A Unified Learning Framework for Memristive Neuromorphic Hardware

Anatoli Gorchetchnikov, Massimiliano Versace, Heather Ames, Ben Chandler,
Jasmin Léveillé, Gennady Livitz, and Ennio Mingolla, *Boston University*
Greg Snider, Rick Amerson, Dick Carter, Hisham Abdalla, and Muhammad Shakeel Qureshi
Hewlett-Packard Laboratories

Abstract— **Realizing adaptive brain functions subserving perception, cognition, and motor behavior on biological temporal and spatial scales remains out of reach for even the fastest computers. Newly introduced memristive hardware approaches open the opportunity to implement dense, low-power synaptic memories of up to 10^{15} bits per square centimeter. Memristors have the unique property of "remembering" the past history of their stimulation in their resistive state and do not require power to maintain their memory, making them ideal candidates to implement large arrays of plastic synapses supporting learning in neural models. Over the past decades, many learning laws have been proposed in the literature to explain how neural activity shapes synaptic connections to support adaptive behavior. To ensure an optimal implementation of a large variety of learning laws in hardware, some general and easily parameterized form of learning law must be designed. This general form learning equation would allow instantiation of multiple learning laws through different parameterizations, without rewiring the hardware. This paper characterizes a subset of local learning laws amenable to implementation in memristive hardware. The analyzed laws belong to four broad classes: Hebb rule derivatives with various methods for gating learning and decay; Threshold rule variations including the covariance and BCM families; Input reconstruction-based learning rules; and Explicit temporal trace-based rules.**

I. Introduction

IMPLEMENTING brain functions enabling perception, cognition, and learning on biological temporal and spatial scales, while easily achieved by mammalian nervous systems, remains unfeasible for current computational architectures. In the past few years work on memristive devices has brought modelers closer to realizing digital brain architectures that can interact with the world in real time while learning [18][19][21]. The mathematical definition of learning equations that are compatible with the newly designed hardware is a major outstanding issue.

One such example of co-development of adaptive hardware and software for memristive devices is the collaboration between Hewlett-Packard and Boston University. The jointly developed Cog Ex Machina (Cog) software architecture is targeted for a variety of hardware platforms including Dendra chips, which are massively parallel processors characterized by dense, memristive memories integrated with conventional circuitry for inference and learning, such that memory and computational circuits are closer together. Cog represents a computation as a directed graph where nodes hold computational state and exchange information across edges. Edges relay information from one node to another using a multidimensional array of tensors (tensor field). Cog implements plasticity in adaptive transformations that use incoming tensor fields to produce output tensor fields on outgoing edges [19].

Plasticity rules implemented in adaptive transformations should be as flexible as possible to accommodate a wide variety of existing learning laws and at the same time should be as simple as possible to reduce the computational cost of learning. One step toward accomplishing this goal is to analyze a large sample of candidate learning laws and recast them as instantiations of one general model.

Existing learning laws can be roughly categorized in the following classes: Basic Hebb rule derivatives; Threshold-based rules; Input reconstruction-based rules; and Explicit temporal trace-based rules. Most of the cases discussed in this paper only require a two layer network, while for networks containing more than two layers, learning proceeds independently between each pair of consecutive layers. A notable exception to this is the back-propagation rule, where the learning in each weight matrix depends on an error term propagated all the way from the top-most layer to the bottom-most layer. A network design capable of implementing this kind of propagation will be introduced in the corresponding section. For all other sections we use the network depicted in Figure 1a.

Here x_i represents the activity of presynaptic neuron i (or vector x represents the activity of all presynaptic neurons); y_j represents the activity of postsynaptic neuron j (y in vector notation), and w_{ij} represents the synaptic weight from i to j (w is the matrix of all weights). The postsynaptic activity y can be calculated via a differential equation:

This work was partially funded by the DARPA SyNAPSE program, contract HR0011-09-3-0001. The views, opinions, and/or findings contained in this article are those of the authors and should not be interpreted as representing the official views or policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the Department of Defense. Massimiliano Versace, Ennio Mingolla, Heather Ames, and Ben Chandler were also supported in part by CELEST, a National Science Foundation Science of Learning Center (NSF SBE-0354378 and NSF OMA-0835976).

A.G., M.V., H. Ames, B.C., J.L., G.L., and E.M. are with the Department of Cognitive and Neural Systems and Center of Excellence for Learning in Education, Science, and Technology, Boston University, 677 Beacon Street, Boston, MA 02215, USA (e-mail: {anatoli, versace, starfly, bchandle, jasminl, glivitz, ennio}@cns.bu.edu)

G.S., R.A., D.C., H. Abdalla., and M.S.Q. are with Hewlett-Packard Laboratories, 1501 Page Mill Rd., Palo Alto, CA 94304, USA (e-mail: {snider.greg, frederic.amerson, richard.carter, hisham.abdalla, shakeel.qureshi}@hp.com)

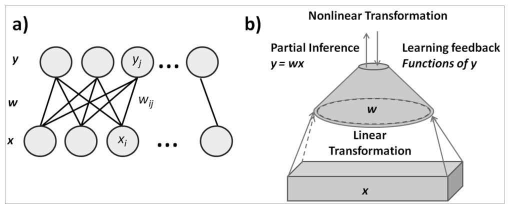

Fig. 1. (a) A simple schematic of network architecture, with presynaptic neurons x feeding (b) learning in memristive devices. Linear and nonlinear transformations cooperate to implement the learning that corresponds to long-term memory. Cog combines the state of a linear transformation, w, with the input, x_i, to produce y_j, a partial inference. It then combines $f(y_j)$, x_i, and w_{ij} to update w and implement learning. The learning feedback consists of a function of y that determines the type of learning that should occur.

$$\frac{dy_j}{dt} = \phi(y_j)\sum_{i=1}^{N} x_i w_{ij} + \psi(y_j) \tag{1}$$

where $\phi()$ represents the resulting activation from the current input, and $\psi()$ represents the internal dynamics of activation. Another way to calculate the activation is through an algebraic equation that resembles an equilibrium solution of equation (1):

$$y_j = \phi(\sum_{i=1}^{N} x_i w_{ij}) \tag{2}$$

Some rules were designed to work with the differential form of the postsynaptic activation while others were set to work with the algebraic form. We will refer to these forms or equations in the sections for corresponding rules. In the following sections, we analyze each learning rule class and discuss its implementation in Cog (Figure 1b).

II. Basic Hebb Rule Derivatives

The premise of the simple mathematical interpretation of the Hebb rule [9] is that there is a direct dependency between synaptic plasticity and repeated and persistent stimulation of a postsynaptic cell by a presynaptic cell, which is usually expressed as a correlation between weight change and the product of presynaptic and postsynaptic activities. All of the rules discussed in this section are based on this fundamental rule and are widely applied in a class of continuous firing dynamical models expressed in differential form; equation (1). A general form for Hebbian rules is:

$$\frac{dw_{ij}}{dt} = \lambda x_i y_j - f(x_i, y_j) w_{ij} \tag{3}$$

where λ is the learning rate and $f()$ is a gating function. In this equation the first term is the Hebbian or correlation term and the second term is a decay term that helps to normalize the weights and sets the equilibrium limits for the weight values. In spiking networks, the Hebb-based Spike Timing Dependent Plasticity (STDP) [14][7] requires a slightly different form similar to equation (3):

$$\frac{dw_{ij}}{dt} = \lambda(x_i y_j - w_{ij}) f(x_i, y_j) \tag{4}$$

Here the gating function $f()$ is applied not only to the decay, but also to the learning process. As a result, the equilibrium value of the weight always follows the Hebbian correlation, but the dynamics of this process are governed by the function $f()$. Both forms can be combined as:

$$\frac{dw_{ij}}{dt} = \lambda(x_i y_j - g(x_i, y_j) w_{ij}) h(x_i, y_j) \tag{5}$$

Hardware constraints put additional limitations on the shapes of the functions $g()$ and $h()$. Since these functions are dependent on the value of y_j only, they have to be computed in the nodes. At the same time, x_i is not available at this level unless it is transmitted from the pre-synaptic node directly to the post-synaptic node, effectively bypassing the linear transformation. If the functions $g()$ and $h()$ are only functions of y_j, the data transfer fits into the one depicted in Figure 1. Our analysis shows that all the rules in the Hebbian family have separable components of these functions that depend on x_i and y_j independently. Furthermore, the component that depends on x_i is linear in all cases and an integral power of x_i in the most general case. Thus, equation (4) can be recast as:

$$\frac{dw_{ij}}{dt} = \lambda(x_i f(y_j) - (\beta_1 x_i^n + g(y_j)) w_{ij})(\beta_2 x_i^m + h(y_j)) \tag{6}$$

where β_1 and β_2 are coefficients. Note that the y_j in the Hebbian term was replaced with a function of y_j, which will be convenient for some of the rules. All functions $f()$, $g()$, and $h()$ now only depend on y_j and not x_i. The recasting of single-equation Hebb rule derivatives are summarized in Tables 1 and 2 where Table 2 indicates the rules that are used for STDP.

For all cases of the single-equation Hebb rule derivatives, parameters n and m in equation (6) are 1, and the most complex function of y_j is cubic. The general form equation (6) is able to accommodate all rules listed in Tables 1 and 2 without requiring an expensive data transfer of x_i.

III. Threshold-based Rules

The primary difference between threshold-based rules and Hebbian rules is the introduction of a threshold in considering the influence of cell activities on learning. While the Hebbian correlation term is positive and leads to an increase in weight whenever both presynaptic and postsynaptic activities are positive, this is not always the case for threshold rules. These rules are set up such that if the activity of the cell is higher than a certain threshold the weight increases, but if it is lower than the threshold the weight decreases. Effectively, the Hebbian weight change is proportional to $x_i y_j$, while threshold-based weight change is proportional to $x_i(y_j - \theta)$.

The equations summarized in this section include covariance laws [3] and multiple variations of the BCM law (e.g.

[1]). These learning laws are commonly used in networks with unit activations governed by equation (2), together with a second equation governing the dynamics of the threshold $\theta_{x,y}$:

$$\frac{d\theta_{x,y}}{dt} = \psi(\theta_{x,y}, x_i, y_j) \tag{7}$$

which is used for learning only. As a result, equation (7) is usually considered a part of the learning rule rather than being a component of cell activation. Learning laws based on such thresholds are then formulated as a system of two equations, one with faster dynamics (threshold in equation (7)), and another with slower dynamics (synaptic weight change). Technically, one can consider, and it is discussed in some variations of the BCM rule, that equation (2) is a limit case of activation with very fast dynamics. Note that while these rules do not seem to require partial inference as output, it might be reasonable to use a running average-based threshold as cell output.

The form of equation (7) in the rules below is quite close to equation (1) if the partial inference is substituted for y and the variable names are changed accordingly. One difference comes from BCM family where partial inference is squared (and sometimes filtered) to calculate the threshold. Another more important difference is that both partial inference itself and a running average of it are used to calculate the weights. The latter prevents a direct merging of this family with the Hebbian family. A general form for the threshold rules is:

$$\frac{dw_{ij}}{dt} = \lambda(x_i f(y_j) - g(x_i, y_j)\theta_{x_i y_j})h(y_j, \theta_{x_i y_j}) - \beta_3 w_{ij} \tag{8}$$

Once again, separating x_i and y_j in the function $g()$ to reduce the data transmission concerns leads to:

$$\frac{dw_{ij}}{dt} = \lambda(x_i f(y_j) - (\beta_1 x_i^n + g(y_j))\theta_{x_i y_j})h(y_j, \theta_{x_i y_j}) - \beta_3 w_{ij} \tag{9}$$

In equation (9), the function $h()$ contains the term $\theta_{x_i y_j}$, which is dependent on x_i. As the analysis shows, this dependency on x_i is not included in any rules of this family. Table 3 presents the summary of the threshold rules recast in terms of the equation (9).

Comparing equations (9) and (5), it is possible to derive a general formulation that includes both families of rules:

$$\begin{aligned} \frac{dw_{ij}}{dt} = \lambda(x_i f(y_j) - (\beta_1 x_i^n + g(y_j))(a w_{ij} + b\theta_{x_i y_j})) \\ (\beta_2 x_i^m + h(y_j, \theta_{x_i y_j})) - \beta_3 w_{ij} \end{aligned} \tag{10}$$

where a and b so far are set so that only one of the two is non-zero. If we exclude the STDP family of learning laws and only consider non-spiking rules, the equation can be reduced to:

$$\begin{aligned} \frac{dw_{ij}}{dt} = \lambda(x_i f(y_j) - (\beta_1 x_i^n + g(y_j))(a w_{ij} + b\theta_{x_i y_j})) \\ h(y_j, \theta_{x_i y_j}) - \beta_3 w_{ij} \end{aligned} \tag{11}$$

IV. Input Reconstruction-based Rules

The rules described in this section require a feedforward pass to calculate the activation of nodes in the network, and a feedback pass through the transposed weight matrix to calculate the values necessary to update the synaptic weights. These rules can generally be implemented in Cog by using a transform to broadcast the transpose of the synaptic weight matrix from the output nodes to the input nodes. The equations summarized in this section include backpropagation [2][17], Harpur's law [8] and the contrastive divergence law [10].

A. Backpropagation

A typical backpropagation network can have multiple layers, with a typical architecture consisting of three layers, where recurrence may be introduced [4][12]. The change in synaptic weights is governed by the following equation:

$$\Delta w_{ij}^m = \eta \delta_i^m V_j^{m-1} \tag{12}$$

where m is the layer and:

$$V_j^m = g(h_i^m) = g(\sum_j w_{ij}^m v_j^{m-1}) \tag{13}$$

$$\delta_i^m = g'(h_i^m)[\zeta_i^m - V_i^M] \tag{14}$$

for $m = max(m)$ and:

$$\delta_i^{m-1} = g'(h_i^{m-1}) \sum_j w_{ij}^m \delta_j^m \tag{15}$$

for $m < max(m)$. Typically, node activation spreads up the layers, whereas learning occurs offline when the error is backpropagated from the output to the hidden and input layers. The process is supervised, and requires multiple iterations through the same input/output pairs in an interleaved fashion. This makes it difficult to use backpropagation in contexts where learning must be fast, such as when learning should occur on-line as the system "behaves", or when the correct output is not known. It is possible to recast the backpropagation algorithm in terms of our general form equation (11) by using the substitutions listed in Table 4. The general form equation of learning in backpropagation is found in Table 6.

TABLE IV
SUBSTITUTIONS TO CONVERT THE CANONICAL BACKPROPAGATION ALGORITHM INTO OUR GENERAL FORM EQUATION

Canonical backpropagation notation	Adjusted notation
H	λ
V_j^{m-1}	x_i
δ_i^m	$\xi(y_j)$
h_j^m	y_j
ζ_i^m	r_j

In this case equation (14) becomes

$$\xi(y_j) = g'(y_j)[r_j - g(y_j)] \tag{16}$$

for the top-most layer, where r_j is the external supervision; equation (15) becomes

$$\xi(y_j) = g'(y_j)\sum_k w_{jk}\xi(y_k) \tag{17}$$

for all other layers, where the index k denotes the cells in the upper layer and y denotes partial inference in this upper layer. Rule (12) simplifies to:

$$\Delta w_{ij} = \lambda x_i \xi(y_j) \tag{18}$$

where $\xi(y_j)$ is calculated on the node according to equation (16) or (17). Equation (18) is just the first term of equation (11). Table 6 depicts an implementation of the backpropagation rule. Compared with the rules below, backpropagation is simpler since it only requires one forward pass through the transform, and the weights are calculated directly from the results of reverse pass. On the other hand, it is more complex than the rules below since for each layer all reverse passes through the network have to be completed before learning can happen at the bottom-most layer.

B. Harpur's Rule

Harpur's algorithm, unlike backpropagation, requires two forward passes through the transform, first with the original input and second with the reconstructed error term as input. It is possible to recast Harpur's algorithm in terms of our general form equation (11) by using the substitutions listed in Table 5.

TABLE V
SUBSTITUTIONS TO CONVERT THE CANONICAL HARPUR'S ALGORITHM INTO OUR GENERAL FORM EQUATION

Canonical Harpur notation	Adjusted notation
H	λ
$x_p - Y_d$	x_i
a	$\xi(y_j)$

Then the learning law for Harpurs rule is defined as:

$$\Delta w_{ij} = \lambda \xi(y_j) x_i \tag{19}$$

Here the input signal x_i is actually an error term computed through input reconstruction:

$$x_i = x_p - Y_d = x_p - \sum_{j=1}^{M} \xi(y_j) w_{ij} \tag{20}$$

and $\xi(y_j)$ is a dynamic function of partial inference updated on each step by the following increment

$$\Delta\xi(y_j) = \mu y_j = \mu \sum_{i=1}^{N} x_i w_{ij} \tag{21}$$

Implementing Harpur's rule in Cog by requires that a complete cycle of learning will two computational cycles, one per forward pass, with the learning itself only occurring during the second pass.

C. Hinton's Contrastive Divergence

In matrix notation, Hinton's contrastive divergence algorithm can be summarized in five steps: $Y = WX$, $\dot{W} = k(XY)$, $X_2 = W^T Y$ (first Gibbs bounce), $Y_2 = WX_2$ (second Gibbs bounce), and $\dot{W} = -k(X_2 Y_2)$. This rule requires two passes and transmission of X_2 values instead of I on the second pass. This raises similar concerns as storing reconstructed input values with other error correcting rules, where it is necessary to store temporary values on nodes at the previous stage. The contrastive divergence rule could be implemented in Cog by requiring two forward passes, but unlike Harpur's rule learning has two stages, with each stage happening after their respective forward pass. To integrate these learning rules into the general form equation, we need to specify the functions that are passed from nodes to the transform and modify the first term of equation (11). They are presented in Table 6.

V. EXPLICIT TEMPORAL TRACE-BASED RULES

This class of learning laws includes rules that require the maintenance of an explicit temporal trace to calculate the synaptic weight change. These rules appear both in the reinforcement learning literature, such as Rescorla-Wagner [16] and Temporal Difference [20], and in perceptual learning [5][6].

A. Rescorla-Wagner

In the Rescorla-Wagner learning equation, the synaptic change is defined as:

$$y_j = \sum_{i=1}^{N} w_{ij} x_i \tag{22}$$

$$\dot{w}_{ij} = \lambda(r_j - y_j)x_i \tag{23}$$

This second equation is then fit into the general form equation; see Table 7. This law is useful, but fails in many complex classical conditioning paradigms. Rescorla-Wagner has a Hebbian component, where x is the input which corresponds to the stimulus (Conditioned Stimulus, or CS), y is the prediction of reward that the stimulus will give, r is the actual reward, and $f()$ is a difference of y_j and reward signal r computed on the nodes and then passed to the transform.

B. Temporal Difference

In the Temporal Difference learning equation, the synaptic change is defined as:

$$\dot{w}_{ij}(t+1) = \lambda(r_j(t+1) + \gamma y_j(t+1) - y_j(t))x_i(t) \tag{24}$$

where

$$y_j = \sum_{i=1}^{N} w_{ij} x_i \tag{25}$$

Equation (24) can be rewritten as:

$$\dot{w}_{ij}(t) = \lambda(r_j(t) + \gamma y_j(t) - y_j(t-1))x_i(t-1) \tag{26}$$

The value of γ illustrates the importance of the rewards that are expected in the future. If $\gamma = 0$, only the reward at the current time matters. If $\gamma > 0$, then in addition to current rewards all future rewards also matter with an exponential decay of the importance. The temporal difference rule works in discrete time steps with the weight change happening on the next time step relative to predicting inputs. This might lead to an issue of having to know x and y from the previous time step, and copies for the current and previous time steps will need to be stored. It may be possible to approximate x_{t-1} with x_t, but this depends on how fast the input changes. The problem with this approximation is that we reduce (if not eliminate) the predictive power of the input, which reduces the utility of learning. By setting $\gamma = 0$ and $\Delta t \to 0$, this rule reduces to the Rescorla-Wagner rule. Another solution is to make it a two-pass rule, when on the first pass we use $x_i(t)$ to compute $y_j(t)$ and on the second pass we use $x_i(t-1)$ to compute weights. This requires double storage in the pre-synaptic nodes. One more solution is to set up the following algorithm by, at each step t: compute $y_j(t)$; compute $\dot{w}_{ij}(t)$ using a precomputed $-y_j(t-1)x_i(t-1)$ from the previous step $t-1$; pre-compute $-y_j(t)x_i(t)$ and store it on the chip for weight update on the next step $t+1$. This procedure requires double storage in the transform.

C. Foldiak

Foldiak proposed a modified Hebbian rule where learning is proportional to the product of the presynaptic activity and the trace of the postsynaptic activity, with the postsynaptically-gated decay also being based on the trace:

$$\Delta w_{ij}^{(t)} = \lambda \bar{y}_j^{(t)}(x_i^{(t)} - w_{ij}^{(t)}) \tag{27}$$

where:

$$\bar{y}_j^{(t)} = (1-\delta)\bar{y}_j^{(t-1)} + \delta y_j^{(t)} \tag{28}$$

With $\Delta t \to 0$ the latter equation reduces to the differential form $\frac{d\bar{y}_j}{dt} = \delta(y_j - \bar{y}_j)$ and the rule becomes close to the ones discussed in the threshold-based rules section. With a finite Δt the rule is similar to the ones used in other reinforcement learning settings (e.g., [20]), hence its inclusion in this section. The trace of the postsynaptic activity has the role of keeping a running average of the "categorization" cell to sample changes in the lower-level cells, mirroring the fact that that the features to be learned are relatively stable in the environment. By renaming $\bar{y}_j$ as θ_{y_j} and δ as ϵ, the Foldiak rule can be fit into the general form equation; see Table 7.

VI. Conclusion

Synaptic-level plasticity is one of the critical mechanisms subserving learning and memory, and is therefore an essential component in the development of a hardware and software framework aimed at implementing adaptive behavior at biological scale. This article contains a preliminary general form equation encompassing several learning law candidates for implementation on massively parallel memristive hardware realized in the Hewlett-Packard and Boston University collaboration. The Cog software architecture implements the above mentioned laws on Dendra chips, which are processors characterized by dense memristive memories coupled with conventional circuitry for inference and learning. This paper characterizes and implements four main families of rules: Basic Hebb rule derivatives, threshold-based rules, input reconstruction-based rules, and temporal trace-based rules. All equations of the four families fit in a general form equation that can be implemented on the Dendra chip.

References

[1] E.L. Bienenstock, L. Cooper and P. Munro, "Theory for the development of neuron selectivity: orientation specificity and binocular interaction in visual cortex," *The Journal of Neuroscience*, vol. 2, no. 1, pp. 31-48, 1982.

[2] A.E. Bryson and Y.C. Ho, *Applied optimal control: Optimization, estimation, and control.* Blaisdell Publishing Company, 1969.

[3] P. Dayan and L.F. Abbott, *Theoretical Neuroscience: Computational and mathematical modeling of neural systems.* The MIT Press, 2001.

[4] J.L. Elman, "Finding structure in time" *Cognitive Science*, vol. 14, pp. 179-211, 1990.

[5] P. Foldiak "Forming sparse representations by local anti-Hebbian learning," *Biological Cybernetics*, vol. 64, pp. 165-170, 1990.

[6] P. Foldiak "Learning invariance from transformation sequences," *Neural Computation*, vol. 3, pp. 194-200, 1991.

[7] A. Gorchetchnikov, M. Versace and M.E. Hasselmo, "A model of STDP based on spatially and temporally local information: derivation and combination with gated decay," *Neural Networks*, vol. 18, pp. 458-466, 2005.

[8] G.F. Harpur and R.W. Prager, "Development of low entropy coding in a recurrent network," *Computation in Neural Systems*, vol. 7, no. 2, pp. 277-284, 1996.

[9] D.O. Hebb, *The Organization of Behavior.* New York: Wiley, 1949.

[10] G.E. Hinton, "Training products of experts by minimizing contrastive divergence," *Neural Computation*, vol. 14, no. 8, pp. 1771-1800, 2002.

[11] N. Intrator and L.N. Cooper, "Objective function formulation of the BCM theory of visual cortical plasticity: Statistical connections, stability conditions," *Neural Networks*, vol. 5, pp. 3-17, 1992.

[12] M.I. Jordan, "Attractor dynamics and parallelism in a connectionist sequential machine," *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, Amherst, MA, pp. 531-586, 1986.

[13] C. Law and L. Cooper, "Formation of receptive fields according to the BCM theory in realistic visual environments," *Proceedings of the National Academy of Sciences*, vol. 91, pp. 7797-7801, 1994.

[14] W.B. Levy and O. Steward, "Temporal contiguity requirements for long-term associative potentiation/depression in the hippocampus," *Neuroscience*, vol. 8, pp. 791-797, 1983.

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

[16] R.A. Rescorla and A.R. Wagner, "A theory of Pavlovian conditioning: Variations in the effectiveness of reinforcement and nonreinforcement," In A.H. Black and W.F. Proskasy, eds, *Classical Conditioning II.* Appleton-Century-Crofts, pp. 64-99, 1972.

[17] D.E. Rumelhart, G.E. Hinton and R.J. Williams, "Learning internal representations by error propagation," In D.E. Rumelhart and J.L. McClelland, eds, *Parallel Distributed Processing.* vol. 1, MIT Press, pp. 318-362, 1986.

[18] G. Snider, "Instar and outstar learning with memristive nanodevices," *Nanotechnology*, vol. 22, 2010.

[19] G. Snider, R. Amerson, D. Carter, H. Abdalla, S. Qureshi, J. Leveille, M. Versace, H. Ames, S. Patrick, B. Chandler, A. Gorchetchnikov and E. Mingolla, “From Synapses to Circuitry: Using Memristive Memory to Explore the Electronic Brain,” *IEEE Computer*, February 2011.
[20] R. Sutton and A. Barto, *Reinforcement Learning.* MIT Press, 1998.
[21] M. Versace and B. Chandler, “MoNETA: A Mind Made from Memristors,” IEEE Spectrum, December 2010.

TABLE I

HEBB RULE DERIVATIVES USED IN FIRING RATE-BASED MODELS, AND CONVENTIONALLY CAPTURED BY EQUATION (3). THESE RULES ARE RECAST IN TERMS OF EQUATION (6), WHERE α IS PARAMETER.

Name	Equation	$f(y_j)$	β_1	$g(y_j)$	β_2	$h(y_j)$
Classic Hebb	$\dot{w}_{ij} = \lambda x_i y_j$	y_j	0	0	0	1
Hebb with passive decay	$\dot{w}_{ij} = \lambda x_i y_j - \alpha w_{ij}$	y_j	0	$\frac{\alpha}{\lambda}$	0	1
Outstar	$\dot{w}_{ij} = \lambda x_i y_j - \alpha x_i w_{ij}$	y_j	$\frac{\alpha}{\lambda}$	0	0	1
Instar	$\dot{w}_{ij} = \lambda x_i y_j - \alpha y_j w_{ij}$	y_j	0	$\frac{\alpha}{\lambda} y_j$	0	1
Oja [15]	$\dot{w}_{ij} = \lambda x_i y_j - \alpha y_j^2 w_{ij}$	y_j	0	$\frac{\alpha}{\lambda} y_j^2$	0	1
Dual OR	$\dot{w}_{ij} = \lambda x_i y_j - \alpha(x_i + y_j) w_{ij}$	y_j	$\frac{\alpha}{\lambda}$	$\frac{\alpha}{\lambda} y_j$	0	1
Dual AND	$\dot{w}_{ij} = \lambda x_i y_j - \alpha x_i y_j w_{ij}$	1	$\frac{\alpha}{\lambda}$	0	0	y_j

TABLE II

SPIKE-TIMING DEPENDENT PLASTICITY (STDP) HEBB RULE DERIVATIVES USED IN SPIKING MODELS AND REPRESENTED BY EQUATION (4) RECAST IN TERMS OF EQUATION (6), WHERE α, α_1, AND α_2 ARE PARAMETERS.

Name	Equation	$f(y_j)$	β_1	$g(y_j)$	β_2	$h(y_j)$
Simple	$\dot{w}_{ij} = \lambda(x_i y_j - w_{ij})$	y_j	0	1	0	1
Pre-synaptic gating	$\dot{w}_{ij} = \lambda(x_i y_j - w_{ij}) \alpha x_i$	y_j	0	1	α	0
Post-synaptic gating	$\dot{w}_{ij} = \lambda(x_i y_j - w_{ij}) \alpha y_j^2$	y_j	0	1	0	αy_j^2
Dual OR	$\dot{w}_{ij} = \lambda(x_i y_j - w_{ij})(\alpha_1 x_i + \alpha_2 y_j^2$	y_j	0	1	α_1	$\alpha_2 y_j^2$
Dual AND	$\dot{w}_{ij} = \lambda(x_i y_j - w_{ij}) \alpha x_i y_j^2$	y_j^3	0	y_j^2	α	0

TABLE III

THRESHOLD-BASED RULE VARIATIONS THAT ARE USUALLY REPRESENTED BY EQUATION (8) AND ARE RECAST IN TERMS OF EQUATION (9), WHERE α AND ϵ ARE PARAMETERS AND δ' IS A DERIVATIVE OF THE SIGMOIDAL ACTIVATION FUNCTION.

Name	Equation	$f(y_j)$	β_1	$g(y_j)$	β_3	$h(y_j, \theta)$
Covariance 1	$\dot{w}_{ij} = \lambda(x_i y_j - x_i \theta_{y_j})$ $\dot{\theta}_{y_j} = \epsilon(y_j - \theta_{y_j})$	y_j	1	0	0	1
Covariance 2	$\dot{w}_{ij} = \lambda(x_i y_j - y_j \theta_{x_i})$ $\dot{\theta}_{x_i} = \epsilon(x_i - \theta_{x_i})$	y_j	0	y_j	0	1
Original BCM	$\dot{w}_{ij} = \lambda(x_i y_j - x_i \theta_{y_j}) y_j - \alpha w_{ij}$ $\dot{\theta}_{y_j} = \frac{y_j}{\epsilon} - \theta_{y_j}$	y_j	1	0	α	y_j
IBCM [11]	$\dot{w}_{ij} = \lambda(x_i y_j - x_i \theta_{y_j}) y_j \delta'(y_j)$ $\dot{\theta}_{y_j} = \epsilon(y_j^2 - \theta_{y_j})$	y_j	1	0	0	$y_j \delta'(y_j)$
LBCM [13]	$\dot{w}_{ij} = \lambda(x_i y_j - x_i \theta_{y_j}) \frac{y_j}{\theta_{y_j}}$ $\dot{\theta}_{y_j} = \epsilon(y_j^2 - \theta_{y_j})$	y_j	1	0	0	$\frac{y_j}{\theta_{y_j}}$
Textbook BCM [3]	$\dot{w}_{ij} = \lambda(x_i y_j - x_i \theta_{y_j}) y_j$ $\dot{\theta}_{y_j} = \epsilon(y_j^2 - \theta_{y_j})$	y_j	1	0	0	y_j

TABLE VI

INPUT RECONSTRUCTION-BASED RULES RECAST IN TERMS OF EQUATION (11)

Name	Equation	$f(y_j)$	β_1	$g(y_j)$	β_3	$h(y_j, \theta)$
Backpropagation	$\dot{w}_{ij} = \lambda x_i \xi(y_j)$	$\xi(y_j)$	0	0	0	1
Harpur	$\dot{w}_{ij} = \lambda x_i \xi(y_j)$	$\xi(y_j)$	0	0	0	1
Contrastive Divergence	$\dot{w}_{ij} = \lambda x_i y_j$	First pass: y_j	0	0	0	1
		Second pass: $-y_j$	0	0	0	1

TABLE VII

TEMPORAL TRACE-BASED RULES RECAST IN TERMS OF EQUATION (11)

Name	Equation	$f(y_j)$	β_1	$g(y_j)$	β_3	$h(y_j, \theta)$
Rescorla-Wagner	$\dot{w}_{ij} = \lambda x_i (r_j - y_j)$	$r_j - y_j$	0	0	0	1
Temporal Difference	$\dot{w}_{ij}(t) = \lambda x_i(t-1)(r_j(t) + \gamma y_j(t) - y_j(t-1))$	$r_j(t) + \gamma y_j(t) - y_j(t-1)$	0	0	0	1
Foldiak	$\dot{w}_{ij} = \lambda \theta_{y_j}(x_i - w_{ij})$ $\dot{\theta}_{y_j} = \epsilon(y_j - \theta_{y_j})$	1	0	1	0	θ_{y_j}