic processor that borrows from existing RAM architecture to integrate adaptive synaptic processing into existing digital computing platforms. This utilization of a spike code allows for the direct coupling of a spike pattern to the activation of synaptic cells in a kT-RAM core. That is, spikes in a spike space are directly mapped to active synapses in a synaptic memory space (a kT-RAM Core) via row and column addressing. Digital instructions provided to the kT-RAM core result in defined voltage drive patterns which result in state extraction and synaptic modification. Some instructions result in Anti-Hebbian (AH) modification to the synapses while other instructions result in Hebbian (H) modification, hence the name AHaH Nodes.² Exploring how instructions can be used to achieve useful ML and other functions has been our primary focus. Through strategic application of instructions, control of synaptic adaptation and implementation of many types of learning algorithms are possible. Stated differently, the rules which govern synaptic adaption are not hard-wired in AHaH nodes but rather emerge from routines of instructions and the space of possible routines is largely unexplored.

²Some instructions are neither Hebbian or anti-Hebbian

3.4 Competing Collections of AHaH Nodes form Autoencoders

The ability to form unsupervised representations of data is given many names in various contexts such as clustering, sparse coding, and independent component analysis. In some cases the output of the process is integer identifiers (clustering) and in other cases it is a weighted basis representation (sparse coding). As we have reported elsewhere [Nugent and W 2014], the binary encoding of a collection of AHaH nodes operating the FF-RU instruction can be used as the basis of a clustering algorithm. We have found that this method is susceptible to the null state³ for non temporally and spatially sparse inputs. Here we present a solution to that problem.

When a neuron's decision boundary falls between opposing data distributions, Hebbian learning acts to maximize the decision margin between the distributions, provided the Hebbian influence falls as the neural activation increases. While one data distribution acts to push the decision boundary away from it, the opposing distribution will push back. The result is an attractor state where the decision boundary splits the opposing distributions cleanly, forming a natural boundary that increases noise-tolerance and coincides with statistical techniques such as ICA and support vector maximization between pattern classes [Nugent and W 2014]. The risk of Hebbian learning occurs when there is no opposing data distribution, which can occur for a number of reasons including weight initialization or temporary data fluctuations. The positive feedback inherent in Hebbian learning will then force the AHaH node to the null state. Anti-Hebbian learning, on the other hand, will force the decision boundary to bifurcate its phase space and avoids the null state. However, a node operating pure anti-Hebbian learning cannot maximize its decision margin. What is required is the best of both worlds: a decision boundary that seeks to bifurcate its phase space while simultaneously maximizing its decision margin.

This ideal decision boundary can be accomplished by gating the application of AHaH plasticity to individual nodes based on information in the group. A competitive auto-encoder is formed of two or more AHaH Nodes that compete for Hebbian feedback. On each evaluation cycle, a spike pattern is loaded and a read-instruction is executed. All AHaH nodes that exceed a threshold (usually set at zero) are used as the encoder output, thus forming a weighted basis output. The AHaH Node with the highest activation is considered the winner and is incremented high (FF-RH instruction) while all nodes that are above threshold but not the winner are incremented low (FF-RL instruction), provided the winner ID is not in a set. When the set is filled, such that all nodes have had a chance to receive Hebbian feedback, or when a set time period has elapsed, the set is cleared of all IDs. While competition for Hebbian feedback is key, a mechanism must be put in place to prevent the rich-get-richer cycle that leads to the null state. This is summarized in Listing 1 below:

Listing 1: Basis encoding with AHaH node collections

```
if (set.size() >= numNodes || count == limit) {
    set.clear();
```

³The null state occurs when all synapses acquire the same multiplicative state and result in the same state output for all input patterns

```
count=0;
}
//nodes is a sorted list of AHaH nodes.
int winner=nodes.get(0).id;
if (!set.contains(winner)) {
    for (int i=0; i<nodes.size(); i++) {
        if (i==0) {
            ktram.execute(nodes.get(i).id, FF, RH); // reward
        } else if (nodes.get(i).value > 0) {
            ktram.execute(nodes.get(i).id, FF, RL); // punish
        } else {
            break;
        }
    }
    set.add(winner);
}
count++;
}
```

The use of a set insures that all AHaH nodes receive the same total amount of Hebbian (positive) feedback, thus preventing the run-away rich-get-richer scenario that leads to the null state. The use of a counter and time limit prevents the case where the set is never filled, for example if the number of AHaH nodes exceeds the number of data regularities. In the lower limit where the number of AHaH nodes is two, we can replace the set with two bits and forgo the counter, thus achieving a simplified routine. We call the result a BinaryPartitioner (Listing 2), which can be seen as an improved version of a single AHaH node operating the FF-RU instruction.

Listing 2: BinaryPartitioner encodeAdapt Method

```
public boolean encodeAdapt(int[] spikes) {
    //encode-->
    //load spike pattern for node A
    ktram.setSpikes(aID, spikes);
    //FFLV-FRLV instructions for low voltage read
    // and minimal memristor disturbance and power
    float y_a = ktram.execute(aID, FFLV, RFLV);
    ktram.setSpikes(bID, spikes);
    float y_b = ktram.execute(bID, FFLV, RFLV);
    boolean y = y_a > y_b;

    //adapt-->
    if (y && !a) {
        ktram.execute(aID, FF, RH); // reward
        if (y_b > 0) {
            ktram.execute(bID, FF, RL); // punish
        }
        a = true;
    } else if (!y && !b) {
        ktram.execute(bID, FF, RH); // reward
        if (y_a > 0) {
            ktram.execute(aID, FF, RL); // punish
        }
    }
}
```

```

    }
    b = true;
}
if (a && b) { // reset
    a=b=false;
}

return y;
}

```

Collections of BinaryPartitioners can be used to increase the number of cluster centers, for example with the use of a decision tree. In this case, the spike output of the encoder is given as the binary address of the leaf node, as seen in Listing 3. Additionally, while a decision tree can produce excellent clusters, its synaptic utilization is exponential in the tree depth (2^D). As an alternative topology and at the cost of clustering performance, partitioners can instead be shared across a depth in the tree.

Listing 3: BinaryPartitionerTree

```

public class BinaryPartitionerTree {

    // variable tree depths
    private int depth=8;
    private BinaryPartitioner[] nodes;

    public BinaryPartitionerTree() {
        nodes = new BinaryPartitioner[(int) Math.pow(2,depth)];
        for (int i = 0; i < nodes.length; i++) {
            nodes[i] = new BinaryCompetitivePartitioner(i);
        }
    }

    public int encodeAdapt(int[] spikes) {
        int path = 0;
        int idx = 1;
        for (int i = 0; i < depth; i++) {
            int nodeIndex = (path | idx);
            if (nodes[nodeIndex].encodeAdapt(spikes)) {
                path = path | idx;
            }
            idx = idx << 1;
        }
        return path;
    }
}

```

3.5 Growth Forests and Error Reflection

A decision tree (DT) forms a compressed representation of data. The path up the tree constitutes the tree's output and each node on the path is a binary question asked of the data. As a tree grows, its encoding becomes more fine-grained.

Distributed representations are important in compositional learning because they provide the ability to generalize on unseen data [Liaw et al. 2002]. In the case of a single decision tree, it is clear that its output is not a distributed representation. However, as Bengio pointed out [Bengio et al. 2010], if we consider the output of a tree to be an integer specifying the DT leaf or path, then we can consider the output of a decision tree forest (DTF) as the encoding of a vector whose elements are these integers, one per tree in the forest. This is a distributed representation, which can express a number of configurations possibly exponential in the number of trees. This is supported by the ability of DTF's to generalize well to unseen data while individual DT do not [Ho 1995].

Consider model M1 formed of a DTF providing a spike encoding of pattern P to supervised linear classifier C. Classifier C compares output Y with a supervised input Y' and, if they differ, marks pattern P as a *mistake*. In response to the mistake, model M1 reprocesses pattern P one or more times, a process we call *reflection*. If each tree in the DTF grows in response to node activations, then the DTF will add resolution to resolve finer details in the phase space region where mistakes occur. Of primary interest to us is that a purely local growth process⁴ combined with feed-forward communication can be used to reduce error of a classification. Mistake reflection and tree growth hints at a general learning approach that does not rely on computing derivatives of continuous functions or back-propagating error signals.

3.6 Compositional Growth Forests

The ability to learn a compositional representation is the underlying objective of deep learning and is responsible for its success in processing typically hierarchical data such as vision, speech, and language. It is therefore tempting to investigate if mistake reflection can be made to work across multiple compositional layers of decision forests.

Consider model M2 composed of two decision trees, DT1 and DT2, where the output spike encoding P of DT1 is given as the input to DT2, which in turn produces its own spike encoding Q that is passed to a classifier C. We ask if the compositional decision trees can 'grow into error' as a single DT can. What happens when DT1 grows such that path P is modified, $P \rightarrow PP'$ to include an additional node or branch? How does DT2 process this additional resolution?

If DT1 adds one or more nodes to its encoding, all nodes of DT2 will see a different spike pattern. Modification of the input pattern $P \rightarrow PP'$ may therefore change the binary evaluation of any node along path Q, including the root node. Consequently, the addition of only one new branch in DT1 could completely alter the spike encoding Q, which will render any classification by C meaningless. In other words, the growth of upstream trees results in catastrophic loss of information for downstream trees. How can DT grow across multiple levels of compositions without causing potentially catastrophic loss of information for downstream DTs? One solution to this problem is to define a protocol for spike encoding and tree growth, which we call the Growth Communication Protocol (GCP):

⁴The growth of tree structures is a defining characteristic of natural self-organization, so we find this learning process inherently appealing as a candidate for self-configuring hardware.

- The collection of node identifiers along a DT evaluation *path* constitute the trees output spike encoding.
- Node identifiers increment as new nodes are created within a tree.⁵
- Nodes in a DT must ignore all spike channels not yet active at the time of its fixation.
- Nodes that spawn child nodes must also fixate.

Each node in a DT is allowed some time after its creation to adapt its internal configurations in reaction to the information carried over the patterns it is processing⁶, for example by operating the AHaH Algorithm specified by the BinaryPartitionerTree of Listing 3. At the point of fixation, one of two conditions must occur. (1) The node must halt all internal modifications by relying on intrinsic stability (non-volatility) of its components or (2) its internal state must be repaired through some active mechanism, for example unsupervised AHaH plasticity operating on structured information. To *fixate* thus refers to the moment when a DT node transitions from a state of change to a state of stasis, irrespective of the mechanisms that enforce the stasis.

If all trees throughout all compositional layers obey the GCP then additional resolution provided by growth in an upstream trees is processed by node growth in down-stream trees and growth becomes coordinated across layers. Through mistake reflection, growth is focused in regions of the phase-space causing classification errors.

4 EXPERIMENTAL RESULTS

The ability to autoencode data is useful for dimensionality reduction, noise tolerance and generalization in classification tasks and to automatically discover data regularities in unsupervised clustering tasks. To demonstrate this, we applied the Binary Partitioner Tree AHaH routine to MNIST and CIFAR images as shown in Figure 4 and Figure 5 respectively.

The growth communication protocol was evaluated on a multilayer convolutional network composed of one or more stages of convolutional growth trees followed by a trimmer⁷, pooling⁸ and linear classification operation. MNIST images were spike-encoded by thresholding pixel values with a threshold of 10^{-9} . Growth trees were composed of nodes operating the BinaryPartitionerTree routine with a depth of 4, leading to a branching factor of 16. Each node fixated after 15,000 (patch) evaluations. Convolutional feature filters were of size 4×4 and pooling filters were size 8×8 . Meta-parameters were not optimized and no attempts at regularization, such as introduction of noise or dropout of synapses, was attempted. Four epochs over the full MNIST training set were used for training.

⁵For example, if a DT of size 8 adds a new node, it should be given the id of 9. Nodes with lower IDs are thus older than nodes with higher IDs.

⁶While synaptic adaptation is preferable, it is not strictly required. Alternatively, node synapses can be initialized randomly and fixed immediately. In this case, learning is strictly a function of tree growth and not synaptic adaption. This has relevance to analog hardware implementations.

⁷A trimmer operation consists of dropping (trimming) spikes with low spike IDs, keeping the top N. In this case, for $N=1$, a spike pixel with spike pattern $[1, 2, 3, 4]$ is reduced to $[4]$, $TRM([1, 2, 3, 4]) = [4]$.

⁸A pooling operation removes duplicate spikes from two or more spike streams when joined. For example the spike patterns $[1, 2, 3]$ and $[2, 3, 4]$ become $[1, 2, 3, 4]$ when pooled.

⁹CIFAR Images were spike encoded by forming a spike from the three most significant bits of the red, blue and green channels and joining them together, leading to a spike image where each pixel was represented with 3 spikes in a spike space of 24.

Figure 4: Example of clustered MNIST images each assigned the same spike ID via the Binary Partitioner Tree

Figure 5: Example of clustered CIFAR images each assigned the same spike ID via the Binary Partitioner Tree.

When standard training was employed, training patterns without distortions were used in original order from the full 60,000 training set. When reflective training was employed, mistakes were accumulated over batches of 6000 without growth and then reflected for 5 cycles with growth. For reflective training, one epoch of standard training was followed by three epochs of mistake reflection.

Model	Train Type	Train	Test
Linear Classifier (LC)	Standard	83.3	83.8
LC	Reflective	85.6	86.3
Linear Specialist Classifier (LSC)	Standard	90.5	90.7
LSC	Reflective	92	91.9
$CNV(4 \times 4) \rightarrow TRM \rightarrow POOL(8 \times 8) \rightarrow LSC$	Standard	97.8	97.5
$CNV(4 \times 4) \rightarrow CNV(4 \times 4) \rightarrow TRM \rightarrow POOL(8 \times 8) \rightarrow LSC$	Standard	98.9	98.2
$CNV(4 \times 4) \rightarrow CNV(4 \times 4) \rightarrow CNV(4 \times 4) \rightarrow TRM \rightarrow POOL(8 \times 8) \rightarrow LSC$	Standard	98.6	97.7
$CNV(4 \times 4) \rightarrow TRM \rightarrow POOL(8 \times 8) \rightarrow LSC$	Reflective	99	97.9
$CNV(4 \times 4) \rightarrow CNV(4 \times 4) \rightarrow TRM \rightarrow POOL(8 \times 8) \rightarrow LSC$	Reflective	99.6	98.1
$CNV(4 \times 4) \rightarrow CNV(4 \times 4) \rightarrow CNV(4 \times 4) \rightarrow TRM \rightarrow POOL(8 \times 8) \rightarrow LSC$	Reflective	99.7	97.5

Table 3: Comparisons of various AHaH models on MNIST classification benchmark with standard and reflective training. CNV is a convolutional patch operator.

Our results demonstrate that the GCP is able to coordinate growth across compositions of decision trees and that reflective mistake learning is able to focus growth to regions of error. The later is evidenced by lower training error for reflective learning when compared to standard training for growth trees of the same size and growth rates.

5 CONCLUSION

We have presented a compositional learning framework based on the layering of growth trees that obey a growth communication protocol (GCP). We demonstrated the GCP with our kT-RAM emulation and benchmarking platform known as the Knowm API. We have presented a framework that extends from metastable switches to memristors to kT-synapses to AHaH nodes to kT-cores to AHaH routines. Our recent results demonstrate the GCP can coordinate growth across multiple layers of decision trees and that mistake reflection can be used to focus growth on error-generating patterns. This proposed solution to compositional machine learning is agnostic to both the hardware methodology used to implement it, as well as the local decision processes that powers nodes in the decision tree forest (DTF). This has implications in making a transition from existing digital CMOS technology to alternate technologies like memristors, as numerous hardware embodiments could be created that would implement the growth communication protocol. The motivations for building such learning hardware is to match primary metrics on state of the art ML algorithms (performance, accuracy, error rate, etc.) while providing superior secondary performance metrics (speed, power, size, etc.). Future work aims to demonstrate the GCP across larger networks and benchmarks.

REFERENCES

- F Merrikh Bayat, M Prezioso, B Chakrabarti, I Kataeva, and DB Strukov. 2016. Advancing memristive analog neuromorphic networks: increasing complexity, and coping with imperfect hardware components. *arXiv preprint arXiv:1611.04465* (2016).
- Yoshua Bengio, Olivier Delalleau, and Clarence Simard. 2010. Decision trees do not generalize to new variations. *Computational Intelligence* 26, 4 (2010), 449–467.
- Geoffrey W Burr, Robert M Shelby, Abu Sebastian, Sangbum Kim, Seyoung Kim, Severin Sidler, Kumar Virwani, Masatoshi Ishii, Pritish Narayanan, Alessandro Fumarola, et al. 2017. Neuromorphic computing using non-volatile memory. *Advances in Physics: X* 2, 1 (2017), 89–124.
- Kristy A Campbell. 2016. Self-Directed Channel Memristor for High Temperature Operation. *arXiv preprint arXiv:1608.05357* (2016).
- Leon Chua. 2014. If it's pinched it's a memristor. *Semiconductor Science and Technology* 29, 10 (2014), 104001.

- Steve K Esser, Alexander Andreopoulos, Rathinakumar Appuswamy, Pallab Datta, Davis Barch, Arnon Amir, John Arthur, Andrew Cassidy, Myron Flickner, Paul Merolla, et al. 2013. Cognitive computing systems: Algorithms and applications for networks of neurosynaptic cores. In *Neural Networks (IJCNN), The 2013 International Joint Conference on*. IEEE, 1–10.
- Tsuyoshi Hasegawa, Alpna Nayak, Takeo Ohno, Kazuya Terabe, Tohru Tsuruoka, James K Gimzewski, and Masakazu Aono. 2011. Memristive operations demonstrated by gap-type atomic switches. *Applied Physics A* 102, 4 (2011), 811–815.
- Tin Kam Ho. 1995. Random decision forests. In *Document Analysis and Recognition, 1995., Proceedings of the Third International Conference on*, Vol. 1. IEEE, 278–282.
- Bryan L. Jackson, Bipin Rajendran, Gregory S. Corrado, Matthew Breitwisch, Geoffrey W. Burr, Roger Cheek, Kailash Gopalakrishnan, Simone Raoux, Charles T. Rettner, Alvaro Padilla, Alex G. Schrott, Rohit S. Shenoy, Bülent N. Kurdi, Chung H. Lam, and Dharmendra S. Modha. 2013. Nanoscale Electronic Synapses Using Phase Change Devices. *J. Emerg. Technol. Comput. Syst.* 9, 2, Article 12 (May 2013), 20 pages.
- Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. 2017. In-datacenter performance analysis of a tensor processing unit. *arXiv preprint arXiv:1704.04760* (2017).
- Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.
- Quoc V Le. 2013. Building high-level features using large scale unsupervised learning. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE, 8595–8598.
- Andy Liaw, Matthew Wiener, et al. 2002. Classification and regression by randomForest. *R news* 2, 3 (2002), 18–22.
- Daisuke Miyashita, Shouhei Kousai, Tomoya Suzuki, and Jun Deguchi. 2016. Time-domain neural network: A 48.5 TSP/s/W neuromorphic chip optimized for deep learning and CMOS technology. In *Solid-State Circuits Conference (A-SSCC), 2016 IEEE Asian*. IEEE, 25–28.
- D Negrov, I Karandashev, V Shakirov, Yu Matveyev, W Dunin-Barkowski, and A Zenkevich. 2017. An approximate backpropagation learning rule for memristor based neural networks using synaptic plasticity. *Neurocomputing* 237 (2017), 193–199.
- M Alexander Nugent and Timothy W Molter. 2017. Thermodynamic-RAM technology stack. *International Journal of Parallel, Emergent and Distributed Systems* (2017), 1–14.
- M A Nugent and Molter T W. 2014. AHaH Computing - From Metastable Switches to Attractors to Machine Learning. *PLoS ONE* 9 (02 2014), e85175.
- Antonio S Oblea, Achyut Timilsina, David Moore, and Kristy A Campbell. 2010. Silver chalcogenide based memristor devices. In *Proc. 2010 IEEE International Joint Conference on Neural Networks (IJCNN)*. IEEE, 1–3.
- Ilia Valov and Michael N Kozicki. 2013. Cation-based resistance change memory. *Journal of Physics D: Applied Physics* 46, 7 (2013), 074005.
- A D Wissner-Gross. 2009. Dielectrophoretic Architectures. (2009), 155–173.
- Ren Wu, Shengen Yan, Yi Shan, Qingqing Dang, and Gang Sun. 2015. Deep image: Scaling up image recognition. *arXiv preprint arXiv:1501.02876* 7, 8 (2015).
- Yuchao Yang, Patrick Sheridan, and Wei Lu. 2012. Complementary resistive switching in tantalum oxide-based resistive memory devices. *Applied Physics Letters* 100, 20 (2012), 203112.