Bidirectional Associative Memories: Unsupervised Hebbian Learning to Bidirectional Backpropagation

Bart Kosko, *Fellow, IEEE*

***Abstract*—Bidirectional associative memories (BAMs) pass neural signals forward and backward through the same web of synapses. Earlier BAMs had no hidden neurons and did not use supervised learning. They tuned their synaptic weights with unsupervised Hebbian or competitive learning. Two-layer feedback BAMs always converge to fixed-point equilibria for threshold or threshold-like neurons. Every rectangular connection matrix is bidirectionally stable. These simpler BAMs extend to arbitrary hidden layers with supervised learning if the resulting bidirectional backpropagation algorithm uses the proper layer likelihood in the forward and backward directions. Bidirectional backpropagation lets users run deep classifiers and regressors in reverse as well as forward. Bidirectional training exploits pattern and synaptic information that forward-only running ignores.**

***Index Terms*—Bidirectional associative memory (BAM), bidirectional backpropagation, global stability, Hebbian learning.**

I. Bidirectional Associative Memories

EVERY real matrix is bidirectionally stable. That matrix theorem holds for two-layer feedback networks so long as the layers have threshold or threshold-like neurons [1]. The feedback network always converges to a bidirectional fixed point subject to minimal conditions on how and when the neurons fire. The result is a bidirectional associative memory (BAM).

This article reviews such BAMs and shows how they extend to the modern probabilistic case of supervised deep learning with bidirectional backpropagation. Fig. 1 compares an older 2-layer unsupervised BAM with a new deep supervised BAM when both networks try to represent a permutation mapping and its inverse. Fig. 2 shows the basic BAM theorem at work as a 2-layer BAM converges rapidly and asynchronously to a bidirectional fixed-point equilibrium.

Fig. 3 shows that the new supervised bidirectional backpropagation algorithm [2] can train a multilayer network forwards and backwards without overwriting the training in the opposite direction. The plots show the training iterations that produced the 3-layer logistic BAM in Fig. 1(b). Fig. 4 shows that a deep neural classifier can run in reverse to advantage if it trains with bidirectional backpropagation and the proper layer likelihood. Running an ordinary deep classifier in reverse produces only noise at the input. Running the same classifier in reverse after proper BAM training produces the input pattern that the network expects to see given its training and the current stimulation. This reverse or top-down signal approximates the centroid of the sampled pattern class. This bidirectional training also tends to improve classification accuracy in deep convolutional classifiers and adversarial networks [3]. It also trained the 3-layer BAM in Fig. 1.

The BAM memory matrix M is the n-by-p weight matrix that connects the two fields of neurons of a 2-layer BAM. Neural signals flow forward from the input neural layer F_X through M. Then neural signals flow backward from the output neural layer F_Y through the matrix transpose M^{T}. Both fields use the *same* web of synapses. Any other 2-layer bidirectional network must use two distinct synaptic weight matrices M and N. It must use M in the forward direction and some p-by-n matrix N in the backward direction that differs from M^{T}. BAMs are minimal heteroassociators in this sense that they use the same matrix in both directions.

Hidden layers of neurons can change a BAM's structure. The hidden layers can help the BAM store and recall more patterns. They can also undermine its feedback stability. A properly trained deep BAM can represent and approximate far more functions than can a 2-layer BAM. But the two-layer BAM matrix theorem may no longer hold.

The two BAMs in Fig. 1 show how a single hidden layer can boost a network's approximation power. Both BAMs try to learn the 4-bit permutation mapping π in Table I and its inverse π^{-1}. The task is to produce a BAM that maps a bipolar vector X_k at the input layer to the corresponding output bipolar vector Y_k over the same web of synapses that maps Y_k back to X_k. So $X_1 = \begin{pmatrix}-1 & -1 & -1 & -1\end{pmatrix}$ must map to $Y_1 = \begin{pmatrix}1 & 1 & -1 & -1\end{pmatrix}$. Representing the inverse map π^{-1} requires that Y_1 map back to X_1. Such bidirectional recall must hold for all 16 paired associations in Table I.

The first panel of Fig. 1 shows a classical 2-layer threshold BAM trained with unsupervised Hebbian learning. It cannot learn all 16 associations in its correlation matrix M in (18). Extensive simulations found that such a BAM can encode at most eight of the bipolar vector associations from Table I. The 2-layer BAM simply lacks the power to represent or approximate most functions. The much larger 2-layer BAM in Fig. 2 does easily store and recall three paired associations with Hebbian learning because the number of neurons in each field greatly exceeds the number of stored patterns.

Manuscript received November 30, 2020; revised December 4, 2020; accepted December 4, 2020. Date of current version January 12, 2021. This article was recommended by Associate Editor R. Kozma.

The author is with the Department of Electrical and Computer Engineering, University of Southern California, Los Angeles, CA 90089 USA (e-mail: kosko@usc.edu).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TSMC.2020.3043249.

Digital Object Identifier 10.1109/TSMC.2020.3043249

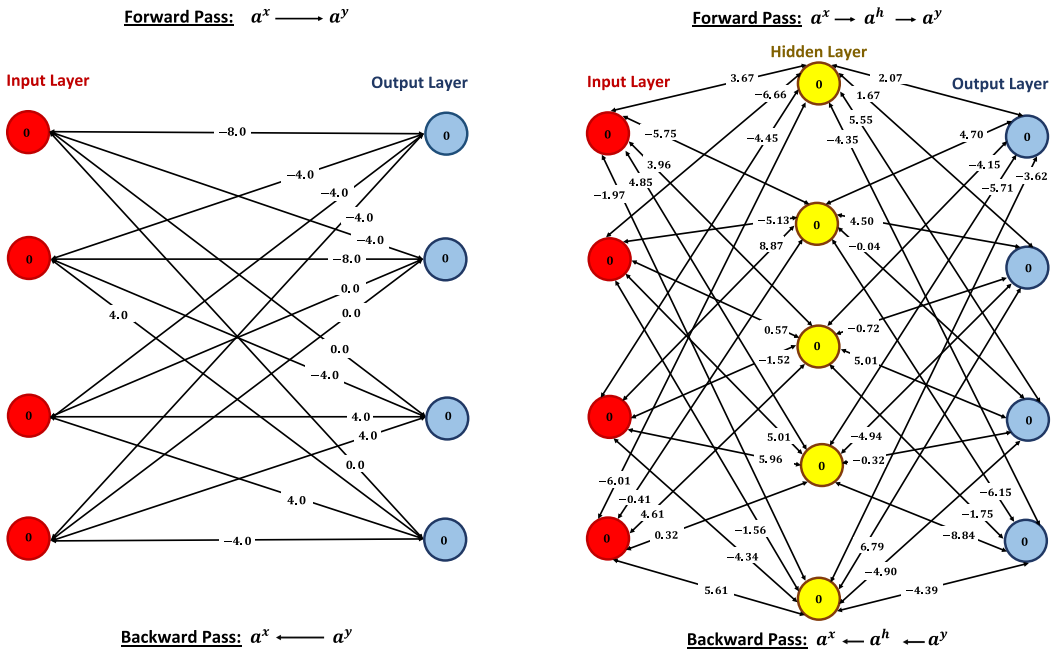

Fig. 1. Bidirectional approximation and representation of the permutation function π in Table I and its inverse π^{-1}. (a) 2-layer BAM with no hidden units. Its correlation memory matrix M in (18) can accurately store and recall only 8 of the 16 vector associations as BAM fixed-point equilibria. Those learned associations are the first eight listed in Table I. (b) 3-layer BAM exactly represents the 4-bit bipolar permutation function π and its inverse π^{-1} over the same web of synapses because it trained with supervised bidirectional backpropagation. A 3-layer BAM with 2^n hidden threshold neurons can exactly represent any n-bit permutation and its inverse [2].

The second panel in Fig. 1 shows a 3-layer BAM trained with bidirectional backpropagation. This 3-layer BAM encodes the entire permutation mapping in Table I and its inverse mapping. It achieves this bidirectional mapping feat at the cost of using hidden threshold (steep logistic) neurons. It also trains with the far more computationally heavy bidirectional backpropagation learning algorithm [2], [3]. It turns out that a 3-layer threshold BAM with 2^n hidden neurons can exactly represent any n-bit permutation mapping. So bidirectional backpropagation here reduced the needed number of hidden neurons from 16 to 5. The final section shows how bidirectional backpropagation uses the BAM's global likelihood structure to solve the problem of how to train synapses in one direction without overwriting their prior training in the reverse direction.

BAMs can also combine the rapid convergence of unsupervised two-layer BAMs with supervised multilayer BAMs. Simple 2-layer BAMs can pretrain the contiguous layers of a deep network with unsupervised learning. These simple BAMs correspond to restricted Boltzmann machine networks in this pretraining context [4]. The statistical version of such Boltzmann-machine pretraining turns out to be a special case of the generalized expectation–maximization (EM) algorithm for maximum-likelihood estimation [5]. The ABAM theorem below shows that this convergence of this adaptive system is deterministic and requires no statistical interpretation.

The next sections present a more general form of the discrete BAM theorem and review the continuous version of the adaptive BAM theorem for unsupervised Hebbian or competitive learning. The final section shows how bidirectional backpropagation can train a deep classifier or regressor by exploiting the probabilistic structure of each neural layer in the forward and backward directions.

TABLE I
4-BIT BIPOLAR PERMUTATION FUNCTION (SELF-BIJECTION) π AND ITS INVERSE π^{-1} THAT THE 3-LAYER BAM NETWORK REPRESENTS EXACTLY IN FIG. 1(b). THE INVERSE π^{-1} MAPS THE OUTPUT y BACK TO THE CORRESPONDING INPUT x

Input x	Output y	Input x	Output y
$[--\,--]$	$[++\,--]$	$[--\,++]$	$[++\,-+]$
$[--\,-+]$	$[++\,+-]$	$[-+\,-+]$	$[-+\,-+]$
$[--\,+-]$	$[++\,++]$	$[-+\,+-]$	$[+-\,--]$
$[+-\,++]$	$[-+\,+-]$	$[-+\,++]$	$[-+\,--]$
$[-+\,--]$	$[+-\,--]$	$[+-\,--]$	$[+-\,++]$
$[++\,-+]$	$[--\,--]$	$[+-\,-+]$	$[-+\,++]$
$[++\,+-]$	$[--\,-+]$	$[+-\,+-]$	$[+-\,+-]$
$[++\,++]$	$[--\,++]$	$[++\,--]$	$[--\,+-]$

II. BAM GLOBAL STABILITY

A Lyapunov argument shows that every 2-layer BAM is globally stable. The discrete case differs from the continuous case both in terms of the BAM system Lyapunov function and in terms of the update strategy. We first present the discrete case in some detail. The next section shows how stability applies in the continuous and more general case when unsupervised learning laws update the memory matrix while the neurons change. The proof technique is similar. Some forms of BAM stability can still hold for neural signals with time delays that can model the transmission times of axonal signals [6]. Other BAM models with time delays ensure solutions with periodic oscillations [7].

Global stability shows that a 2-layer BAM system converges to a BAM fixed-point attractor. So such BAMs avoid complex periodic equilibria. Nor do they wander chaotically in the state space. But global stability does not show *which* attractor the BAM converges to. The result is akin to knowing that an airplane will land at an airport but not knowing which airport.

Discrete BAM stability involves a back-and-forth convergence path from an initial input state vector A to a final fixed-point attractor (A_f, B_f). The input state A passes forward through the memory matrix M. The nonlinear operations at the output layer produce the output state vector B. The state vectors A and B are binary or bipolar vectors in practice. The output state vector B feeds back through the transpose matrix M^{T} and leads to the new input state vector A'. Input A' feeds forward through M and leads to the new output state vector B'. This pendulum-like process continues until the fixed input vector A_f produces the fixed output B_f and conversely. Then the vector pair (A_f, B_f) defines a BAM fixed-point equilibrium. The next section shows that a BAM always (and rapidly) converges to a BAM fixed-point equilibrium for threshold or threshold-like neurons for any connection matrix M. The result holds whether all neurons in a field update at once or if a random subset updates asynchronously.

A. Discrete BAM Stability

A discrete BAM can update its neurons in almost arbitrary fashion and still ensure global stability to a BAM fixed-point attractor. Such updates can only decrease the bounded Lyapunov function. The stability result holds for any synaptic weight matrix M because the bidirectional operation in effect symmetrizes the rectangular matrix M into a 2-by-2 block matrix with null diagonal blocks and with M and M^{T} as the off-diagonal blocks.

A problem can arise if the forward and backward updates occur at the same time. Then the Lyapunov function can increase in some cases. This problem does not arise in practice since users update BAM neurons in one direction at a time. It does not arise at all in continuous BAMs. It can arise in the special case of a discrete Hopfield network if the user updates all neurons at the same time [8], [9].

The original stability theorem for discrete BAMs assumed that the neurons were on–off threshold neurons. The proof showed that the inner-product input to an updated neuron had the same sign as the update had. So the product of both signs was positive. Changes in the global energy or the Lyapunov function just summed the negative of these positive terms. The network converged to a bidirectional fixed point when the energy function reached a lower bound. This stepwise decrease was not trivial because the threshold state changes ensured a minimal downward step size. We here extend this theorem to allow steep sigmoid functions such as logistics to approximate the thresholds. The same proof goes through if we assume that there remains a minimal downward step size in the energy function.

The 2-layer BAM consists of n threshold or threshold-like neurons in the X field and p such neurons in the Y field. The arbitrary n-by-p matrix M connects the input neuron field F_X to the output field F_Y. The output field F_Y can be a hidden layer in more general mutlilayer BAMs. The fields F_X and F_Y can also have fixed self-connections or intrafield connections so long as the corresponding n-by-n and p-by-p connection matrices are symmetric. We assume for simplicity that they are null matrices. We also here ignore exogenous inputs to the neurons without any loss of generality. The section on continuous neurons below restores these external inputs.

The following activation notation describes the BAM network operations. The ith neuron in F_X converts the inner-product input o_i^x to the input sigmoidal activation $a_i^x(o_i^x)$. Let $\mathbf{a}^X(t)$ denote the row vector of input F_X neurons at time t: $\mathbf{a}^X(t) = (a_1^x(t), \ldots, a_n^x(t))$. The stand-alone activation a_i^x also implies at a given time that it receives input o_i^x: $a_i^x = a_i^x(o_i^x)$. Let $\mathbf{a}^Y$ likewise denote the row vector of p output (or hidden) neural activations in the field F_Y: $\mathbf{a}^Y = (a_1^y(o_1^y), \ldots, a_p^y(o_p^y))$. The memory matrix value m_{ij} of M denotes the directed synaptic path from the ith neuron in F_X to the jth neuron in F_Y.

The default activation a_j^y is a steep binary or bipolar logistic function of its argument o_j^y. The binary activation has the form

$$a_j^y(o_j^y) = \frac{1}{1 + \exp(-c\, o_j^y)} \tag{1}$$

$$= \frac{1}{1 + \exp(-c \sum_{i=1}^n a_i^x m_{ij})} \tag{2}$$

for inner-product input

$$o_j^y = \sum_{i=1}^n a_i^x m_{ij} \tag{3}$$

and for steepness parameter $c > 0$. Values $c \geq 5$ produce *de facto* binary threshold functions. But the logistic activation has a simple and smooth derivative

$$\frac{\partial a_j^y}{\partial o_j^y} = c\, a_j^y (1 - a_j^y). \tag{4}$$

The activation a_j^y can also be a classical binary threshold with threshold T_j

$$a_j^y(o_j^y) = \begin{cases} 1 & \text{if } o_j^y > T_j \\ 0 & \text{if } o_j^y < T_j. \end{cases} \tag{5}$$

The jth neuron does not change state if $o_j^y = T_j$ (or the neuron can randomly break the tie). This allows asynchronous updates because then a neuron need not make an update decision at a given time t even if the inner-product input o_j^y exceeds or falls below the threshold T_j. The threshold neuron simply maintains its current on-or-off status in such cases.

The output activation vector $\mathbf{a}^Y$ of p logistic values defines a point in the p-dimensional unit hypercube $I^p = [0, 1]^p : \mathbf{a}^Y \in I^p$. So the output state vector $\mathbf{a}^Y$ defines a finite fuzzy set [10], [11]. Its time evolution defines a sequence of p-dimensional fuzzy sets in the unit p-cube. The bipolar logistic activation b_j^y scales and translates the corresponding binary activation a_j^y to a value in the bipolar interval $[-1, 1]$: $b_j^y = 2a_j^y - 1$. Then the bipolar state vector $\mathbf{b}^Y$ is a point in the p-dimensional bipolar cube $[-1, 1]^p$.

The network energy or Lyapunov function E has a forward-pass component E_f and a backward-pass component E_b. The energy has the sum form $E = E_f + E_b$. We will see below that bidirectional backpropagation has a similar summed-error structure because of the log-likelihood structure that arises from the joint bidirectional likelihood: $\ln p(\mathbf{y}|\mathbf{x}, M)p(\mathbf{x}|\mathbf{y}, M) = \ln p(\mathbf{y}|\mathbf{x}, M) + \ln p(\mathbf{x}|\mathbf{y}, M)$. This is the key insight in deriving the bidirectional backpropagation algorithm [2].

The directional energy E itself is just a smooth function of the system coordinates $x_1, \ldots, x_n$ and $y_1, \ldots, y_p$. A simple Taylor's series expansion gives the energy as a quadratic form since the first-order condition for an extremum is that all first partial derivatives equal zero [9]. This gives the forward-pass energy E_f as the scaled quadratic form $E_f = E_f(\mathbf{a}^X, \mathbf{a}^Y|M) = -(1/2)\mathbf{a}^X M(\mathbf{a}^Y)^{\mathrm{T}}$. The backward-pass energy E_b has likewise a quadratic form: $E_b = -(1/2)\mathbf{a}^Y M^{\mathrm{T}}(\mathbf{a}^X)^{\mathrm{T}}$. Taking the transpose shows that these two energies are equal because the transpose of a scalar is just the scalar: $E_b = -(1/2)\mathbf{a}^Y M^{\mathrm{T}}(\mathbf{a}^X)^{\mathrm{T}} = (-(1/2)\mathbf{a}^Y M^{\mathrm{T}}(\mathbf{a}^X)^{\mathrm{T}})^{\mathrm{T}} = -(1/2)\mathbf{a}^X M(\mathbf{a}^Y)^{\mathrm{T}} = E_f$ for any matrix M. Then the total energy of the 2-layer BAM equals the unscaled quadratic form

$$E = E_f + E_b = -\frac{1}{2}\mathbf{a}^X M\big(\mathbf{a}^Y\big)^{\mathrm{T}} - \frac{1}{2}\mathbf{a}^Y M^{\mathrm{T}}\big(\mathbf{a}^X\big)^{\mathrm{T}} \tag{6}$$

$$= -\mathbf{a}^X M\big(\mathbf{a}^Y\big)^{\mathrm{T}} \tag{7}$$

$$= -\sum_{i=1}^{n}\sum_{j=1}^{p} a_i^x a_j^y m_{ij}. \tag{8}$$

So the BAM energy is just the unweighted average of the forward and backward energies. The absolute matrix entries $|m_{ij}|$ give a finite lower (and upper) bound on the total BAM energy E

$$E\big(\mathbf{a}^X, \mathbf{a}^Y|M\big) \geq -\sum_{i=1}^{n}\sum_{j=1}^{p}\big|m_{ij}\big|. \tag{9}$$

The proof of the basic discrete BAM theorem shows that any activation state change Δa_i^x or Δa_j^y in either direction must decrease the energy E. So $\Delta E < 0$ holds along BAM state trajectories.

Discrete BAM Theorem: Every connection matrix M is bidirectionally stable for threshold or threshold-like neurons and for asynchronous or synchronous state updates.

Proof: Consider a forward pass from the input neural field F_X to the output or hidden neural field F_Y. Assume that at least one output neuron changes state from time t to the current time $t+1$: $|\Delta a_j^y(t+1)| = |a_j^y(t+1) - a_j^y(t)| > 0$. Then the update vector $\Delta\mathbf{a}^Y(t+1) = (\Delta a_1^y(t+1), \ldots, \Delta a_p^y(t+1))$ is not the null vector. So the updates at F_Y at time $t+1$ can involve any of the $2^p - 1$ asynchronous or random update choices of the p output neurons.

We assume that the jth neuron in F_Y has a binary threshold activation as in (5). This involves no loss of generality because the convergence argument uses only the sign of $\Delta a_j^y(t+1)$. We do assume that the logistic activation (1) or any other smooth sigmoid activation is sufficiently steep so that discrete state changes are not trivially small. We also assume for simplicity that all threshold neurons have zero thresholds: $T_j = 0$ in (5). Then either $\Delta a_j^y(t+1) = 1$ or $\Delta a_j^y(t+1) = -1$ holds. This exclusive-or disjunction holds because the nonzero state change $\Delta a_j^y(t+1) \neq 0$ implies that either $\Delta a_j^y(t+1) = a_j^y(t+1) - a_j^y(t) = 1 - 0 = 1$ or $\Delta a_j^y(t+1) = a_j^y(t+1) - a_j^y(t) = 0 - 1 = -1$ holds. The nonzero state change $\Delta a_j^y(t+1) \neq 0$ for a logistic activation likewise implies that either $\Delta a_j^y(t+1) > 0$ for an activation increase or $\Delta a_j^y(t+1) < 0$ for an activation decrease.

Consider first the change in the BAM energy ΔE for a forward pass. Put $\Delta E(t+1) = E(t+1) - E(t)$. Then the threshold update in (5) can only decrease the energy on the forward pass if $\Delta a_j^y(t+1) \neq 0$ for at least one output neuron

$$\Delta E(t+1) = -\mathbf{a}^X(t)M\big(\Delta\mathbf{a}^Y(t+1)\big)^{\mathrm{T}} \tag{10}$$

$$= -\sum_{j=1}^{p}\left(\sum_{i=1}^{n} a_i^x(t)m_{ij}\right)\Delta a_j^y(t+1) \tag{11}$$

$$< 0 \tag{12}$$

because $\mathrm{sign}(\Delta a_j^y(t+1)) = \mathrm{sign}(\sum_{i=1}^{n} a_i^x(t)m_{ij})$ from (5). The sum over the p output neurons shows that $\Delta E < 0$ holds along trajectories for the synchronous update of all output neurons or for any non-null asynchronous update choice of F_Y neurons because the sign equality holds for every output neuron such that $\Delta a_j^y(t+1) \neq 0$.

The same argument holds for the backward-sweep update epoch. Then the output activation state vector $\Delta a^Y(t+1)$ passes through the transpose M^{T}. Assume that it updates at least one input neuron so that $\Delta a_i^x(t+1) \neq 0$ holds

$$\Delta E(t+1) = -\mathbf{a}^Y(t+1)M^{\mathrm{T}}\big(\Delta\mathbf{a}^X(t+1)\big)^{\mathrm{T}} \tag{13}$$

$$= -\sum_{i=1}^{n}\left(\sum_{j=1}^{p} a_j^y(t+1)m_{ij}\right)\Delta a_i^x(t+1) \tag{14}$$

$$< 0 \tag{15}$$

because $\mathrm{sign}(\Delta a_i^x(t+1)) = \mathrm{sign}(\sum_{j=1}^{p} a_j^y(t+1)m_{ij})$. The backward update sweep follows the forward update sweep. So synchronous or asynchronous updates at one field cannot interfere with updates at the other. So the bounded energy E decreases along trajectories: $\Delta E < 0$. ■

The proof shows that we can weaken the assumption that each neuron update strictly decreases the energy in a forward or a backward sweep. The total energy inequality $\Delta E < 0$ requires only that the negative neuron updates in one field outweigh the positive updates in that field. So requiring only net-negative updates allows mild forms of random or noisy update schemes. We can further weaken the assumption and allow simultaneous updates in both fields so long as the total number of negative neuron updates outweighs the total number of positive updates at discrete-time increment t. We can still further weaken the assumption by allowing nonmontonic activation functions such as Gaussians so long as the sigmoidal updates swamp them in the final update tally.

The next section shows how this basic BAM global stability extends to the more complex case where synaptic learning takes place while the neurons at F_X and F_Y change as well. The

results hold for the general class of Cohen–Grossberg neural models that include the important special cases of additive and shunting neural dynamics [12].

III. Adaptive BAM Theorems

A natural question is whether the memory matrix M itself can change slightly and still preserve BAM stability as the neurons update. The fact that every matrix M is bidirectionally stable suggests that this should hold for at least slight perturbations of the matrix elements m_{ij}.

Any change to the memory matrix M is a form of learning. Learning in a feedback system risks instability because changing the synaptic values in M can undermine the stability of the two neural fields F_X and F_Y. Changing the neural fields also risks destabilizing learning. Neural stability itself corresponds to pattern formation. The final snapshot of Fig. 2 shows this with the stable recall of the (S, E) association. So pattern learning should make just those changes to M that allow it to encode new neural patterns at F_X and F_Y

Pattern learning creates a *stability–plasticity* dilemma for learning in any feedback system. The synaptic values in M must be plastic enough so that changes encode the neural patterns that play out across the connected neural fields. But the synaptic values in M cannot change so much that they undo the stable neural patterns that the synaptic values try to encode.

The next section develops a discrete adaptive BAM theorem that balances both horns of the stability–plasticity dilemma with unsupervised Hebbian learning. The result extends to the continuous case. It also opens the way to supervised bidirectional backpropagation in the multilayer case.

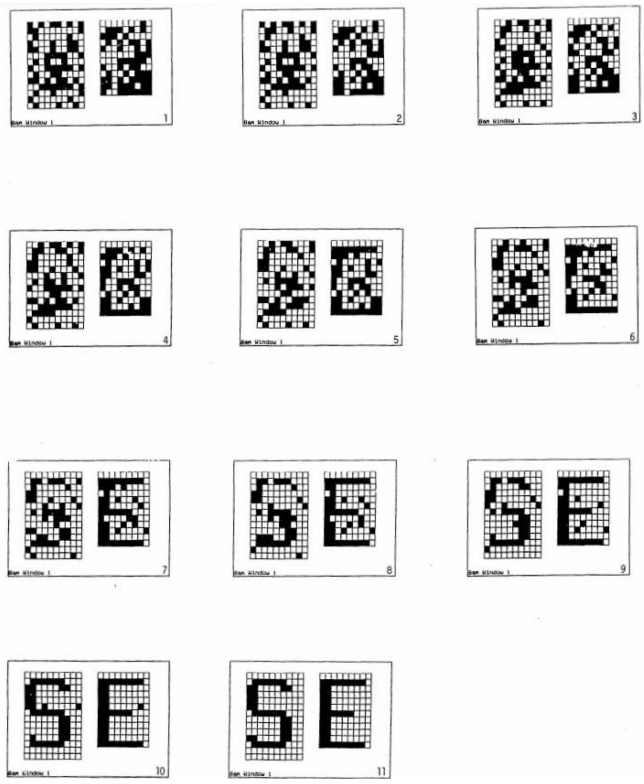

Fig. 2. Asynchronous recall in a discrete Hebbian BAM with two layers of threshold neurons. A 40% noise-corrupted version of the association (S, E) stimulated the BAM as an input. The network updated six neurons at random per field on its way to convergence to the BAM fixed-point equilibrium (S, E). The input field F_X had 140 threshold neurons. The output field F_Y had 108 threshold neurons. The 140-by-108 memory matrix M summed three bipolar outer-product matrices to encode the three associative patterns (S, E), (M, V), and (G, N).

A. Hebbian Correlation Matrix Learning

A simple way to update M is also one of the oldest: just add up outer-product correlation matrices [13], [14]. The resulting memory matrix M can encode only a small number of pattern associations as BAM fixed-point equilibria as in Fig. 1(a) or Fig. 2.

Such correlation encoding gives insight into why these correlation BAM matrices lead to accurate pattern recall if the number m of associations is small. It also suggests unsupervised learning laws that extend BAM stability.

A BAM correlation memory matrix M encodes m bipolar vector associations (X_j, Y_j). The discrete (offline) learning processing starts with m bipolar associations $(X_1, Y_1), \ldots, (X_m, Y_m)$ for bipolar row vectors $X_j \in \{-1, 1\}^n$ and $Y_j \in \{-1, 1\}^p$. Encode the jth association (X_j, Y_j) in the n-by-p bipolar outer-product matrix $M_j = X_j^{\mathrm{T}} Y_j$. Then encode all m associations as the sum of the m outer-product matrices M_j

$$M = \sum_{j=1}^{m} M_j = \sum_{j=1}^{m} X_j^{\mathrm{T}} Y_j. \tag{16}$$

This simple correlation form of distributed learning shows how to erase or forget the jth association: just subtract M_j from M. The matrix $-M_j$ associates the stimulus or if-part vector X_j with the response or then-part complement Y_j^c since $Y_j^c = -Y_j$ for bipolar vectors. Binary vector associations (A_j, B_j) admit bipolar encoding through the transformations $X_j = 2A_j - I$ and $Y_j = 2B_j - I$ if I is the respective n-vector or p-vector of all 1s.

Correlation encoding is a type of unsupervised Hebbian learning in the product sense that *neurons that fire together wire together* [9]. It likewise creates a BAM memory matrix M that accurately stores and recalls only a few associations (X_j, Y_j) as BAM fixed points.

The BAM memory matrix in Fig. 1(a) stores eight bipolar associations from the bipolar 4-bit permutation map π in Table I. The permutation map $\pi : \{-1, 1\}^4 \to \{-1, 1\}^4$ is just one of the 16! or 20, 922, 789, 888, 000 such 4-bit bipolar permutations. Each permutation map is one-to-one and onto. So the inverse π^{-1} exists. It just maps the 4-bit string on the right back to the corresponding 4-bit string on the left.

Extensive simulations showed that the correlation technique (16) can store and recall at most 8 of the 16 vector associations in permutation mapping π in Table I. These eight associations correspond to the first eight entries of Table I. The first associative pair is (X_1, Y_1) with $X_1 = \begin{pmatrix} -1 & -1 & -1 & -1 \end{pmatrix}$ and $Y_1 = \begin{pmatrix} 1 & 1 & -1 & -1 \end{pmatrix}$. This gives the first outer-product memory matrix M_1 as

$$M_1 = X_1^{\mathrm{T}} Y_1 = \begin{pmatrix} -1 & -1 & 1 & 1 \\ -1 & -1 & 1 & 1 \\ -1 & -1 & 1 & 1 \\ -1 & -1 & 1 & 1 \end{pmatrix}. \tag{17}$$

The other seven matrices likewise give 4-by-4 bipolar matrices. Then the final BAM memory matrix M sums these first eight outer-product matrices

$$M = \sum_{j=1}^{8} X_j^{\mathrm{T}} Y_j = \begin{pmatrix} -8 & -4 & 0 & 0 \\ -4 & -8 & -4 & 4 \\ -4 & 0 & 4 & 4 \\ -4 & 0 & 4 & -4 \end{pmatrix}. \tag{18}$$

The BAM matrix M in (18) recalls the first eight associations (X_j, Y_j) in Table I as reverberating bidirectional fixed points. Consider the synchronous update with zero threshold when X_1 passes through M: $X_1 M = (20, 12, -4, -4) \rightarrow (1, 1, -1, -1) = Y_1$. Thresholding the backward pass of Y_1 through M^{T} gives $Y_1 M^{\mathrm{T}} = (-12, -12, -12, -4) \rightarrow (-1, -1, -1, -1) = X_1$. So the pair (X_1, Y_1) reverberates in perpetuity until an external stimulus perturbs the system. Slight perturbations to these vectors still converge to the same bidirectional fixed-point equilibrium. Another example is the 7th pair (X_7, Y_7) with $X_7 = (1, 1, 1, -1)$ and $Y_7 = (-1, -1, -1, 1)$. Then the forward pass gives $X_7 M = (-12, -12, -4, 12) \rightarrow (-1, -1, -1, 1) = Y_7$. The backward pass gives $Y_7 M^{\mathrm{T}} = (12, 20, 4, -4) \rightarrow (1, 1, 1, -1) = X_7$. So the pair (X_7, Y_7) is also a BAM fixed point.

B. Hebbian Correlation Decoding

This section shows why correlation BAMs tend to recall the correct learned association if the number m of stored associations is small relative to the input and output dimensions n and p: $m < \min(n, p)$. The argument uses only the structure of bipolar and binary vector spaces and simple assumptions about the distribution of 1s in a vector and the similarity of metrics. It does not use BAM feedback stability directly. It shows instead that correlation learning increases the probability that threshold decoding will select the appropriate vector at each iteration on the path toward BAM convergence. The argument further shows that bipolar inputs X_k produce more accurate recall on average than do binary inputs A_k.

The analysis starts with the correlation coefficient c_{ij}

$$c_{ij} = X_i \cdot X_j = X_i X_j^{\mathrm{T}} \tag{19}$$

where $X_i \cdot X_j$ denotes the ordinary inner product of two finite vectors. The correlation coefficient c_{ij} is an integer that lies in the range $-n \leq c_{ij} \leq n$. It scales an output bipolar vector Y_k in associative recall from a correlation matrix M as in (16)

$$X_k M = n Y_k + \sum_{j \neq k}^{m} c_{kj} Y_j \tag{20}$$

because $c_{kk} = X_k X_k^{\mathrm{T}} = n$ since the bipolar vector X_k lies in the n-cube $\{-1, 1\}^n$ of bipolar n-vectors. The first term nY_k in (20) acts as the signal term because the correlation matrix M stores the pair (X_k, Y_k). The second term in (20) acts as cross-talk noise from the other $m - 1$ associations (X_j, Y_j) stored in M.

The sign and magnitude of the correlation coefficients c_{ij} assist the threshold decoding process. The input key X_k puts the maximum positive weight n on Y_k in (20). An input X that is close X_k gives a correlation coefficient $X \cdot X_k$ that is likewise close to n. So this signal coefficient does its best to magnify Y_k in the output sum (20). This magnification only helps the threshold process in (5) recover Y_k.

The other $m-1$ coefficients c_{kj} also help make the crosstalk term in (20) more likely to threshold to Y_k. This effect depends on a subtle connection between the bipolar cube $\{-1, 1\}^n$ of bipolar vectors X_k and the binary or Boolean cube $\{0, 1\}^n$ of the corresponding bit vectors A_k. Define the l^1 or Hamming distance $H(A_i, A_j)$ between the bit vectors A_i and A_j as the number of slots in which the two vectors differ

$$H(A_i, A_j) = \sum_{u=1}^{n} |a_i^u - a_j^u| \tag{21}$$

since $|a_i^u - a_j^u| = 0$ if and only if $a_i^u = 0 = a_j^u$ or $a_i^u = 1 = a_j^u$ at the uth slot. Then we can interchange the bipolar-based correlation coefficients c_{ij} with the Hamming distances $H(A_i, A_j)$ through the equality

$$c_{ij} = n - 2H(A_i, A_j). \tag{22}$$

The equality follows from writing $c_{ij} = X_i \cdot X_j$ as the number of slots where the two bipolar vectors are equal minus the number where they differ. The latter difference is just the Hamming distance $H(A_i, A_j)$. So the former difference is $n - H(A_i, A_j)$.

The equality (22) shows that the sign and magnitude of the correlation coefficient c_{ij} describe the metrical relationship of the two bit vectors A_i and A_j in the Boolean n-cube

$$c_{ij} \gtreqless 0 \quad \text{if and only if} \quad H(A_i, A_j) \lesseqgtr \frac{n}{2}. \tag{23}$$

Two bipolar vectors X_i and X_j are closer to each other than each is to the other's complement if $c_{ij} > 0$. Then the corresponding bit vectors A_i and A_j are less than half their space away from each other: $H(A_i, A_j) < n/2$. The bipolar vectors are farther apart if $c_{ij} < 0$. Then the two-bit vectors are more than half their space away from each other: $H(A_i, A_j) > n/2$.

Assume that an approximate continuity condition holds between the distances in the input Boolean cube $\{0, 1\}^n$ and in the output Boolean cube $\{0, 1\}^p$

$$\frac{1}{n} H(A_i, A_j) \approx \frac{1}{p} H(B_i, B_j) \tag{24}$$

if B_i is the bit vector that corresponds to the output bipolar vector Y_i. Then the $m - 1$ correlation coefficients c_{ij} in the crosstalk term in (20) use their signs and magnitudes to make the $m - 1$ bipolar vectors Y_j tend to match the signal term Y_k.

This thresholding effect improves if the number $m - 1$ of crosstalk terms is small. It also improves to the extent that the input vectors X_j or A_j are spread out or approximately orthogonal. Repeated BAM sweeps can compound these denoising effects. So the Hebbian correlation structure of M gives at least a partial explanation of *where* the BAM converges given an input key X that resembles X_k more than it resembles any of the other stored input vectors X_j.

This same metrical argument helps explain the common observation that correlation-matrix BAMs converge faster and more accurately if the input keys are bipolar vectors X_k rather than binary vectors A_k. A recent comparison of using bipolar versus binary logistic neurons at the BAM fields found that bipolar inputs tended to converge faster than binary inputs by

an order of magnitude for a memory matrix based on samples from the MNIST dataset of hand-drawn digits [5].

The key result is that the correlation coefficient c_{ij} has *twice* the magnitude of the mixed coefficient $A_i \cdot X_j$

$$c_{ij} = 2A_i \cdot X_j \tag{25}$$

if the number of 1s in A_i equals the number of 0s: $|A_i| = A_i \cdot I = n/2$ for each input bit vector A_i. The assumption that $|A_i| = n/2$ holds does not depend on the order of the bits in the vector. It implies that $|X_i| = 0$ since $|X_i| = X_i \cdot I = (2A_i - I)I^{\mathrm{T}} = n - n = 0$ since $X_i = 2A_i - I$ for the row vector I of n 1s.

The inner-product equality in (25) holds approximately if the number of 1s in A_i is approximately $n/2$: $|A_i| \approx n/2$. This tends to hold for large n in the sense of a simple symmetric random walk that starts at the origin.

We show now that the expected number of 1s in A_i is exactly $n/2$ if the n bit-vector components a_i^u define n independent Bernoulli random variables with common success probability $p = 1/2$. Then the bipolar transform $x_i^u = 2a_i^u - 1$ defines the corresponding bipolar component x_i^u as an independent Rademacher random variable: $P(x_i^u = 1) = p$ and $P(x_i^u = -1) = q = 1 - p$. The expected value of the n random bipolar slot variables x_i^u is just the sum of the expectations: $E[X_i \cdot I] = n(p - q) = 0$ if $p = 1/2 = q$. Then slot independence gives the variance as $V[X_i \cdot I] = 4npq = n$ if $p = 1/2$. Then the expected number of bits in A_i is $E[|A_i|] = E[A_i \cdot I] = E[1/2(I + X_i) \cdot I] = n/2 + 1/2E[X_i \cdot I] = n/2 + 1/2n(p - q) = np$. So $E[|A_i|] = n/2$ holds. Independence likewise implies that $V[|A_i|] = 1/4\ V[X_i \cdot I] = npq$. So the variance of the sum of 1s is $n/4$. These last two results also follow from the fact that a sum of independent and identically distributed Bernoulli random variables is a binomial random variable. The strong law of large numbers [15] further shows that the sample mean of the independent Bernoulli-component vectors A_i converges with probability one to a constant vector A whose components sum to $n/2$.

The argument for the inner-product equality (25) also uses the bipolar transform $X_i = 2A_i - I$

$$X_i \cdot X_j - A_i \cdot X_j = \left(X_i - \frac{1}{2}X_i - \frac{1}{2}I\right) \cdot X_j \tag{26}$$

$$= \frac{1}{2}X_i \cdot X_j \tag{27}$$

since $|A_j| = n/2$ holds and implies that $1/2I \cdot X_j = 1/2|X_j| = 0$. Rearrangement gives $c_{ij} = X_i \cdot X_j = 2A_i \cdot X_j$. Then (22) and (23) imply the further metrical insight into the magnitude and sign of bipolar keys compared with binary keys

$$c_{ij} \gtreqless A_i \cdot X_j \quad \text{if and only if} \quad H(A_i, A_j) \lesseqgtr \frac{n}{2} \tag{28}$$

if $|A_i| = n/2$ holds at least approximately for all bit vectors A_i. The same assumptions should hold for the p-dimensional bit vectors B_i on the reverse BAM pass.

C. Discrete ABAM Theorem

The previous section showed how discrete correlation learning helps explain where a BAM converges in some cases. Offline correlation learning with (16) also gives a simple algorithm for programming a 2-layer BAM. This section shows that these local correlations naturally extend the discrete BAM theorem to the adaptive case. The next section presents continuous versions of this result for Hebbian and competitive learning.

Suppose that the memory weight m_{ij} updates slightly *after* both the F_X and F_Y fields have updated at time $t + 1$

$$m_{ij}(t + 1) = m_{ij}(t) + \Delta m_{ij}(t + 1). \tag{29}$$

Then what learning increment $\Delta m_{ij}(t+1) = m_{ij}(t+1) - m_{ij}(t)$ still ensures stability with the energy function $E(\mathbf{a}^X, \mathbf{a}^Y|M) = -\mathbf{a}^X M(\mathbf{a}^Y)^{\mathrm{T}}$ in (8)?

The memory matrix update ΔM produces the energy change $\Delta E_M(t + 1)$

$$\Delta E_M(t + 1) = -\mathbf{a}^X \Delta M\left(\mathbf{a}^Y\right)^{\mathrm{T}} \tag{30}$$

$$= -\sum_{i=1}^{n}\sum_{j=1}^{p} a_i^x(t + 1)a_j^y(t + 1)\Delta m_{ij}(t + 1). \tag{31}$$

So the energy decrease $\Delta E_M(t + 1) < 0$ holds if the learning increment $\Delta m_{ij}(t + 1)$ simply correlates the *local* input and output activations a_i^x and a_j^y. This gives an *unsupervised* and local Hebbian learning law

$$\Delta m_{ij}(t + 1) = c_{t+1}a_i^x(t + 1)a_j^y(t + 1) \tag{32}$$

for sufficiently decreasing rate c_{t+1} to bound m_{ij}. Then inserting this Hebbian learning law $\Delta M(t + 1) = (\mathbf{a}^X)^{\mathrm{T}}\mathbf{a}^Y$ into the energy update ΔE_M decreases the bounded Lyapunov function along adaptive BAM trajectories

$$\Delta E_M(t + 1) = -\sum_{i=1}^{n}\sum_{j=1}^{p}\left(\Delta m_{ij}(t + 1)\right)^2 < 0. \tag{33}$$

The total energy state change is $\Delta E = \Delta E_f + \Delta E_b + \Delta E_M < 0$ so long as a single neuron activation or synaptic weight changes. This proves the discrete adaptive BAM theorem for Hebbian learning.

Discrete ABAM Theorem: Hebbian learning $\Delta M(t + 1) = (\mathbf{a}^X)^{\mathrm{T}}\mathbf{a}^Y$ is bidirectionally stable for threshold or threshold-like neurons and for asynchronous or synchronous state updates.

Learning in practice scales the learning increment ΔM with a learning-rate constant $c_{t+1} > 0$ to bound M. Simulation may also scale the activations. We omit these rate constants for clarity in the next section.

D. Continuous ABAM Theorems

The discrete Hebbian correlation learning law $\Delta m_{ij} = a_i^x a_j^y$ in (29) goes over in the continuous case to a Hebbian activation learning law for bounded activations [9]

$$\dot{m}_{ij}(t) = -m_{ij} + a_i^x(t)a_j^y(t) \tag{34}$$

where the overdot denotes time differentiation: $\dot{m}_{ij} = dm_{ij}/dt$.

The term $-m_{ij}$ describes the synapse's inherent passive decay in the absence of neural stimulation. Then $\dot{m}_{ij} = -m_{ij}$ implies exponential forgetting of all initial memory $m_{ij}(0)$: $m_{ij}(t) = m_{ij}(0)e^{-t} \to 0$. Some form of such exponential

decay holds for most biological as well as electrical memory elements. We assume without loss of generality that the activations are bounded in the unit interval: $0 \leq a_i^x \leq 1$ and similarly for a_j^y.

The Hebbian learning law quickly learns stable activation values a_i^x and a_j^y. The equilibrium condition $\dot{m}_{ij} = 0$ implies that the synaptic weight m_{ij} encodes a Hebbian correlation: $m_{ij} = a_i^x a_j^y$. This learned correlation value is either 0 or 1 for stable binary coding or either -1 or 1 for stable bipolar coding. The bipolar learning increment always obeys the bound $-1 \leq a_i^x a_j^y \leq 1$.

The Hebbian law (34) is a first-order linear differential equation. So it has an exact solution. The solution is straightforward once the activations have stabilized. Suppose that the learning product $a_i^x a_j^y$ equals either -1 or $+1$ after the neural fields have stabilized. Suppose first that $a_i^x a_j^y = 1$. Then the learning law $\dot{m}_{ij} + m_{ij} = 1$ is a first-order inhomogeneous linear differential equation with constant coefficients. It has the exact solution

$$m_{ij}(t) = e^{-t} m_{ij}(0) + \int_0^t e^{s-t} ds \tag{35}$$

$$= e^{-t} m_{ij}(0) + 1 - e^{-t}. \tag{36}$$

So $m_{ij}(t) \to 1$ exponentially quickly for any initial memorized information $m_{ij}(0)$. The case $a_i^x a_j^y = -1$ likewise leads to -1 exponentially quickly.

The Hebbian adaptive BAM system combines the correlation learning law (34) with dynamical equations that describe the input arguments o_i^x and o_j^y. We here state only *additive* dynamics because they both show the proof technique and often occur in practice and in biological modeling

$$\dot{o}_i^x = -o_i^x + \sum_{j=1}^{p} a_j^y\left(o_j^y\right) m_{ij} + I_i^x \tag{37}$$

$$\dot{o}_j^y = -o_j^y + \sum_{i=1}^{n} a_i^x\left(o_i^x\right) m_{ij} + I_j^y \tag{38}$$

for external forcing inputs I_i^x and I_j^y. We assume here for simplicity that these inputs vary so slowly as to be constants. The sum terms are self-excitation terms. The proof of the Hebbian ABAM theorem requires that the activations are bounded and nondecreasing: $a_i^{x\prime} = da_i^x / do_i^x \geq 0$. This always holds for logistic and other sigmoidal activations. The more general Cohen–Grossberg dynamics can require other assumptions to keep the Lyapunov system bounded [12].

The continuous Lyapunov energy function E must allow for changing memory weight values m_{ij}. This amounts to adding a constant to the earlier quadratic energy. That constant turns out to be the scaled trace value $(1/2)\text{Trace}(MM^{\text{T}})$. It had a zero time derivative but now depends on the Hebbian learning law (34). This gives the ABAM energy function as a sum of five terms plus the scaled trace term

$$E\left(\mathbf{a}^X, \mathbf{a}^Y | M\right) = -\sum_{i=1}^{n}\sum_{j=1}^{p} a_i^x\left(o_i^x\right) a_j^y\left(o_j^y\right) m_{ij} + \sum_{i=1}^{n}\int_0^{o_i^x} a_i^{x\prime}\left(u_i^x\right) u_i^x du_i^x - \sum_{i=1}^{n} a_i^x\left(o_i^x\right) I_i^x + \sum_{j=1}^{p}\int_0^{o_j^y} a_j^{y\prime}\left(v_j^y\right) v_j^y dv_j^y - \sum_{j=1}^{p} a_j^y\left(o_j^y\right) I_j^y + \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{p} m_{ij}^2. \tag{39}$$

Taking the time derivative of E requires the triple product rule of differentiation for the quadratic sum and the chain rule for the two integrals. This gives

$$\dot{E} = -\sum_{i=1}^{n} a_i^{x\prime}\left(o_i^x\right)\dot{o}_i^x\left[-o_i^x + \sum_{j=1}^{p} a_j^y\left(o_j^y\right) m_{ij} + I_i^x\right] - \sum_{j=1}^{p} a_j^{y\prime}\left(o_j^y\right)\dot{o}_j^y\left[-o_j^y + \sum_{i=1}^{n} a_i^x\left(o_i^x\right) m_{ij} + I_j^y\right] - \sum_{i=1}^{n}\sum_{j=1}^{p} \dot{m}_{ij}\left[a_i^x a_j^y - m_{ij}\right]. \tag{40}$$

Then inserting the dynamical models (34) and (37)-(38) shows that the bounded energy function decreases along system trajectories

$$\dot{E} = -\sum_{i=1}^{n} a_i^x\left(\dot{o}_i^x\right)^2 - \sum_{j=1}^{p} a_j^y\left(\dot{o}_j^y\right)^2 - \sum_{i=1}^{n}\sum_{j=1}^{p} \dot{m}_{ij}^2 \tag{41}$$

$$< 0 \tag{42}$$

for any change in the doubly dynamical system of changing neurons and changing synapses. This gives the Hebbian ABAM theorem.

Hebbian ABAM Theorem: The Hebbian ABAM system (34) and (37)-(38) is globally stable.

The ABAM view of a brain or similar system is that the system continually moves through global equilibria. The equilibria change as external and internal stimuli perturb the many interconnected neural fields and synapses and as they struggle to converge. A thought or perception would correspond to such a stable global equilibrium before it dissolved and transformed into another equilibrium. ABAM convergence depends in no way on the number $n+p$ of neurons or on the number np of synapses. Nor does it require any form of synchronization or supervision. A large-scale ABAM would exist in a type of default dream state subject to external stimulus forcing from the environment.

The ABAM theorem also holds for competitive learning with a steep win–loss activation. All synapses change in general with some form of Hebbian learning. Competitive learning limits learning to the synapses that flow into that neuron in the output field F_Y that wins that layer's competition for input activation [16]

$$\dot{m}_{ij}(t) = a_j^y\left[a_i^x(t) - m_{ij}\right] \tag{43}$$

where the win–loss activation a_j^y is such a steep binary logistic or threshold that it acts like the indicator function I_{C_k} for the kth input pattern class C_k. Suppose the jth neuron in F_Y wins: $a_j^y = 1$. Then the weight m_{ij} converges exponentially quickly to the input activation a_i^x.

Intralayer synaptic connections at F_Y can guide the competitive dynamics for activation from the F_X field. Most competitive learning models in practice approximate these competitive dynamics by simply picking the winner as that neuron in F_Y whose synaptic fan-in vector most closely matches the current input activation vector $\mathbf{a}^X$. Competitive networks form the basic building blocks of the family of adaptive resonance theory (ART) networks [17], [18]. A competitive ABAM gives a crude approximation of a simple ART system. It also ensures rapid global stability that does not depend on the size of the network.

A steep output activation a_j^y and (43) preserve the decreasing structure of the last term on the right-hand side of (40)

$$\dot{m}_{ij}[a_i^x a_j^y - m_{ij}] = a_j^y[a_i^x - m_{ij}][a_i^x a_j^y - m_{ij}] \quad (44)$$

$$= (a_i^x - m_{ij})^2 \quad (45)$$

if the jth neuron in F_Y wins: $a_j^y = 1$. The right-hand side of (44) equals zero if the jth neuron loses: $a_j^y = 0$. So the far right-hand term in (40) can only decrease. That still gives $\dot{E} \leq 0$ along competitive ABAM trajectories. That energy decrease establishes the competitive ABAM theorem.

The final section shows how to extend BAMs to multiple layers and how to train them with supervision.

IV. Bidirectional Backpropagation for Deep Networks

The new bidirectional backpropagation algorithm shows how supervised learning can train a deep neural classifier or regressor in both the forward and backward directions [2]. This defines a generalized BAM because the neural signals flow forward and backward in deep sweeps through the *same* web of synapses. Rectangular matrices M_h still connect contiguous layers from the input field F_X through the hidden layers on to the final output field F_Y. The backward flow still uses the transpose M_h^{T} of these synaptic matrices. So the basic and minimal BAM structure still holds.

Bidirectional backpropagation's time complexity is $O(n)$ for n training samples because unidirectional backpropagation has $O(n)$ complexity. The bidirectional forward and backward sweeps give a total complexity of $O(n) = O(n) + O(n)$. So bidirectional backpropagation scales like ordinary backpropagation.

The forward sweep starts with an input pattern $\mathbf{x}$ and ends with the output $\mathbf{y} = N(\mathbf{x})$. Denote the result of the backward pass through the network as $N^{\mathrm{T}}(\mathbf{y}) = N^{\mathrm{T}}(N(\mathbf{x}))$ to reflect the backward pass through the transpose matrices M_h^{T}.

The backward signal $N^{\mathrm{T}}(\mathbf{y})$ that arrives back at the input layer acts as a type of network *attentive focus* on the input pattern $\mathbf{x}$ that stimulated the BAM network. This also resembles the top-down signal in ART. The network *expects* to see the pattern $N^{\mathrm{T}}(\mathbf{y})$ at the input given what it has learned and given the input stimulus $\mathbf{x}$ [16], [18].

Bidirectional backpropagation does not require that a network have a point inverse. The 3-layer BAM representation of the permutation function π and its inverse π^{-1} in Fig. 1 is the exception and not the rule. Most vector mappings do not have a *point* inverse. But they always have a *set-theoretical* inverse or pullback mapping $N^{-1} : 2^{R^K} \rightarrow 2^{R^n}$. The pullback maps output sets B back to input sets A: $A = N^{-1}(B) = \{\mathbf{x} \in R^n : N(\mathbf{x}) \in B\}$ if $B \subset R^K$. So a classifier's K output unit bit vectors $\mathbf{e}_1, \ldots, \mathbf{e}_K$ partition the input pattern space R^n into K pattern classes: $R^n = N^{-1}(\mathbf{e}_1) \cup \cdots \cup N^{-1}(\mathbf{e}_K)$.

The BAM's backward-pass output $N^{\mathrm{T}}(\mathbf{e}_k)$ is a point in the input pattern space R^n. It is not the pullback set $N^{-1}(\mathbf{e}_k)$. It may not even lie in $N^{-1}(\mathbf{e}_k)$.

The backward-pass output $N^{\mathrm{T}}(\mathbf{e}_k)$ may also serve as the answer to a *why* question: Why did the network produce this output $\mathbf{e}_k$? What caused the observed output? The ordinary forward-pass output $N(\mathbf{x})$ can answer a corresponding *what-if* question: What happens if $\mathbf{x}$ stimulates the system? What will it cause?

A multilayer BAM can define a classifier or a regressor or some combination of both. A typical feedforward deep classifier in fact defines a bidirectional classifier–regressor. The input pattern vector $\mathbf{x} \in \mathbb{R}^n$ enters the input layer F_X of n neurons with identity activations: $a_i^x(x_i) = x_i$. These identity neurons act as data registers. There may be several hidden layers of neurons that have logistic or quasilinear rectified-linear (ReLU) activations or that have some other form of activations.

What defines the classifier network is its final layer F_Y of K classifier neurons. These K neurons are almost always softmax activations in the literature of machine learning [19], [20]

$$a_j^y(o_j^y) = \frac{\exp(o_j^y)}{\sum_{k=1}^K \exp(o_k^y)}. \quad (46)$$

Then the output activation vector $\mathbf{a}^Y$ defines a K-dimensional probability vector. Supervised training of the classifier almost always uses 1-in-K encoding for the output softmax neurons. This technique codes the kth target vector $\mathbf{t}_k$ as the kth unit basis bit vector $\mathbf{e}_k$ if the input pattern $\mathbf{x}$ comes from the kth pattern class.

A related practice in machine learning rounds off the actual output probability vector $\mathbf{y}$ from (46) into a unit bit vector $\mathbf{e}_k$. The basis vector $\mathbf{e}_k$ has a 1 in the kth slot and has 0s in the other $K - 1$ slots. Users then declare that the deep network has classified the input pattern $\mathbf{x}$ to the kth decision class if the raw output $\mathbf{y} = N(\mathbf{x})$ rounds off to the unit basis vector $\mathbf{e}_k$.

The softmax activation (46) generalizes the logistic activation in (1)–(3) to K decision classes. The softmax probability functions extend the simple 2-class Bayes-theorem structure of a logistic probability to a K-class form of the Bayes theorem. The softmax activation also has a more complicated partial derivative with respect to its input

$$\frac{\partial a_j^y}{\partial o_k^y} = \begin{cases} -a_k^y a_j^y & \text{if } j \neq k \\ a_j^y(1 - a_j^y) & \text{if } j = k. \end{cases} \quad (47)$$

Only the case of $j = k$ corresponds to the non-negative logistic derivative (4). The other $K - 1$ cases when $j \neq k$ yield a non-positive derivative. Using this derivative and the multinomial likelihood of a softmax layer yields the same error-times-signal form (61) of the main backpropagation learning term as holds for logistic and identity layers. So backpropagation invariance still holds [5].

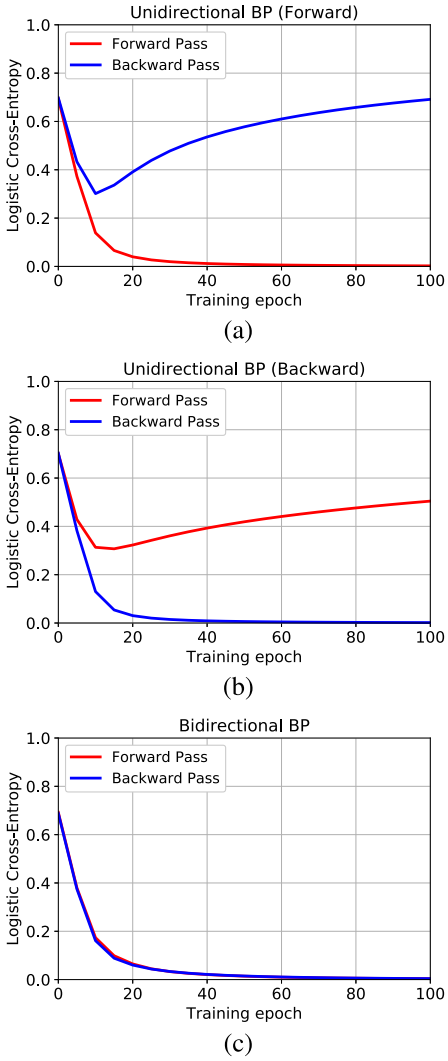

Fig. 3. Bidirectional backpropagation avoids the directional overwriting of ordinary unidirectional backpropagation. It uses the joint log likelihood in (55) and so uses a sum of forward and backward error functions. The three plots compare the training of the 3-layer logistic BAM in Fig. 1 that learns the permutation map π and its inverse π^{-1} from Table I. (a) Unidirectional backpropagation trains well on the forward error but overwrites the learning of the inverse map on the backward pass. (b) Unidirectional backpropagation trains well on the backward error but overwrites the learning of the permutation map on the forward direction. (c) Bidirectional BP trained the network with the summed double cross-entropies in (69). It produced no overwriting in either direction.

Identity neurons at a classifier's input layer may appear trivial because users treat them as ports for data entry. They are not trivial in terms of their likelihood. They imply a layer likelihood that is vector normal with a default white or diagonal covariance matrix $\mathbf{K}$ [5]: $\mathbf{t} \sim \mathcal{N}(\mathbf{t}|\mathbf{s}, \mathbf{K})$ for target vector $\mathbf{t}$. This also holds when the output neurons have identity activations in a regression network that approximates a function or a time series. Taking the logarithm of the vector normal likelihood gives the negative of the squared-error function in the second right-hand term of (69).

Bidirectional backpropagation itself involves two key probabilistic insights. The first is that the algorithm iteratively and locally maximizes the *joint* forward and backward network probabilities $p_f(\mathbf{y}|\mathbf{x}, \Theta)$ and $p_b(\mathbf{x}|\mathbf{y}, \Theta)$. These directional probabilities may also depend on several intervening hidden layers. The multilayer BAMs in Figs. 1(b) and 4 present the simplest case where the algorithm focuses on the likelihood structure of just the output and input neural layers. Their layer likelihood or probability structure also controls proper noise injection during training and may affect other algorithmic processes. The second insight is that the multiplication theorem of probability factors the directional probabilities into conditional-probability factors involving the layers.

Bidirectional backpropagation is a form of maximum-likelihood estimation. It seeks the locally optimal network weight or parameter vector Θ^* that maximizes the joint network likelihood

$$\Theta^* = \underset{\Theta}{\operatorname{argmax}}\, p_f(\mathbf{y}|\mathbf{x}, \Theta) p_b(\mathbf{x}|\mathbf{y}, \Theta) \tag{48}$$

where for the moment we omit the dependence on hidden layers. The logarithm is monotone increasing. So bidirectional backpropagation equally seeks the joint maximization of the directional log likelihoods

$$\Theta^* = \underset{\Theta}{\operatorname{argmax}} \ln p_f(\mathbf{y}|\mathbf{x}, \Theta) + \ln p_b(\mathbf{x}|\mathbf{y}, \Theta). \tag{49}$$

The gradient $\nabla_\Theta L = \nabla_\Theta \ln p_f(\mathbf{y}|\mathbf{x}, \Theta) + \nabla_\Theta \ln p_b(\mathbf{x}|\mathbf{y}, \Theta)$ gives the bidirectional backpropagation algorithm if backpropagation invariance holds at the input and output layers.

The multiplication theorem describes the likelihood structure of any deep neural network N for input vector $\mathbf{x}$ and output vector as $\mathbf{y}$. This is the same factorizing of a joint probability that shows how to find the probability of getting dealt three aces without replacement from a deck of 52 cards: $P(A_1 \cap A_2 \cap A_3) = P(A_1)P(A_2|A_1)P(A_3|A_1 \cap A_2) = (4/52)(3/51)(2/50) = 1/5525$.

The multiplication theorem applies whenever a network has hidden layers. So suppose that the network has k hidden layers of neurons $\mathbf{h}_1, \ldots, \mathbf{h}_k$. Suppose further that a large vector or array of parameters Θ^n describes the network structure at learning epoch n. Then the multiplication theorem shows how to factor the *total* forward network likelihood $p(\mathbf{y}, \mathbf{h}_k, \ldots, \mathbf{h}_1|\mathbf{x}, \Theta^n)$ into a product of the layer likelihoods in a forward pass through the deep network

$$\begin{aligned} p\big(\mathbf{y}, \mathbf{h}_k, \ldots, \mathbf{h}_1|\mathbf{x}, \Theta^n\big) &= p\big(\mathbf{y}|\mathbf{h}_k, \ldots, \mathbf{h}_1, \mathbf{x}, \Theta^n\big) \\ &\quad \times p\big(\mathbf{h}_k|\mathbf{h}_{k-1}, \ldots, \mathbf{h}_1, \mathbf{x}, \Theta^n\big) \cdots \\ &\quad p\big(\mathbf{h}_2|\mathbf{h}_1, \mathbf{x}, \Theta^n\big) p\big(\mathbf{h}_1|\mathbf{x}, \Theta^n\big). \end{aligned} \tag{50}$$

Taking logarithms in (50) gives the total forward log likelihood $L(\mathbf{x})$ for input pattern $\mathbf{x}$ at learning epoch n

$$L(\mathbf{x}) = L(\mathbf{y}|\mathbf{x}) + L(\mathbf{h}_k|\mathbf{x}) + \cdots + L(\mathbf{h}_1|\mathbf{x}) \tag{51}$$

where $L(\mathbf{h}_k|\mathbf{x}) = \ln p(\mathbf{h}_k|\mathbf{h}_{k-1}, \ldots, \mathbf{h}_1, \mathbf{x}, \Theta^n)$.

The total backward likelihood $p(\mathbf{x}, \mathbf{h}_1, \ldots, \mathbf{h}_k|\mathbf{y}, \Theta^n)$ has the reverse product form

$$\begin{aligned} p\big(\mathbf{x}, \mathbf{h}_1, \ldots, \mathbf{h}_k|\mathbf{y}, \Theta^n\big) &= p\big(\mathbf{x}|\mathbf{h}_1, \ldots, \mathbf{h}_k, \mathbf{y}, \Theta^n\big) \\ &\quad \times p\big(\mathbf{h}_1|\mathbf{h}_2, \ldots, \mathbf{h}_k, \mathbf{y}, \Theta^n\big) \cdots \\ &\quad p\big(\mathbf{h}_{k-1}|\mathbf{h}_k, \mathbf{y}, \Theta^n\big) p\big(\mathbf{h}_k|\mathbf{y}, \Theta^n\big). \end{aligned} \tag{52}$$

Taking logarithms gives the total backward log likelihood $L(\mathbf{y})$ for "input" $\mathbf{y}$ on the backward sweep through N^{T}

$$L(\mathbf{y}) = L(\mathbf{x}|\mathbf{y}) + L(\mathbf{h}_1|\mathbf{y}) + \cdots + L(\mathbf{h}_k|\mathbf{y}) \tag{53}$$

where $L(\mathbf{h}_j|\mathbf{y}) = \ln p(\mathbf{h}_j|\mathbf{h}_{j+1}, \ldots, \mathbf{h}_k, \mathbf{y}, \Theta^n)$.

A total *bidirectional* sweep passes neural signals forward through the deep network N and then backwards through its reverse mapping N^{T}. The reverse mapping N^{T} is not a point inverse. It is just the result of passing signals from the output back through all the neurons and transpose matrices in the network. This back *and* forth signal flow gives the total bidirectional network probability $p(\mathbf{x}, \mathbf{y}|\Theta^n)$ as the product

$$p\big(\mathbf{x}, \mathbf{y}|\Theta^n\big) = p\big(\mathbf{y}, \mathbf{h}_k, \ldots, \mathbf{h}_1|\mathbf{x}, \Theta^n\big)p\big(\mathbf{x}, \mathbf{h}_1, \ldots, \mathbf{h}_k|\mathbf{y}, \Theta^n\big). \tag{54}$$

Taking logarithms gives at last the total bidirectional log likelihood $L(\mathbf{x}, \mathbf{y})$ of the deep BAM network N as the sum of the directional log likelihoods

$$L(\mathbf{x}, \mathbf{y}) = \ln p\big(\mathbf{x}, \mathbf{y}|\Theta^n\big) = L(\mathbf{x}) + L(\mathbf{y}). \tag{55}$$

Multiplying the terminal layer likelihoods by prior probabilities gives penalized or Bayesian bidirectional backpropagation. Taking logarithms gives the total log posterior of the bidirectional network.

The next insight follows from a recent theorem. Backpropagation is a special case of the generalized EM algorithm for maximum-likelihood estimation of parameters [5], [21]. This important result follows ultimately from the fact that Shannon entropy minimizes cross entropy by way of Jensen's inequality and the concavity of the logarithm. Each gradient step of backpropagation equals exactly the gradient partial-maximization step of generalized EM as the network climbs the nearest hill of probability or log likelihood in the parameter space. The result follows from the EM trick of rearranging *any* hidden-layer posterior $p(\mathbf{h}|\mathbf{y}, \mathbf{x}, \Theta) = p(\mathbf{h}, \mathbf{y}|\mathbf{x}, \Theta)/p(\mathbf{y}|\mathbf{x}, \Theta)$ as the network's (forward) output probability $p(\mathbf{y}|\mathbf{x}, \Theta) = p(\mathbf{h}, \mathbf{y}|\mathbf{x}, \Theta)/p(\mathbf{h}|\mathbf{y}, \mathbf{x}, \Theta)$. Take logarithms of both sides. Then take expectations of both sides with respect to the hidden posterior $p(\mathbf{h}|\mathbf{y}, \mathbf{x}, \Theta^n)$ at iteration n of the parameter vector Θ^n. This gives the network log likelihood as $\ln p(\mathbf{y}|\mathbf{x}, \Theta) = Q(\Theta|\Theta^n) + H(\Theta|\Theta^n)$ for EM's surrogate likelihood $Q(\Theta|\Theta^n)$ and the cross entropy $H(\Theta|\Theta^n)$ at n. But the Shannon entropy $H(\Theta^n|\Theta^n)$ minimizes the cross entropy at each n: $H(\Theta^n|\Theta^n) \leq H(\Theta|\Theta^n)$. So the gradient is null: $\nabla_{\Theta^n} H(\Theta^n|\Theta^n) = \mathbf{0}$. Then $\nabla_{\Theta^n} \ln p(\mathbf{y}|\mathbf{x}, \Theta^n) = \nabla_{\Theta^n} Q(\Theta^n|\Theta^n)$ holds identically at each learning iteration n. So $BP = EM$. This result also shows how to noise-boost ordinary and bidirectional backpropagation by first noise-boosting generalized EM [3], [5].

The proof that $BP = EM$ further requires that *backpropagation invariance* holds at each layer of neurons. The layer's log-likelihood's parameter gradient $\nabla_{\Theta^n} L$ must give back the *same* basic signal-times-error BP learning law as in (61) [5]. This follows in turn if each layer's log likelihood L equals the negative of the layer's error function. The partial derivatives involving the inner hidden layers have the same form since here we focus only on training with respect to the BAM network's terminal layers. The same invariance holds internally in general because of the layer-likelihood factorization in (50)–(55).

There are three main cases of layer likelihoods in practice. A classifier's output layer of K softmax neurons has a layer likelihood of a one-shot multinomial or categorical probability distribution. A pass through the layer corresponds to the roll of a K-sided die. Then the log likelihood L equals the negative cross entropy. A regressor's output or hidden layer of identity neurons has a layer likelihood of a vector normal probability density. Then the log likelihood L equals the negative squared error of classical backpropagation [22], [23]. So maximizing the layer probability minimizes the squared error and vice versa. This property shows that backpropagation invariance holds for the gradient optimization of a softmax neural layer [5].

The third case occurs for a layer of logistic or sigmoidal neurons. The 3-layer BAM in Fig. 1(b) is a small-scale example. The output layer can consist of K such logistic neurons for a high-capacity network. These networks code with some of the 2^n vertices of the n-dimensional unit hypercube rather than code with just the n vertices of the embedded simplex as with softmax classifiers [24]. The layer likelihood corresponds to a product of Bernoulli probabilities or independent flips of K coins [5]. Then the log likelihood L equals the negative double cross entropy in (58).

The bidirectional BP algorithm updates the joint bidirectional log likelihood $L = L(\mathbf{x}) + L(\mathbf{y})$ one directional sweep at a time. That means it learns with the *sum* of the negative error function $E(\Theta^n) = E_f(\Theta^n) + E_b(\Theta^n)$ at learning epoch n. Training with either one of these directional errors alone results in overwriting or undoing the previous training in the reverse direction. Training with both errors avoids such overwriting and jointly reduces both errors. This may explain why earlier neural classifiers and regressors have operated only in the forward mode. Training them in reverse can only undermine the forward training since they use just a forward-only error function.

The 3-layer logistic BAM in Fig. 1(b) uses logistic–logistic bidirectional backpropagation to train its steep logistic neurons. We replaced all logistic neurons with actual threshold functions with zero thresholds after training. The logistic neurons and backpropagation invariance imply that the input and output layer likelihoods are products of independent Bernoulli probabilities [5]. So the forward likelihood $p_f(\mathbf{y}|\mathbf{x}, \Theta)$ has the product-Bernoulli form

$$p_f(\mathbf{y}|\mathbf{x}, \Theta) = \prod_{k=1}^{K} \big(a_k^y\big)^{y_k} \ \big(1 - a_k^y\big)^{1-y_k} \tag{56}$$

for bit-vector target $\mathbf{y}$. The backward-pass likelihood $p_b(\mathbf{x}|\mathbf{y}, \Theta)$ has a similar form. Taking logarithms gives the total bidirectional log likelihood and error $E(\Theta)$ as the double cross entropy

$$E(\Theta) = E_f(\Theta) + E_b(\Theta) \tag{57}$$

$$= -\sum_{k=1}^{K} y_k \ \ln a_k^y + (1 - y_k) \ \ln\big(1 - a_k^y\big) - \sum_{i=1}^{I} x_i \ \ln a_i^x + (1 - x_i) \ \ln\big(1 - a_i^x\big) \tag{58}$$

for I input logistic neurons.

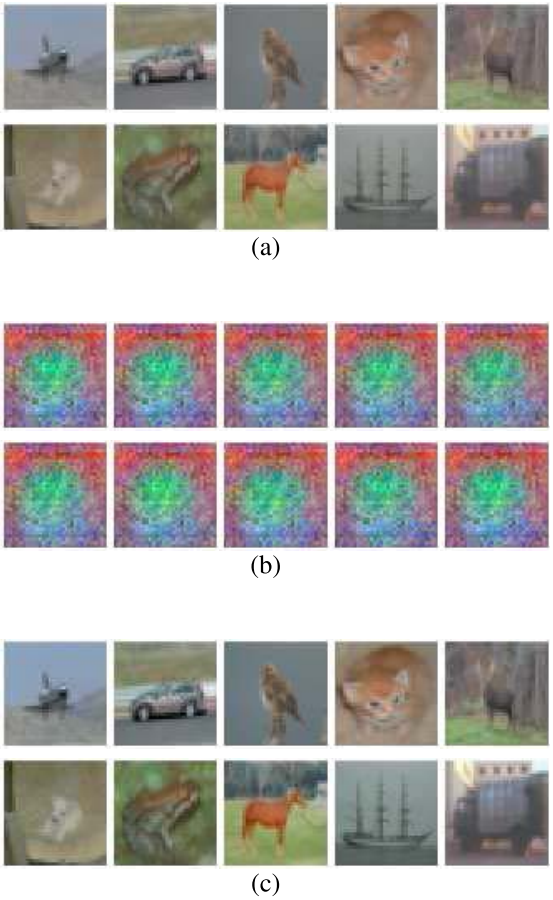

Fig. 4. Bidirectional recall in a deep neural classifier N after training with unidirectional versus bidirectional backpropagation on the CIFAR-10 image dataset. The multilayer BAM used seven hidden layers of 512 ReLU units each. Neural signals swept forwards and backwards through those layers from the 3072 input identity neurons to the ten output softmax neurons. (a) The sample class centroids of the ten image pattern classes from the CIFAR-10 dataset. (b) Backward-pass predictions given the class-label (unit basis) output vectors $\mathbf{e}_k$ after training with ordinary or unidirectional backpropagation. The network produced only noise-like backward-pass feedback signals $N^{\mathrm{T}}(\mathbf{e}_k)$. (c) Backward-pass predictions after training with bidirectional backpropagation and using the proper joint error in (69). The feedback signals $N^{\mathrm{T}}(\mathbf{e}_k)$ closely matched the corresponding sample class centroids.

We now show that the basic error-times-signal form of backpropagation learning holds for the forward pass to a logistic layer of K output logistic neurons. The same result holds for the backward pass to an input layer of logistic neurons. Suppose at the logistic output field F_Y we want to observe the target bit vector $\mathbf{t} \in \{0, 1\}^K$ as the neural output $\mathbf{y} = N(\mathbf{x})$. Denote the forward logistic log likelihood as L_f

$$L_f = \ln p_f(\mathbf{t}|\mathbf{x}, \Theta) \tag{59}$$

$$= \sum_{k=1}^{K} t_k \ \ln a_k^y + (1 - t_k) \ \ln\bigl(1 - a_k^y\bigr). \tag{60}$$

Note that $p_f(\mathbf{t}|\mathbf{x}, \Theta) = \exp(-E_f(\Theta))$ from (58). So minimizing the double cross entropy $E_f(\Theta)$ maximizes the forward likelihood and conversely.

The central result is that backpropagation's first main learning term $\partial L_f / \partial m_{jk}$ has the form

$$\frac{\partial L_f}{\partial m_{jk}} = \bigl(t_k - a_k^y\bigr) a_j^h \tag{61}$$

if the weight matrix M connects the network's final hidden layer to the output layer of K logistic neurons in F_Y. This hidden layer has n_h neurons with activations a_j^h. The derivation uses three partial derivatives. The first is the partial derivative of the inner-product input $o_k^y = \sum_{l=1}^{n_h} a_l^h m_{lk}$ with respect to the weight m_{jk}: $\partial o_k^y / \partial m_{jk} = a_j^h$. The second is the partial derivative of the output logistic activation a_k^y with respect to o_k^y. It has the product form in (4) with $c = 1$ for simplicity. The third is the straightforward partial derivative of L_f with respect to the output activation a_k^y. Then the result (61) follows from the chain rule and substitution for the three partial derivatives:

$$\frac{\partial L_f}{\partial m_{jk}} = \frac{\partial L_f}{\partial a_k^y} \frac{\partial a_k^y}{\partial o_k^y} \frac{\partial o_k^y}{\partial m_{jk}} \tag{62}$$

$$= \frac{\partial L_f}{\partial a_k^y} \frac{\partial a_k^y}{\partial o_k^y} a_j^h \tag{63}$$

$$= \frac{\partial L_f}{\partial a_k^y} a_k^y \bigl(1 - a_k^y\bigr) a_j^h \tag{64}$$

$$= \left[t_k \frac{1}{a_k^y} - (1 - t_k) \frac{1}{1 - a_k^y} \right] a_k^y \bigl(1 - a_k^y\bigr) a_j^h \tag{65}$$

$$= \bigl[t_k \bigl(1 - a_k^y\bigr) - (1 - t_k) a_k^y \bigr] a_j^h \tag{66}$$

$$= \bigl(t_k - a_k^y\bigr) a_j^h. \tag{67}$$

Backpropagation invariance ensures that the parameter gradient $\nabla_{\Theta^n} L$ gives the same result for all other layer likelihoods if the layer likelihoods match the probabilistic structure of the layer activations.

The deep classifier in Fig. 4 had an input layer of 3072 identity neurons. They let the input layer encode images from the CIFAR-10 dataset. The network used seven hidden layers of 512 quasilinear ReLU units. The output layer used ten softmax neurons and 1-in-K encoding with unit basis vectors $\mathbf{e}_k$.

The input layer of identity neurons had a vector-normal layer likelihood with a diagonal covariance matrix since this layer functioned as the output layer of a regressor on the backward pass. Taking the negative log likelihood gave the backward error E_b as the squared error. The classifier's 10 output softmax neurons had a cross entropy for its forward error E_f. Then the total bidirectional error $E(\Theta)$ was the sum of these directional errors

$$E(\Theta) = E_f(\Theta) + E_b(\Theta) \tag{68}$$

$$= -\sum_{k=1}^{K} y_k \ln a_k^y + \frac{1}{2} \sum_{i=1}^{I} \bigl(x_i - a_i^x\bigr)^2 \tag{69}$$

for I input identity neurons.

The second panel of Fig. 4 shows that running the unidirectionally trained classifier produced only noise on its backward pass because it trained only with E_f. The third panel shows that the bidirectionally trained classifier produced an

accurate estimate of the correct pattern-class centroid because it instead trained bidirectionally with (69). Centroids minimize the squared error of the regression. The bidirectionally trained BAM network also had higher classification accuracy. The accuracy increased further by carefully injecting EM-based (not blind) noise during its training [3].

V. Conclusion

The basic BAM theorem states that every real matrix is bidirectional stable for a two-layer neural network with threshold-like neurons. This result extends in many directions for different neural and synaptic dynamics. The new bidirectional backpropagation algorithm shows how to further extend the BAM structure to any number of hidden layers. This allows classifier or regressor networks to run in reverse through the same synaptic web and improve performance. The supervised algorithm does require far more computation and careful choice of the activation and likelihood structure of each neural layer.

References

[1] B. Kosko, "Bidirectional associative memories," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 18, no. 1, pp. 49–60, Jan./Feb. 1988.

[2] O. Adigun and B. Kosko, "Bidirectional backpropagation," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 50, no. 5, pp. 1982–1994, May 2020.

[3] O. Adigun and B. Kosko, "Noise-boosted bidirectional backpropagation and adversarial learning," *Neural Netw.*, vol. 120, pp. 9–31, Dec. 2019.

[4] Y. Bengio, "Learning deep architectures for AI," *Found. Trends Mach. Learn.*, vol. 2, no. 1, pp. 1–127, 2009.

[5] B. Kosko, K. Audhkhasi, and O. Osoba, "Noise can speed backpropagation learning and deep bidirectional pretraining," *Neural Netw.*, vol. 129, pp. 359–384, Sep. 2020.

[6] K. Gopalsamy and X.-Z. He, "Delay-independent stability in bidirectional associative memory networks," *IEEE Trans. Neural Netw.*, vol. 5, no. 6, pp. 998–1002, Nov. 1994.

[7] J. Cao and L. Wang, "Periodic oscillatory solution of bidirectional associative memory networks with delays," *Phys. Rev. E, Stat. Phys. Plasmas Fluids Relat. Interdiscip. Top.*, vol. 61, no. 2, p. 1825, 2000.

[8] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *in Proc. Nat. Acad. Sci. USA*, vol. 79, no. 8, pp. 2554–2558, 1982.

[9] B. Kosko, *Neural Networks and Fuzzy Systems*. Englewood Cliffs, NJ, USA: Prentice-Hall, 1991.

[10] L. A. Zadeh, "Fuzzy sets," *Inf. Control*, vol. 8, no. 3, pp. 338–353, 1965.

[11] B. Kosko, "Additive fuzzy systems: From generalized mixtures to rule continua," *Int. J. Intell. Syst.*, vol. 33, no. 8, pp. 1573–1623, 2018.

[12] M. A. Cohen and S. Grossberg, "Absolute stability of global pattern formation and parallel memory storage by competitive neural networks," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. SMC-13, no. 5, pp. 815–826, Sep./Oct. 1983.

[13] J. A. Anderson, "A memory storage model utilizing spatial correlation functions," *Kybernetik*, vol. 5, no. 3, pp. 113–119, 1968.

[14] T. Kohonen, "Correlation matrix memories," *IEEE Trans. Comput.*, vol. C-100, no. 4, pp. 353–359, Apr. 1972.

[15] R. V. Hogg, J. McKean, and A. T. Craig, *Introduction to Mathematical Statistics*. Harlow, U.K.: Pearson, 2013.

[16] S. Grossberg, "Nonlinear neural networks: Principles, mechanisms, and architectures," *Neural Netw.*, vol. 1, no. 1, pp. 17–61, 1988.

[17] S. Grossberg, "How does a brain build a cognitive code?" *Psychol. Rev.*, vol. 87, p. I-51, 1980.

[18] S. Grossberg, "Towards solving the hard problem of consciousness: The varieties of brain resonances and the conscious experiences that they support," *Neural Netw.*, vol. 87, pp. 38–95, Mar. 2017.

[19] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[20] M. I. Jordan and T. M. Mitchell, "Machine learning: Trends, perspectives, and prospects," *Science*, vol. 349, no. 6245, pp. 255–260, 2015.

[21] K. Audhkhasi, O. Osoba, and B. Kosko, "Noise-enhanced convolutional neural networks," *Neural Netw.*, vol. 78, pp. 15–23, Jun. 2016.

[22] P. J. Werbos, "Beyond regression: New tools for prediction and analysis in the behavioral sciences," Ph.D. dissertation, Appl. Math., Harvard Univ., Cambridge, MA, USA, 1974.

[23] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 323–533, Oct. 1986.

[24] O. Adigun and B. Kosko, "High capacity neural block classifiers with logistic neurons and random coding," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, 2020, pp. 1–9.

Bart Kosko (Fellow, IEEE) received degrees in philosophy, economics, applied mathematics, electrical engineering, and law.

He is a Professor of Electrical and Computer Engineering, and Law, a past Director of the Signal and Image Processing Institute, and a licensed attorney at the University of Southern California, Los Angeles, CA, USA. He has published the textbooks *Neural Networks and Fuzzy Systems* and *Fuzzy Engineering*, the trade books *Fuzzy Thinking*, *Heaven in a Chip*, and *Noise*, the edited volume *Neural Networks and Signal Processing*, the co-edited volume *Intelligent Signal Processing*, and the novels *Nanotime* and *Cool Earth*.

Prof. Kosko was a co-recipient of the Best Paper Award at the 2017 International Joint Conference on Neural Networks. He organized the first IEEE International Conference on Neural Networks in 1987 that inaugurated the IJCNN conference series.