Bidirectional Backpropagation

Olaoluwa Adigun, *Member, IEEE*, and Bart Kosko, *Fellow, IEEE*

***Abstract*—We extend backpropagation (BP) learning from ordinary unidirectional training to bidirectional training of deep multilayer neural networks. This gives a form of backward chaining or inverse inference from an observed network output to a candidate input that produced the output. The trained network learns a bidirectional mapping and can apply to some inverse problems. A bidirectional multilayer neural network can exactly represent some invertible functions. We prove that a fixed three-layer network can always exactly represent any finite permutation function and its inverse. The forward pass computes the permutation function value. The backward pass computes the inverse permutation with the same weights and hidden neurons. A joint forward–backward error function allows BP learning in both directions without overwriting learning in either direction. The learning applies to classification and regression. The algorithms do not require that the underlying sampled function has an inverse. A trained regression network tends to map an output back to the centroid of its preimage set.**

***Index Terms*—Backpropagation (BP) learning, backward chaining, bidirectional associative memory, function approximation, function representation, inverse problems.**

I. Bidirectional Backpropagation

We EXTEND the familiar unidirectional backpropagation (BP) algorithm [1]–[5] to the bidirectional case. Unidirectional BP maps an input vector to an output vector by passing the input vector forward through the network's visible and hidden neurons and its connection weights. Bidirectional BP (B-BP) combines this forward pass with a backward pass through the *same* neurons and weights. It does not use two separate feedforward or unidirectional networks.

B-BP training endows a multilayered neural network $N : \mathbb{R}^n \to \mathbb{R}^p$ with a form of backward inference. The forward pass gives the usual predicted neural output $N(\mathbf{x})$ given a vector input $\mathbf{x}$. The output vector value $\mathbf{y} = N(\mathbf{x})$ answers the *what-if* question that $\mathbf{x}$ poses: What would we observe if $\mathbf{x}$ occurred? What would be the effect? The backward pass answers the *why* question that $\mathbf{y}$ poses: Why did $\mathbf{y}$ occur? What type of input would cause $\mathbf{y}$? Feedback convergence to a resonating bidirectional fixed-point attractor [6], [7] gives a long-term or equilibrium answer to both the what-if and why questions. This paper does not address the global stability of multilayered bidirectional networks.

Bidirectional neural learning applies to large-scale problems and big data because the BP algorithm scales linearly with training data. BP has time complexity $O(n)$ for n training samples because both the forward and backward passes have complexity $O(n)$. So the B-BP algorithm still has $O(n)$ complexity because $O(n) + O(n) = O(n)$. This linear scaling does not hold for most machine-learning algorithms. An example is the quadratic complexity $O(n^2)$ of support-vector kernel methods [8].

We first show that multilayer bidirectional networks have sufficient power to exactly represent permutation mappings. These mappings are invertible and discrete. We then develop the B-BP algorithms that can approximate these and other mappings if the networks have enough hidden neurons.

A neural network N exactly *represents* a function f just in case $N(\mathbf{x}) = f(\mathbf{x})$ for all input vectors $\mathbf{x}$. Exact representation is much stronger than the more familiar property of function approximation: $N(\mathbf{x}) \approx f(\mathbf{x})$. Feedforward multilayer neural networks can uniformly approximate continuous functions on compact sets [9], [10]. Additive fuzzy systems are also uniform function approximators [11]. But additive fuzzy systems have the further property that they can exactly represent any real function if it is bounded [12]. This exact representation needs only two fuzzy rules because the rules absorb the function into their fuzzy sets. This holds more generally for generalized probability mixtures because the fuzzy rules define the mixed probability densities [13], [14].

Figs. 1 and 2 show bidirectional 3-layer networks of zero-threshold neurons. Both networks exactly represent the 3-bit permutation function f in Table I where $\{-, -, +\}$ denotes $\{-1, -1, 1\}$. So f is a self-bijection that rearranges the 8 vectors in the bipolar hypercube $\{-1, 1\}^3$. This f is just one of the 8! or 40 320 permutation maps or rearrangements on the bipolar hypercube $\{-1, 1\}^3$. The forward pass converts the input bipolar vector $(1, 1, 1)$ to the output bipolar vector $(-1, -1, 1)$. The backward pass converts $(-1, -1, 1)$ to $(1, 1, 1)$ over the *same* fixed synaptic connection weights. These same weights and neurons similarly convert the other 7 input vectors in the first column of Table I to the corresponding 7 output vectors in the second column and vice versa.

Theorem 1 states that a multilayer bidirectional network can exactly represent any finite bipolar or binary permutation function. This result requires a hidden layer with 2^n hidden neurons for an n-bit permutation function on the bipolar hypercube $\{-1, 1\}^n$. Fig. 3 shows such a network. Using so many hidden neurons is not practical or necessary in most real-world cases. The exact bidirectional representation in Fig. 1 uses only 4 hidden threshold neurons to represent the 3-bit permutation function. This was the smallest hidden layer that we found

Manuscript received May 18, 2017; revised September 2, 2017; accepted November 10, 2017. This paper was recommended by Associate Editor G.-B. Huang. *(Corresponding author: Bart Kosko.)*

The authors are 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 of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSMC.2019.2916096

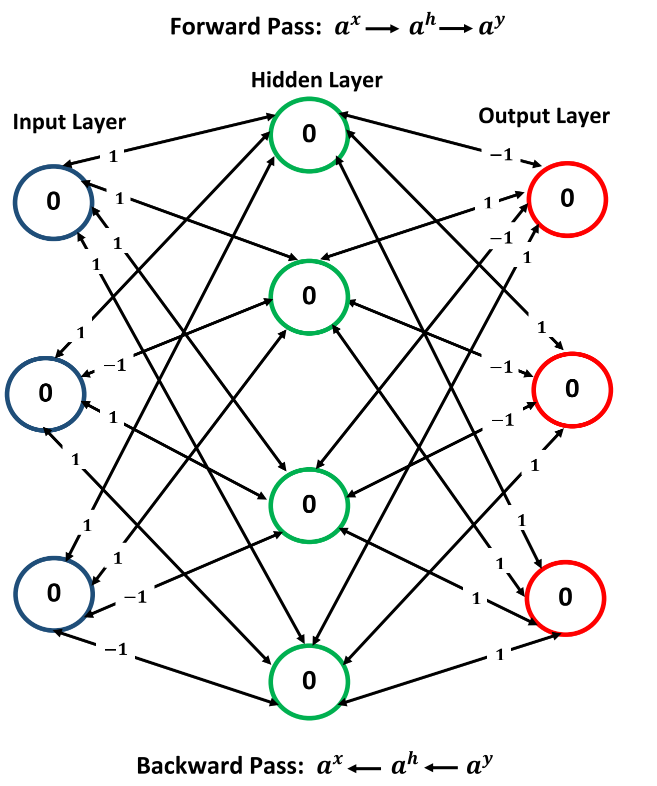

Fig. 1. Exact bidirectional representation of a permutation map. The 3-layer bidirectional threshold network exactly represents the invertible 3-bit bipolar permutation function f in Table I. The network uses four hidden neurons. The forward pass takes the input bipolar vector $\mathbf{x}$ at the input layer and feeds it forward through the weighted edges and the hidden layer of threshold neurons to the output layer. The backward pass feeds the output bipolar vector $\mathbf{y}$ back through the same weights and neurons. All neurons are bipolar and use zero thresholds. The bidirectional network computes $\mathbf{y} = f(\mathbf{x})$ on the forward pass. It computes the inverse value $f^{-1}(\mathbf{y})$ on the backward pass.

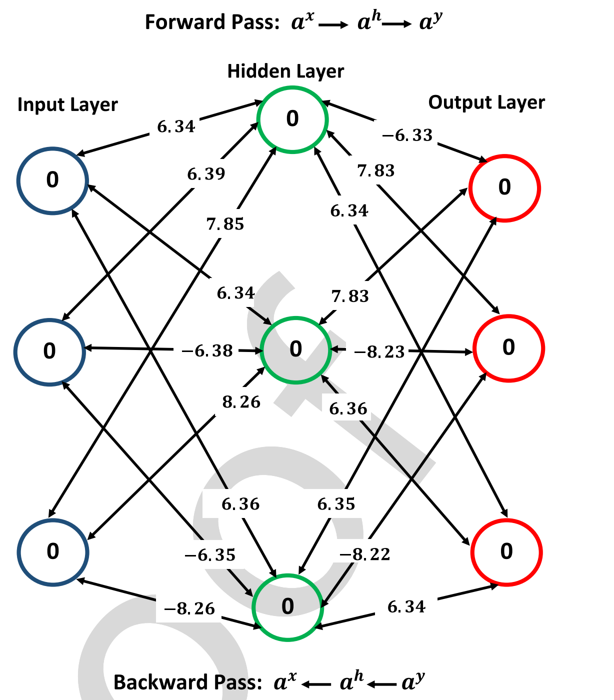

Fig. 2. Learned bidirectional representation of the 3-bit permutation in Table I. The bidirectional BP algorithm found this representation using the double-classification learning laws of Section III. It used only three hidden neurons. All the neurons were bipolar and had zero thresholds. Zero thresholding gave an exact representation of the 3-bit permutation.

through guesswork. Many other bidirectional representations also use fewer than 8 hidden neurons.

We seek instead a practical learning algorithm that can learn bidirectional approximations from sample data. Fig. 2 shows a learned bidirectional representation of the same 3-bit permutation in Table I. It uses only 3 hidden neurons. The B-BP algorithm tuned the neurons' threshold values as well as their connection weights. All the learned threshold values were near zero. We rounded them to zero to achieve the bidirectional representation with just 3 hidden neurons.

The rest of this paper derives the B-BP algorithm for regression and classification in both directions and for mixed classification–regression. This takes some care because training the weights in one direction tends to overwrite their BP training in the other direction. The B-BP algorithm solves this problem by minimizing a *joint* error function. The lone error function is cross entropy for unidirectional classification. It is squared error for unidirectional regression. Fig. 4 compares ordinary BP training and overwriting with B-BP training.

The learned approximation tends to improve if we add more hidden neurons. Fig. 5 shows that the B-BP training cross-entropy error falls as the number of hidden neurons grows when learning the 5-bit permutation in Table II.

Fig. 6 shows a deep 8-layer bidirectional approximation of the nonlinear function $f(x) = 0.5\sigma(6x + 3) + 0.5\sigma(4x - 1.2)$ and its inverse. The network used 6 hidden layers with 10 bipolar logistic neurons per layer. A bipolar logistic activation σ scales and translates an ordinary unit-interval-valued logistic

$$\sigma(x) = \frac{2}{1 + e^{-x}} - 1. \quad (1)$$

The final sections show that similar B-BP algorithms hold for training double-classification networks and mixed classification–regression networks. The B-BP learning laws are the same for regression and classification subject to these conditions: regression minimizes the squared error and uses identity output neurons. Classification minimizes the cross entropy and uses softmax output neurons. Both cases maximize the network likelihood or log-likelihood function. Logistic input and output neurons give the same B-BP learning laws if the network minimizes the bipolar cross entropy in (114). We call this *backpropagation invariance*.

B-BP learning also approximates noninvertible functions. The algorithm tends to learn the centroid of many-to-one functions. Suppose that the target function $f : \mathbb{R}^n \to \mathbb{R}^p$ is not one-to-one or injective. So it has no inverse f^{-1} point mapping. But it does have a *set-valued* inverse or preimage pullback mapping $f^{-1} : 2^{\mathbb{R}^p} \to 2^{\mathbb{R}^n}$ such that $f^{-1}(B) = \{x \in \mathbb{R}^n : f(x) \in B\}$ for any $B \subset \mathbb{R}^p$. Suppose that the n input training samples $x_1, \ldots, x_n$ map to the same output training sample $y : f^{-1}(\{y\}) = \{x_1, \ldots, x_n\}$. Then B-BP learning tends to map y to the centroid $\bar{x}$ of $f^{-1}(\{y\})$ because the centroid minimizes the mean-squared error of regression.

TABLE I
3-BIT BIPOLAR PERMUTATION FUNCTION f

Input x	Output t
$[+ + +]$	$[- - +]$
$[+ + -]$	$[- + +]$
$[+ - +]$	$[+ + +]$
$[+ - -]$	$[+ - +]$
$[- + +]$	$[- + -]$
$[- + -]$	$[- - -]$
$[- - +]$	$[+ - -]$
$[- - -]$	$[+ + -]$

Fig. 7 shows such an approximation for the noninvertible target function $f(x) = \sin x$. The forward regression approximates $\sin x$. The backward regression approximates the average or centroid of the two points in the preimage set of $y = \sin x$. Then $f^{-1}(\{y\}) = \sin^{-1}(y) = \{\theta, \pi - \theta\}$ for $0 < \theta < (\pi/2)$ if $0 < y < 1$. This gives the pullback's centroid as $(\pi/2)$. The centroid equals $-(\pi/2)$ if $-1 < y < 0$.

B-BP differs from earlier neural approaches to approximating inverses. Hwang *et al.* [15] developed an inverse algorithm for query-based learning in binary classification. Their BP-based algorithm is not bidirectional. It instead exploits the data-weight inner-product input to neurons. It holds the weights constant while it tunes the data for a given output. Saad *et al.* [16], [17] have applied this inverse algorithm to problems in aerospace and elsewhere. B-BP also differs from the more recent bidirectional extreme-learning-machine algorithm that uses a two-stage learning process but in a unidirectional network [18].

II. BIDIRECTIONAL EXACT REPRESENTATION OF BIPOLAR PERMUTATIONS

This section proves that there exist multilayered neural networks that can exactly bidirectionally represent some invertible functions. We first define the network variables. The proof uses threshold neurons. The B-BP algorithms below use soft-threshold logistic sigmoids for hidden neurons.

A bidirectional neural network is a multilayer network $N : X \rightarrow Y$ that maps the input space X to the output space Y *and conversely* through the same set of weights. The backward pass uses the matrix transposes of the weight matrices that the forward pass uses. Such a network is a bidirectional associative memory or BAM [6], [7]. The original BAM theorem [6] states that any *two*-layer neural network is globally bidirectionally stable for any sole rectangular weight matrix $\mathbf{W}$ with real entries.

The forward pass sends the input vector $\mathbf{x}$ through the weight matrix $\mathbf{W}$ that connects the input layer to the hidden layer. The result passes on through matrix $\mathbf{U}$ to the output layer. The backward pass sends the output $\mathbf{y}$ from the output layer back through the hidden layer to the input layer. Let $I, J,$ and K denote the respective numbers of input, hidden, and output neurons. Then the $I \times J$ matrix $\mathbf{W}$ connects the input layer to the hidden. The $J \times K$ matrix $\mathbf{U}$ connects the hidden layer to the output layer.

The hidden-neuron input o_j^h has the affine form

$$o_j^h = \sum_{i=1}^{I} w_{ij} a_i^x(x_i) + b_j^h \tag{2}$$

where weight w_{ij} connects the ith input neuron to the jth hidden neuron, a_i^x is the activation of the ith input neuron, and b_j^h is the bias of the jth hidden neuron. The activation a_j^h of the jth hidden neuron is a bipolar threshold

$$a_j^h\left(o_j^h\right) = \begin{cases} -1 & \text{if } o_j^h \leq 0 \\ 1 & \text{if } o_j^h > 0. \end{cases} \tag{3}$$

The B-BP algorithm in the next section uses soft-threshold bipolar logistic functions for the hidden activations because such sigmoid functions are differentiable. The proof below also modifies the hidden thresholds to take on binary values in (14) and to fire with a slightly different condition.

The input o_k^y to the kth output neuron from the hidden layer is also affine

$$o_k^y = \sum_{j=1}^{J} u_{jk} a_j^h + b_k^y \tag{4}$$

where weight u_{jk} connects the jth hidden neuron to the kth output neuron. Term b_k^y is the additive bias of the kth output neuron. The output activation vector $\mathbf{a}^y$ gives the predicted outcome or target on the forward pass. The kth output neuron has bipolar threshold activation a_k^y

$$a_k^y(o_k^y) = \begin{cases} -1 & \text{if } o_k^y \leq 0 \\ 1 & \text{if } o_k^y > 0. \end{cases} \tag{5}$$

The forward pass of an input bipolar vector $\mathbf{x}$ from Table I through the network in Fig. 1 gives an output activation vector $\mathbf{a}^y$ that equals the table's corresponding target vector $\mathbf{y}$. The backward pass feeds $\mathbf{y}$ from the output layer back through the hidden layer to the input layer. Then the backward-pass input o_j^{hb} to the jth hidden neuron is

$$o_j^{hb} = \sum_{k=1}^{K} u_{jk} a_k^y(y_k) + b_j^h \tag{6}$$

where y_k is the output of the kth output neuron. The term a_k^y is the activation of the kth output neuron. The backward-pass activation of the jth hidden neuron a_j^{hb} is

$$a_j^{hb}\left(o_j^{hb}\right) = \begin{cases} -1 & \text{if } o_j^{hb} \leq 0 \\ 1 & \text{if } o_j^{hb} > 0. \end{cases} \tag{7}$$

The backward-pass input o_i^{xb} to the ith input neuron is

$$o_i^{xb} = \sum_{j=1}^{J} w_{ij} a_j^{hb} + b_i^x \tag{8}$$

where b_i^x is the bias for the ith input neuron. The input-layer activation $\mathbf{a}^x$ gives the predicted value for the backward pass. The ith input neuron has bipolar activation

$$a_i^{xb}\left(o_i^{xb}\right) = \begin{cases} -1 & \text{if } o_i^{xb} \leq 0 \\ 1 & \text{if } o_i^{xb} > 0. \end{cases} \tag{9}$$

We can now state and prove the bidirectional representation theorem for bipolar permutations. The theorem also applies to binary permutations because the input and output neurons have bipolar threshold activations.

Theorem 1 (Exact Bidirectional Representation of Bipolar Permutation Functions): Suppose that the invertible function $f : \{-1, 1\}^n \to \{-1, 1\}^n$ is a permutation. Then there exists a 3-layer bidirectional neural network $N : \{-1, 1\}^n \to \{-1, 1\}^n$ that exactly represents f in the sense that $N(\mathbf{x}) = f(\mathbf{x})$ and that $N^{-1}(\mathbf{x}) = f^{-1}(\mathbf{x})$ for all $\mathbf{x}$. The hidden layer has 2^n threshold neurons.

Proof: The proof constructs weight matrices $\mathbf{W}$ and $\mathbf{U}$ so that exactly one hidden neuron fires on both the forward and the backward passes. Fig. 3 shows the proof technique for the special case of a 3-bit bipolar permutation. We structure the network so that an input vector $\mathbf{x}$ fires only one hidden neuron on the forward pass. The output vector $\mathbf{y} = \mathbf{N}(\mathbf{x})$ fires only the same hidden neuron on the backward pass.

The bipolar permutation f is a bijective map of the bipolar hypercube $\{-1, 1\}^n$ onto itself. The bipolar hypercube contains the 2^n input bipolar column vectors $\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_{2^n}}$. It likewise contains the 2^n output bipolar vectors $\mathbf{y_1}, \mathbf{y_2}, \ldots, \mathbf{y_{2^n}}$. The network uses 2^n corresponding hidden threshold neurons. So $J = 2^n$.

Matrix $\mathbf{W}$ connects the input layer to the hidden layer. Matrix $\mathbf{U}$ connects the hidden layer to the output layer. Define $\mathbf{W}$ so that its columns list all 2^n bipolar input vectors. Define $\mathbf{U}$ so that the columns of its transpose $\mathbf{U}^{\mathrm{T}}$ list all 2^n transposed bipolar output vectors:

$$\mathbf{W} = \begin{bmatrix}\mathbf{x_1} & \mathbf{x_2} & \ldots & \mathbf{x_{2^n}}\end{bmatrix}$$
$$\mathbf{U}^{\mathrm{T}} = \begin{bmatrix}\mathbf{y_1} & \mathbf{y_2} & \ldots & \mathbf{y_{2^n}}\end{bmatrix}.$$

We show next both that these weight matrices fire only one hidden neuron and that the forward pass of any input vector $\mathbf{x_n}$ gives the corresponding output vector $\mathbf{y_n}$. Assume that each neuron has zero bias.

Pick a bipolar input vector $\mathbf{x}_m$ for the forward pass. Then the input activation vector $\mathbf{a}^x(\mathbf{x}_m) = (a_1^x(x_m^1), \ldots, a_n^x(x_m^n))$ equals the input bipolar vector $\mathbf{x}_m$ because the input activations (9) are bipolar threshold functions with zero threshold. So $\mathbf{a}^x$ equals $\mathbf{x}_m$ because the vector space is bipolar $\{-1, 1\}^n$.

The hidden layer input $\mathbf{o}^h$ is the same as (2). It has the matrix-vector form

$$\mathbf{o}^h = \mathbf{W}^{\mathrm{T}}\mathbf{a}^x \tag{10}$$
$$= \mathbf{W}^{\mathrm{T}}\mathbf{x}_m \tag{11}$$
$$= \left(o_1^h, o_2^h, \ldots, o_n^h, \ldots, o_{2^n}^h\right)^{\mathrm{T}} \tag{12}$$
$$= \left(\mathbf{x}_1^{\mathrm{T}}\mathbf{x}_m, \mathbf{x}_2^{\mathrm{T}}\mathbf{x}_m, \ldots, \mathbf{x}_j^{\mathrm{T}}\mathbf{x}_m, \ldots, \mathbf{x}_{2^n}^{\mathrm{T}}\mathbf{x}_m\right)^{\mathrm{T}} \tag{13}$$

since o_j^h is the inner product of the bipolar vectors $\mathbf{x}_j$ and $\mathbf{x}_m$ from the definition of $\mathbf{W}$.

The input o_j^h to the jth neuron of the hidden layer obeys $o_j^h = n$ when $j = m$. It obeys $o_j^h < n$ when $j \neq m$. This holds because the vectors $\mathbf{x}_j$ are bipolar with scalar components in $\{-1, 1\}$. The magnitude of a bipolar vector in $\{-1, 1\}^n$ is $\sqrt{n}$. The inner product $\mathbf{x}_j^{\mathrm{T}}\mathbf{x}_m$ is a maximum when both vectors have the same direction. This occurs when $j = m$. The inner product is otherwise less than n. Fig. 3 shows a bidirectional neural network that fires just the sixth hidden neuron. The weights for the network in Fig. 3 are

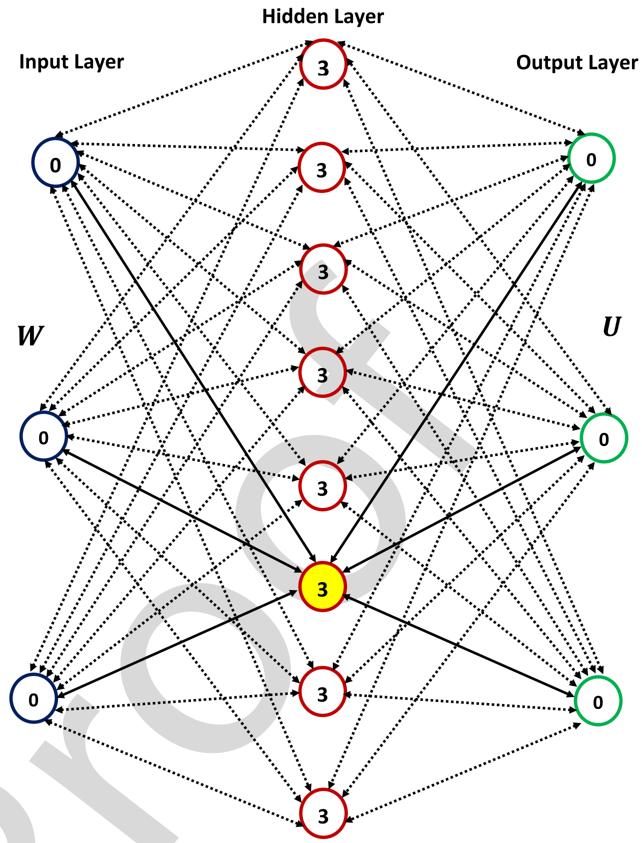

Fig. 3. Bidirectional network structure for the proof of Theorem 1. The input and output layers have n threshold neurons. The hidden layer has 2^n neurons with threshold values of n. The 8 fan-in 3-vectors of weights in $\mathbf{W}$ from the input to the hidden layer list the 2^3 elements of the bipolar cube $\{-1, 1\}^3$. So they list the eight vectors in the input column of Table I. The 8 fan-in 3-vectors of weights in $\mathbf{U}$ from the output to the hidden layer list the eight bipolar vectors in the output column of Table I. The threshold value for the sixth and highlighted hidden neuron is 3. Passing the sixth input vector $(-1, 1, -1)$ through $\mathbf{W}$ leads to the hidden-layer vector $(0, 0, 0, 0, 0, 1, 0, 0)$ of thresholded values. Passing this 8-bit vector through $\mathbf{U}$ produces after thresholding the sixth output vector $(-1, -1, -1)$ in Table I. Passing this output vector back through the transpose of $\mathbf{U}$ produces the same unit bit vector of thresholded hidden-unit values. Passing this vector back through the transpose of $\mathbf{W}$ produces the original bipolar vector $(-1, 1, -1)$.

$$\mathbf{W} = \begin{bmatrix} 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \end{bmatrix}$$

$$\mathbf{U}^T = \begin{bmatrix} -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 \\ -1 & 1 & 1 & -1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \end{bmatrix}.$$

Now comes the key step in the proof. Define the hidden activation a_j^h as a *binary* (not bipolar) threshold function where n is the threshold value

$$a_j^h\left(o_j^h\right) = \begin{cases} 1 & \text{if } o_j^h \geq n \\ 0 & \text{if } o_j^h < n. \end{cases} \tag{14}$$

Then the hidden-layer activation $\mathbf{a}^h$ is the *unit* bit vector $(0, 0, \ldots, 1, \ldots, 0)^{\mathrm{T}}$, where $a_j^h = 1$ when $j = m$ and where $a_j^h = 0$ when $j \neq m$. This holds because all 2^n bipolar vectors $\mathbf{x}_m$ in $\{-1, 1\}^n$ are distinct. So exactly one of these 2^n vectors achieves the maximal inner-product value $n = \mathbf{x}_m^{\mathrm{T}}\mathbf{x}_m$. So $a_j^h(o_j^h) = 0$ for $j \neq m$ and $a_m^h(o_m^h) = 1$. The bidirectional network in Fig. 3 represents the 3-bit bipolar permutation in Table I.

The input vector $\mathbf{o}^y$ to the output layer is

$$\mathbf{o}^y = \mathbf{U}^{\mathrm{T}}\mathbf{a}^h \tag{15}$$

$$= \sum_{j=1}^{J} \mathbf{y}_j \ a_j^h \tag{16}$$

$$= \mathbf{y}_m \tag{17}$$

where a_j^h is the activation of the jth hidden neuron. The activation $\mathbf{a}^y$ of the output layer is

$$\mathbf{a}^y\left(o_j^y\right) = \begin{cases} 1 & \text{if} \quad o_j^y \geq 0 \\ -1 & \text{if} \quad o_j^y < 0. \end{cases} \tag{18}$$

The output layer activation leaves $\mathbf{o}^y$ unchanged because $\mathbf{o}^y$ equals $\mathbf{y}_m$ and because $\mathbf{y}_m$ is a vector in $\{-1, 1\}^n$. So

$$\mathbf{a}^y = \mathbf{y}_m. \tag{19}$$

So the forward pass of an input vector $\mathbf{x}_m$ through the network yields the desired corresponding output vector $\mathbf{y}_m$ if $\mathbf{y}_m = f(\mathbf{x}_m)$ for the bipolar permutation map f.

Consider next the backward pass through the network N. The backward pass propagates the output vector $\mathbf{y}_m$ through the hidden layer back to the input layer. The hidden layer input $\mathbf{o}^{hb}$ has the same inner-product form as in (6):

$$\mathbf{o}^{hb} = \mathbf{U} \ \mathbf{y}_m \tag{20}$$

where $\mathbf{o}^{hb} = (\mathbf{y}_1^{\mathrm{T}}\mathbf{y}_m, \mathbf{y}_2^{\mathrm{T}}\mathbf{y}_m, \ldots, \mathbf{y}_j^{\mathrm{T}}\mathbf{y}_m, \ldots, \mathbf{y}_{2^n}^{\mathrm{T}}\mathbf{y}_m)^{\mathrm{T}}$.

The input o_j^{hb} of the jth neuron in the hidden layer equals the inner product of $\mathbf{y}_j$ and $\mathbf{y}_m$. So $o_j^{hb} = n$ when $j = m$. But now $o_j^{hb} < n$ when $j \neq m$. This holds because again the magnitude of a bipolar vector in $\{-1, 1\}^n$ is $\sqrt{n}$. The inner product o_j^{hb} is a maximum when vectors $\mathbf{y}_m$ and $\mathbf{y}_j$ lie in the same direction. The activation $\mathbf{a}^{hb}$ for the hidden layer has the same components as in (14). So the hidden-layer activation $\mathbf{a}^{hb}$ again equals the unit bit vector $(0, 0, \ldots, 1, \ldots, 0)^{\mathrm{T}}$ where $a_j^{hb} = 1$ when $j = m$ and $a_j^{hb} = 0$ when $j \neq m$.

Then the input vector $\mathbf{o}^{xb}$ for the input layer is

$$\mathbf{o}^{xb} = \mathbf{W} \ \mathbf{a}^{hb} \tag{21}$$

$$= \sum_{j=1}^{J} \mathbf{x}_j \ \mathbf{a}^{hb} \tag{22}$$

$$= \mathbf{x}_m. \tag{23}$$

The ith input neuron has a threshold activation that is the same as

$$a_i^{xb}\left(o_i^{xb}\right) = \begin{cases} 1 & \text{if} \quad o_i^{xb} \geq 0 \\ -1 & \text{if} \quad o_i^{xb} < 0 \end{cases} \tag{24}$$

where o_i^{xb} is the input of ith neuron in the input layer. This activation leaves $\mathbf{o}^{xb}$ unchanged because $\mathbf{o}^{xb}$ equals $\mathbf{x}_m$ and because the vector $\mathbf{x}_m$ lies in $\{-1, 1\}^n$. So

$$\mathbf{a}^{xb} = \mathbf{o}^{xb} \tag{25}$$

$$= \mathbf{x}_m. \tag{26}$$

So the backward pass of any target vector $\mathbf{y}_m$ yields the desired input vector $\mathbf{x}_m$ if $f^{-1}(\mathbf{y}_m) = \mathbf{x}_m$. This completes the backward pass and the proof. ■

III. Bidirectional Backpropagation Algorithms

A. Double Regression

We now derive the first of three B-BP learning algorithms. The first case is double regression where the network performs regression in both directions.

B-BP training minimizes both the forward error E_f and backward error E_b. B-BP alternates between backward training and forward training. Forward training minimizes E_f while holding E_b constant. Backward training minimizes E_b while holding E_f constant. E_f is the error at the output layer. E_b is the error at the input layer. Double regression uses squared error for both error functions.

The forward pass sends the input vector $\mathbf{x}$ through the hidden layer to the output layer. The network uses only one hidden layer for simplicity and with no loss of generality. The B-BP double-regression algorithm applies to any number of hidden layers in a deep network.

The hidden-layer input values o_j^h are the same as in (2). The jth hidden activation a_j^h is the binary logistic map

$$a_j^h\left(o_j^h\right) = \frac{1}{1 + e^{-o_j^h}} \tag{27}$$

where (4) gives the input o_k^y to the kth output neuron. The hidden activations can be logistic or any other sigmoidal function so long as they are differentiable. The activation for an output neuron is the identity function

$$a_k^y = o_k^y \tag{28}$$

where a_k^y is the activation of kth output neuron.

The error function E_f for the forward pass is squared error

$$E_f = \frac{1}{2}\sum_{k=1}^{K}\left(y_k - a_k^y\right)^2 \tag{29}$$

where y_k denotes the value of the kth neuron in the output layer. Ordinary unidirectional BP updates the weights and other network parameters by propagating the error from the output layer back to the input layer.

The backward pass sends the output vector $\mathbf{y}$ through the hidden layer to the input layer. The input to the jth hidden neuron o_j^{hb} is the same as in (6). The activation a_j^{hb} for the jth hidden neuron is

$$a_j^{hb} = \frac{1}{1 + e^{-o_j^{hb}}}. \tag{30}$$

The input o_i^x for the ith input neuron is the same as (8). The activation at the input layer is the identity function

$$a_i^{xb}\left(o_i^{xb}\right) = o_i^{xb}. \tag{31}$$

A nonlinear sigmoid (or Gaussian) activation can replace the linear function.

The backward-pass error E_b is also squared error

$$E_b = \frac{1}{2}\sum_{i=1}^{I}\left(x_i - a_i^x\right)^2. \tag{32}$$

The partial derivative of the hidden-layer activation in the forward direction is

$$\frac{\partial a_j^h}{\partial o_j^h} = \frac{\partial}{\partial o_j^h}\left(\frac{1}{1+e^{-o_j^h}}\right) \tag{33}$$
$$= \frac{e^{-o_j^h}}{\left(1+e^{-o_j^h}\right)^2} \tag{34}$$
$$= \frac{1}{1+e^{-o_j^h}}\left[1 - \frac{1}{1+e^{-o_j^h}}\right] \tag{35}$$
$$= a_j^h\left(1 - a_j^h\right). \tag{36}$$

Let $a_j^{h'}$ denote the derivative of a_j^h with respect to the inner-product term o_j^h. We again use the superscript b to denote the backward pass.

The partial derivative of E_f with respect to the weight u_{jk} is

$$\frac{\partial E_f}{\partial u_{jk}} = \frac{1}{2}\frac{\partial}{\partial u_{jk}}\sum_{k=1}^{K}\left(y_k - a_k^y\right)^2 \tag{37}$$
$$= \frac{\partial E_f}{\partial a_k^y}\frac{\partial a_k^y}{\partial o_k^y}\frac{\partial o_k^y}{\partial u_{jk}} \tag{38}$$
$$= \left(a_k^y - y_k\right)a_j^h. \tag{39}$$

The partial derivative of E_f with respect to w_{ij} is

$$\frac{\partial E_f}{\partial w_{ij}} = \frac{1}{2}\frac{\partial}{\partial w_{ij}}\sum_{k=1}^{K}\left(y_k - a_k^y\right)^2 \tag{40}$$
$$= \left(\sum_{k=1}^{K}\frac{\partial E_f}{\partial a_k^y}\frac{\partial a_k^y}{\partial o_k^y}\frac{\partial o_k^y}{\partial a_j^h}\right)\frac{\partial a_j^h}{\partial o_j^h}\frac{\partial o_j^h}{\partial w_{ij}} \tag{41}$$
$$= \sum_{k=1}^{K}\left(a_k^y - y_k\right)u_{jk}\; a_j^{h'}\; x_i \tag{42}$$

where $a_j^{h'}$ is the same as in (36). The partial derivative of E_f with respect to the bias b_k^y of the kth output neuron is

$$\frac{\partial E_f}{\partial b_k^y} = \frac{1}{2}\frac{\partial}{\partial b_k^y}\sum_{k=1}^{K}\left(y_k - a_k^y\right)^2 \tag{43}$$
$$= \frac{\partial E_f}{\partial a_k^y}\frac{\partial a_k^y}{\partial o_k^y}\frac{\partial o_k^y}{\partial b_k^y} \tag{44}$$
$$= a_k^y - y_k. \tag{45}$$

The partial derivative of E_f with respect to the bias b_j^h of the jth hidden neuron is

$$\frac{\partial E_f}{\partial b_j^h} = \frac{1}{2}\frac{\partial}{\partial b_j^h}\sum_{k=1}^{K}\left(y_k - a_k^y\right)^2 \tag{46}$$
$$= \left(\sum_{k=1}^{K}\frac{\partial E_f}{\partial a_k^y}\frac{\partial a_k^y}{\partial o_k^y}\frac{\partial o_k^y}{\partial a_j^h}\right)\frac{\partial a_j^h}{\partial o_j^h}\frac{\partial o_j^h}{\partial b_j^h} \tag{47}$$
$$= \sum_{k=1}^{K}\left(a_k^y - y_k\right)u_{jk}a_j^{h'} \tag{48}$$

where $a_j^{h'}$ is the same as in (36).

The partial derivative of the hidden-layer activation a_j^{hb} in the backward direction is

$$\frac{\partial a_j^{hb}}{\partial o_j^{hb}} = \frac{\partial}{\partial o_j^{hb}}\left(\frac{1}{1+e^{-o_j^{hb}}}\right) \tag{49}$$
$$= \frac{e^{-o_j^{hb}}}{\left(1+e^{-o_j^{hb}}\right)^2} \tag{50}$$
$$= \frac{1}{1+e^{-o_j^{hb}}}\left[1 - \frac{1}{1+e^{-o_j^{hb}}}\right] \tag{51}$$
$$= a_j^{hb}\left(1 - a_j^{hb}\right). \tag{52}$$

The partial derivative of E_b with respect to w_{ij} is

$$\frac{\partial E_b}{\partial w_{ij}} = \frac{1}{2}\frac{\partial}{\partial w_{ij}}\sum_{k=1}^{K}\left(x_i - a_i^{xb}\right)^2 \tag{53}$$
$$= \frac{\partial E_b}{\partial a_i^{xb}}\frac{\partial a_i^{xb}}{\partial o_i^{xb}}\frac{\partial o_i^{xb}}{\partial w_{ij}} \tag{54}$$
$$= \left(a_i^{xb} - x_i\right)a_j^{hb}. \tag{55}$$

The partial derivative of E_b with respect to u_{jk} is

$$\frac{\partial E_b}{\partial u_{jk}} = \frac{1}{2}\frac{\partial}{\partial u_{jk}}\sum_{i=1}^{I}\left(x_i - a_i^{xb}\right)^2 \tag{56}$$
$$= \left(\sum_{i=1}^{I}\frac{\partial E_b}{\partial a_i^{xb}}\frac{\partial a_i^{xb}}{\partial o_i^{xb}}\frac{\partial o_i^{xb}}{\partial a_j^{hb}}\right)\frac{\partial a_j^{hb}}{\partial o_j^{hb}}\frac{\partial o_j^{hb}}{\partial u_{jk}} \tag{57}$$
$$= \sum_{i=1}^{I}\left(a_i^{xb} - x_i\right)w_{ij}a_j^{hb'}y_k \tag{58}$$

where $a_j^{hb'}$ is the same as in (52).

The partial derivative of E_b with respect to the bias b_i^x of ith input neuron is

$$\frac{\partial E_b}{\partial b_i^x} = \frac{1}{2}\frac{\partial}{\partial b_i^x}\sum_{i=1}^{I}\left(x_i - a_i^{xb}\right)^2 \tag{59}$$
$$= \frac{\partial E_b}{\partial a_i^{xb}}\frac{\partial a_i^{xb}}{\partial o_i^{xb}}\frac{\partial o_i^{xb}}{\partial b_i^x} \tag{60}$$
$$= a_i^{xb} - x_i. \tag{61}$$

The partial derivative of E_b with respect to the bias b_j^h of jth hidden neuron is

$$\frac{\partial E_b}{\partial b_j^h} = \frac{1}{2}\frac{\partial}{\partial b_j^h}\sum_{i=1}^{I}\left(x_i - a_i^{xb}\right)^2 \tag{62}$$

$$= \left(\sum_{i=1}^{I}\frac{\partial E_b}{\partial a_i^{xb}}\frac{\partial a_i^{xb}}{\partial o_i^{xb}}\frac{\partial o_i^{xb}}{\partial a_j^{hb}}\right)\frac{\partial a_j^{hb}}{\partial o_j^{hb}}\frac{\partial o_j^{hb}}{\partial b_j^h} \tag{63}$$

$$= \sum_{i=1}^{I}\left(a_i^{xb} - x_i\right)w_{ij}a_j^{hb'} \tag{64}$$

where $a_j^{hb'}$ is the same as in (52).

The error function at the input layer is the backward-pass error E_b. The error function at the output layer is the forward-pass error E_f.

The above update laws for forward regression have the final form (for learning rate $\eta > 0$)

$$u_{jk}^{(n+1)} = u_{jk}^{(n)} - \eta\left(a_k^y - y_k\right)a_j^h \tag{65}$$

$$w_{ij}^{(n+1)} = w_{ij}^{(n)} - \eta\left(\sum_{k=1}^{K}\left(a_k^y - y_k\right)u_{jk}a_j^{h'}x_i\right) \tag{66}$$

$$b_j^{h(n+1)} = b_j^{h(n)} - \eta\left(\sum_{k=1}^{K}\left(a_k^y - y_k\right)u_{jk}a_j^{h'}\right) \tag{67}$$

$$b_k^{y(n+1)} = b_k^{y(n)} - \eta\left(a_k^y - y_k\right). \tag{68}$$

The dual update laws for backward regression have the final form

$$u_{jk}^{(n+1)} = u_{jk}^{(n)} - \eta\left(\sum_{i=1}^{I}\left(a_i^{xb} - x_i\right)w_{ij}a_j^{hb'}y_k\right) \tag{69}$$

$$w_{ij}^{(n+1)} = w_{ij}^{(n)} - \eta\left(a_i^{xb} - x_i\right)a_j^{hb} \tag{70}$$

$$b_i^{x(n+1)} = b_i^{x(n)} - \eta\left(a_i^{xb} - x_i\right) \tag{71}$$

$$b_j^{h(n+1)} = b_j^{h(n)} - \eta\left(\sum_{i=1}^{I}\left(a_i^{xb} - x_i\right)w_{ij}a_j^{hb'}\right). \tag{72}$$

B-BP training minimizes E_f while holding E_b constant. It then minimizes E_b while holding E_f constant. Equations (65)–(68) state the update rules for forward training. Equations (69)–(72) state the update rules for backward training. Each training iteration involves forward training and then backward training.

Algorithm 1 summarizes the B-BP algorithm. It shows how to combine forward and backward training in B-BP. Fig. 6 shows how double-regression B-BP approximates the invertible function $f(x) = 0.5\sigma(6x+3) + 0.5\sigma(4x-1.2)$ if $\sigma(x)$ denotes the bipolar logistic function in (1). The approximation used a deep 8-layer network with six layers of ten bipolar logistic neurons each. The input and output layer each contained only a single identity neuron.

B. Double Classification

We now derive a B-BP algorithm where the network's forward pass acts as a classifier network and so does its backward pass. We call this double classification.

We present the derivation in terms of cross entropy for the sake of simplicity. Our double-classification simulations used the slightly more general form of cross entropy in (114) that we call *logistic* cross entropy. The simpler cross-entropy derivation applies to softmax input neurons and output neurons (with implied 1-in-K coding). Logistic input and output neurons require logistic cross entropy for the same BP derivation because then the same final BP partial derivatives result.

The simplest double-classification network uses Gibbs or softmax neurons at both the input and output layers. This creates a winner-take-all structure at those layers. Then the kth softmax neuron in the output layer codes for the kth input pattern. The output layer represents the pattern as a K-length unit bit vector with a "1" in the kth slot and a "0" in the other $K-1$ slots [3], [19]. The same 1-in-I binary encoding holds for the ith neuron at the input layer. The softmax structure implies that the input and output fields each compute a discrete probability distribution for each input.

Classification networks differ from regression networks in another key aspect: they do not minimize squared error. They instead minimize the *cross entropy* of the given target vector and the softmax activation values of the output or input layers [3]. Equation (79) states the forward cross entropy at the output layer if y_k is the desired or target value of the kth output neuron. Then a_k^y is its actual softmax activation value. The entropy structure applies because both the target vector and the input and output vectors are probability vectors. Minimizing the cross entropy maximizes the Kullback–Leibler divergence [20] and vice versa [19].

The classification BP algorithm depends on another optimization equivalence: minimizing the cross entropy is equivalent to maximizing the network's likelihood or log-likelihood [19]. We will establish this equivalence because it implies that the *BP learning laws have the same form for both classification and regression*. We will prove the equivalence for only the forward direction. It applies equally in the backward direction. The result unifies the BP learning laws. It also allows carefully selected noise to enhance the network likelihood because BP is a special case [19], [21] of the expectation–maximization algorithm for iteratively maximizing a likelihood with missing data or hidden variables [22].

Denote the network's forward probability density function as $p_f(\mathbf{y}|\mathbf{x}, \Theta)$. The vector Θ lists all parameters in the network. The input vector $\mathbf{x}$ passes through the multilayer network and produces the output vector $\mathbf{y}$. Then the network's forward likelihood $L_f(\Theta)$ is the natural logarithm of the forward network probability: $L_f(\Theta) = \ln p_f(\mathbf{y}|\mathbf{x}, \Theta)$.

We will show that $p_f(\mathbf{y}|\mathbf{x}, \Theta) = \exp\{-E_f(\Theta)\}$. So BP's forward pass computes the forward cross entropy as it maximizes the likelihood [19].

The key assumption is that output softmax neurons in a classifier network are independent because there are no intralayer connections among them. Then the network probability density $p_f(\mathbf{y}|\mathbf{x}, \Theta)$ factors into a product of K-many marginals [3]: $p_f(\mathbf{y}|\mathbf{x}, \Theta) = \prod_{k=1}^{K} p_f(y_k|\mathbf{x}, \Theta)$. This gives

$$L_f(\Theta) = \ln p_f(\mathbf{y}|\mathbf{x}, \Theta) \tag{73}$$

$$= \ln \prod_{k=1}^{K} p_f(y_k|\mathbf{x}, \Theta) \tag{74}$$

$$= \ln \prod_{k=1}^{K} \left(a_k^y\right)^{y_k} \tag{75}$$

$$= \sum_{k=1}^{K} y_k \ln a_k^y \tag{76}$$

$$= -E_f(\Theta) \tag{77}$$

from (79) since $\mathbf{y}$ is a 1-in-K-encoded unit bit vector. Then exponentiation gives $p_f(\mathbf{y}|\mathbf{x}, \Theta) = \exp\{-E_f(\Theta)\}$. Minimizing the forward cross entropy E_f is equivalent to maximizing the negative cross entropy $-E_f$. So minimizing E_f maximizes the forward network likelihood L and vice versa.

The third equality (75) holds because the kth marginal factor $p_f(y_k|\mathbf{x}, \Theta)$ in a classifier network equals the exponentiated softmax activation $(a_k^t)^{y_k}$. This holds because $y_k = 1$ if k is the correct class label for the input pattern $\mathbf{x}$ and $y_k = 0$ otherwise. This discrete probability vector defines an output categorical distribution. It is a single-sample multinomial.

We now derive the B-BP algorithm for double classification. The algorithm minimizes the error functions separately where $E_f(\Theta)$ is the forward cross entropy in (75) and $E_b(\Theta)$ is the backward cross entropy in (81). We first derive the forward B-BP classifier algorithm. We then derive the backward portion of the B-BP double-classification algorithm.

The forward pass sends the input vector $\mathbf{x}$ through the hidden layer or layers to the output layer. The input activation vector $\mathbf{a}^x$ is the vector $\mathbf{x}$.

We assume only one hidden layer for simplicity. The derivation applies to deep networks with any number of hidden layers. The input to the jth hidden neuron o_j^h has the same linear form as in (2). The jth hidden activation a_j^h is the same ordinary unit-interval-valued logistic function in (27). The input o_k^y to the kth output neuron is the same as in (4). The hidden activations can also be ReLU or hyperbolic tangents or many other functions.

The forward classifier's output-layer neurons use Gibbs or softmax activations

$$a_k^y = \frac{e^{(o_k^y)}}{\sum_{l=1}^{K} e^{(o_l^y)}} \tag{78}$$

where a_k^y is the activation of the kth output neuron. Then the forward error E_f is the cross entropy

$$E_f = -\sum_{k=1}^{K} y_k \ln \ a_k^y \tag{79}$$

between the binary target values y_k and the actual output activations a_k^y.

We next describe the backward pass through the classifier network. The backward pass sends the output target vector $\mathbf{y}$ through the hidden layer to the input layer. So the initial activation vector $\mathbf{a^y}$ equals the target vector $\mathbf{y}$. The input to the jth neuron of the hidden layer o_j^{hb} has the same linear form as (6). The activation of the jth hidden neuron is the same as (30).

The backward-pass input to the ith input neuron is also the same as (8). The input activation is Gibbs or softmax

$$a_i^{xb} = \frac{e^{(o_i^{xb})}}{\sum_{l=1}^{I} e^{(o_l^{xb})}} \tag{80}$$

where a_i^{xb} is the backward-pass activation for the ith neuron of the input neuron. Then the backward error E_b is the cross entropy

$$E_b = -\sum_{i=1}^{I} x_i \ln \ a_i^{xb} \tag{81}$$

where x_i is the target value of the ith input neuron.

The partial derivatives of the hidden activation a_j^h and a_j^{hb} are the same as in (36) and (52).

The partial derivative of the output activation a_k^y for the forward classification pass is

$$\frac{\partial a_k^y}{\partial o_k^y} = \frac{\partial}{\partial o_k^y}\left(\frac{e^{(o_k^y)}}{\sum_{l=1}^{K} e^{(o_l^y)}}\right) \tag{82}$$

$$= \frac{e^{o_k^y}\left(\sum_{l=1}^{K} e^{(o_l^y)}\right) - e^{o_k^y} e^{o_k^y}}{\left(\sum_{l=1}^{K} e^{(o_l^y)}\right)^2} \tag{83}$$

$$= \frac{e^{o_k^y}\left(\sum_{l=1}^{K} e^{(o_l^y)} - e^{o_k^y}\right)}{\left(\sum_{l=1}^{K} e^{(o_l^y)}\right)^2} \tag{84}$$

$$= a_k^y\left(1 - a_k^y\right). \tag{85}$$

The partial derivative when $l \neq k$ is

$$\frac{\partial a_k^y}{\partial o_l^y} = \frac{\partial}{\partial o_l^y}\left(\frac{e^{(o_k^y)}}{\sum_{m=1}^{K} e^{(o_m^y)}}\right) \tag{86}$$

$$= \frac{-e^{o_k^y} e^{o_l^y}}{\left(\sum_{l=1}^{K} e^{(o_l^y)}\right)^2} \tag{87}$$

$$= -a_k^y \ a_l^y. \tag{88}$$

So the partial derivative of a_k^y with respect to o_l^k is

$$\frac{\partial a_k^y}{\partial o_l^y} = \begin{cases} -a_k^y \ a_l^y & \text{if } \ l \neq k \\ a_k^y\left(1 - a_k^y\right) & \text{if } \ l = k. \end{cases} \tag{89}$$

Denote this derivative as $a_k^{y\prime}$. The derivative $a_i^{xb\prime}$ of the backward classification pass has the same form because both sets of classifier neurons have softmax activations.

The partial derivative of the forward cross entropy E_f with respect to u_{jk} is

$$\frac{\partial E_f}{\partial u_{jk}} = -\frac{\partial}{\partial u_{jk}} \sum_{k=1}^{K} y_k \ln \ a_k^y \tag{90}$$

$$= \sum_{k=1}^{K}\left(\frac{\partial E_f}{\partial a_k^y} \ \frac{\partial a_k^y}{\partial o_k^y} \ \frac{\partial o_k^y}{\partial u_{jk}}\right) \tag{91}$$

$$= -\left(\frac{y_k}{a_k^y}\left(1 - a_k^y\right)a_k^y - \sum_{l\neq k}^{K} \frac{y_l}{a_l^y} a_k^y a_l^y\right) a_j^h \tag{92}$$

$$= \left(a_k^y - y_k\right)a_j^h. \tag{93}$$

The partial derivative of the forward cross entropy E_f with respect to the bias b_k^y of the kth output neuron is

$$\frac{\partial E_f}{\partial b_k^y} = \frac{\partial}{\partial b_k^y}\sum_{k=1}^{K} y_k \ln\ a_k^y \tag{94}$$

$$= \sum_{k=1}^{K}\left(\frac{\partial E_f}{\partial a_k^y}\ \frac{\partial a_k^y}{\partial o_k^y}\ \frac{\partial o_k^y}{\partial b_k^y}\right) \tag{95}$$

$$= -\left(\frac{y_k}{a_k^y}\big(1-a_k^y\big)a_k^y - \sum_{l\neq k}^{K}\frac{y_l}{a_l^y}a_k^y a_l^y\right) \tag{96}$$

$$= a_k^y - y_k. \tag{97}$$

Equations (93) and (97) show that the derivatives of E_f with respect to u_{jk} and b_k^y for double classification are the same as for double regression in (39) and (45). The activations of the hidden neurons are the same as for double regression. So the derivatives of E_f with respect to w_{ij} and b_j^h are the same as the respective ones in (42) and (48).

The partial derivative of E_b with respect to w_{ij} is

$$\frac{\partial E_b}{\partial w_{ij}} = -\frac{\partial}{\partial w_{ij}}\sum_{i=1}^{I} x_i\ \ln a_i^{xb} \tag{98}$$

$$= \sum_{i=1}^{I}\left(\frac{\partial E_b}{\partial a_i^{xb}}\ \frac{\partial a_i^{xb}}{\partial o_i^{xb}}\ \frac{\partial o_i^{xb}}{\partial w_{ij}}\right) \tag{99}$$

$$= -\left(\frac{x_i}{a_i^{xb}}\Big(1-a_i^{xb}\Big)a_i^{xb} - \sum_{l\neq i}^{I}\frac{x_l}{a_l^{xb}}a_i^{xb}a_l^{xb}\right)a_j^{hb} \tag{100}$$

$$= \Big(a_i^{xb} - x_i\Big)a_j^{hb}. \tag{101}$$

The partial derivative of E_b with respect to the bias b_i^x of the ith input neuron is

$$\frac{\partial E_b}{\partial b_i^x} = -\frac{\partial}{\partial b_i^{xb}}\sum_{i=1}^{I} x_i\ \ln a_i^{xb} \tag{102}$$

$$= \sum_{i=1}^{I}\left(\frac{\partial E_b}{\partial a_i^{xb}}\ \frac{\partial a_i^{xb}}{\partial o_i^{xb}}\ \frac{\partial o_i^{xb}}{\partial b_i^x}\right) \tag{103}$$

$$= -\left(\frac{x_i}{a_i^{xb}}\Big(1-a_i^{xb}\Big)a_i^{xb} - \sum_{l\neq i}^{I}\frac{x_l}{a_l^{xb}}a_i^{xb}a_l^{xb}\right) \tag{104}$$

$$= a_i^{xb} - x_i. \tag{105}$$

Equations (101) and (105) likewise show that the derivatives of E_b with respect to w_{ij} and b_i^x for double classification are the same as for double regression in (53) and (59). The activations of the hidden neurons are the same as for double regression. So the derivatives of E_b with respect to u_{jk} and b_j^h are the same as the respective ones in (58) and (64).

B-BP training for double classification also alternates between minimizing E_f while holding E_b constant and minimizing E_b while holding E_f constant. The forward and backward errors are again cross entropies.

The update laws for forward classification have the final form

$$u_{jk}^{(n+1)} = u_{jk}^{(n)} - \eta\Big(\big(a_k^y - y_k\big)a_j^h\Big) \tag{106}$$

$$w_{ij}^{(n+1)} = w_{ij}^{(n)} - \eta\left(\sum_{k=1}^{K}\big(a_k^y - y_k\big)u_{jk}a_j^{h'}x_i\right) \tag{107}$$

$$b_j^{h(n+1)} = b_j^{h(n)} - \eta\left(\sum_{k=1}^{K}\big(a_k^y - y_k\big)u_{jk}a_j^{h'}\right) \tag{108}$$

$$b_k^{y(n+1)} = b_k^{y(n)} - \eta\big(a_k^y - y_k\big). \tag{109}$$

The dual update laws for backward classification have the final form

$$u_{jk}^{(n+1)} = u_{jk}^{(n)} - \eta\left(\sum_{i=1}^{I}\Big(a_i^{xb} - x_i\Big)w_{ij}a_j^{hb'}y_k\right) \tag{110}$$

$$w_{ij}^{(n+1)} = w_{ij}^{(n)} - \eta\Big(\Big(a_i^{xb} - x_i\Big)a_j^{hb}\Big) \tag{111}$$

$$b_i^{x(n+1)} = b_i^{x(n)} - \eta\Big(a_i^{xb} - x_i\Big) \tag{112}$$

$$b_j^{h(n+1)} = b_j^{h(n)} - \eta\left(\sum_{i=1}^{I}\Big(a_i^{xb} - x_i\Big)w_{ij}a_j^{hb'}\right). \tag{113}$$

The derivation shows that the update rules for double classification are the same as the update rules for double regression.

B-BP training minimizes E_f while holding E_b constant. It then minimizes E_b while holding E_f constant. Equations (106)–(109) are the update rules for forward training. Equations (110)–(113) are the update rules for backward training. Each training iteration involves first running forward training and then running backward training. Algorithm 1 again summarizes the B-BP algorithm.

The more general case of double classification uses logistic neurons at the input and output layer. Then the BP derivation requires the slightly more general *logistic* cross-entropy performance measure. We used the logistic cross-entropy $E_{\log}$ for double classification training because the input and output neurons were logistic (rather than softmax)

$$E_{\log} = -\sum_{k=1}^{K} y_k \ln a_k^y - \sum_{k=1}^{K}(1-y_k)\ln\big(1-a_k^y\big). \tag{114}$$

Partially differentiating $E_{\log}$ for logistic input and output neurons gives back the same B-BP learning laws as does differentiating cross entropy for softmax input and output neurons.

C. Mixed Case: Classification and Regression

We last derive the B-BP learning algorithm for the mixed case of a neural classifier network in the forward direction and a regression network in the backward direction.

This mixed case describes the common case of neural image classification. The user needs only add backward-regression training to allow the same classifier net to predict which image input produced a given output classification. Backward regression estimates this answer as the centroid of the inverse set-theoretic mapping or preimage. The B-BP algorithm achieves this by alternating between minimizing E_f and minimizing E_b. The forward error E_f is the same as the cross entropy in the double-classification network above. The backward error E_b is the same as the squared error in double regression.

The input space is likewise the I-dimensional real space $\mathbb{R}^I$ for regression. The output space uses 1-in-K binary encoding for classification. The output neurons of regression networks use identity functions as activations. The output neurons of classifier networks use softmax activations.

The forward pass sends the input vector $\mathbf{x}$ through the hidden layer to the output layer. The input activation vector $\mathbf{a}^x$ equals $\mathbf{x}$. We again consider only a single hidden layer for simplicity. The input o_j^h to the jth hidden neuron is the same as in (2). The activation a_j^h of the jth hidden layer is the ordinary logistic activation in (27). Equation (4) defines the input o_k^y to the kth output neuron. The output activation is softmax. So the output activation a_k^y is the same as in (78). The forward error E_f is the cross entropy in (79). The forward pass in this mixed case is the same as the forward pass for double classification. So (42), (48), (93), and (97) give the respective derivatives of the forward error E_f with respect to w_{ij}, b_j^h, u_{jk}, and b_y^k.

The backward pass propagates the 1-in-K vector $\mathbf{y}$ from the output through the hidden layer to the input layer. The output layer activation vector $\mathbf{a}^{\mathbf{y}}$ equals $\mathbf{y}$. The input o_j^{hb} to the jth hidden neuron for the backward pass is the same as in (6). Equation (30) gives the activation a_j^{hb} for the jth hidden unit in the backward pass. Equation (8) gives the input o_i^{xb} for the ith input neuron. The activation a_i^{xb} of the ith input neuron for the backward pass is the same as in (31). The backward error E_b is the squared error in (32).

The backward pass in this mixed case is the same as the backward pass for double regression. So (55), (58), (61), and (64) give the respective derivatives of the backward error E_b with respect to w_{ij}, b_i^x, u_{jk}, and b_j^h.

The update laws for forward classification–regression training have the final form

$$u_{jk}^{(n+1)} = u_{jk}^{(n)} - \eta\left(a_k^y - y_k\right)a_j^h \tag{115}$$

$$w_{ij}^{(n+1)} = w_{ij}^{(n)} - \eta\left(\sum_{k=1}^{K}\left(a_k^y - y_k\right)u_{jk}a_j^{h'}x_i\right) \tag{116}$$

$$b_j^{h(n+1)} = b_j^{h(n)} - \eta\left(\sum_{k=1}^{K}\left(a_k^y - y_k\right)u_{jk}a_j^{h'}\right) \tag{117}$$

$$b_k^{y(n+1)} = b_k^{y(n)} - \eta\left(a_k^y - y_k\right). \tag{118}$$

The update laws for backward classification–regression training have the final form

$$u_{jk}^{(n+1)} = u_{jk}^{(n)} - \eta\left(\sum_{i=1}^{I}\left(a_i^{xb} - x_i\right)w_{ij}a_j^{hb'}y_k\right) \tag{119}$$

$$w_{ij}^{(n+1)} = w_{ij}^{(n)} - \eta\left(a_i^{xb} - x_i\right)a_j^{hb} \tag{120}$$

$$b_i^{x(n+1)} = b_i^{x(n)} - \eta\left(a_i^{xb} - x_i\right) \tag{121}$$

TABLE II
5-BIT BIPOLAR PERMUTATION FUNCTION

Input x	Output t	Input x	Output t
[− − − − −]	[+ + − + +]	[+ − − − −]	[− + + + +]
[− − − − +]	[− − + − −]	[+ − − − +]	[− + − − −]
[− − − + −]	[− − − + −]	[+ − − + −]	[+ − − + −]
[− − − + +]	[+ + + − +]	[+ − − + +]	[− − + − +]
[− − + − −]	[+ + − + −]	[+ − + − −]	[− + − + +]
[− − + − +]	[+ − − + +]	[+ − + − +]	[+ + − − +]
[− − + + −]	[− + + − +]	[+ − + + −]	[+ + + + +]
[− − + + +]	[− − + + +]	[+ − + + +]	[− − + + −]
[− + − − −]	[+ − + + +]	[+ + − − −]	[+ + + − −]
[− + − − +]	[+ − − − +]	[+ + − − +]	[− + − + −]
[− + − + −]	[+ − + + −]	[+ + − + −]	[+ − − − −]
[− + − + +]	[− + + − −]	[+ + − + +]	[− − − + +]
[− + + − −]	[− + + + −]	[+ + + − −]	[− − − − −]
[− + + − +]	[+ + − − −]	[+ + + − +]	[− + − − +]
[− + + + −]	[+ − + − +]	[+ + + + −]	[+ + + + −]
[− + + + +]	[− − − − +]	[+ + + + +]	[+ − + − −]

TABLE III
FORWARD-PASS CROSS ENTROPY E_f

Hidden Neurons	Backpropagation Training: Forward	Backward	Bidirectional
5	0.4222	1.4534	0.4729
10	0.0881	1.8173	0.3045
20	0.0132	4.7554	0.0539
50	0.0037	4.4039	0.0034
100	0.0014	5.8473	0.0029

$$b_j^{h(n+1)} = b_j^{h(n)} - \eta\left(\sum_{i=1}^{I}\left(a_i^{xb} - x_i\right)w_{ij}a_j^{hb'}\right). \tag{122}$$

B-BP training minimizes E_f while holding E_b constant. It then minimizes the E_b while holding E_f constant. Equations (115)–(118) state the update rules for forward training. Equations (119)–(122) state the update rules for backward training. Algorithm 1 shows how forward learning combines with backward learning in B-BP.

IV. SIMULATION RESULTS

We tested the B-BP algorithm for double classification on a 5-bit permutation function. We used 3-layer networks with different numbers of hidden neurons. The neurons used bipolar logistic activations. The performance measure was the logistic cross entropy in (114). The B-BP algorithm produced either an exact representation or an approximation. The permutation function bijectively mapped the 5-bit bipolar vector space $\{-1, 1\}^5$ of 32 bipolar vectors onto itself. Table II displays the permutation test function. We compared the forward and backward forms of unidirectional BP with B-BP. We also tested whether adding more hidden neurons improved network approximation accuracy.

The forward pass of standard BP used logistic cross entropy as its error function. The backward pass did as well. B-BP summed the forward and backward errors for its joint error. We computed the test error for the forward and backward passes. Each plotted error value averaged 20 runs.

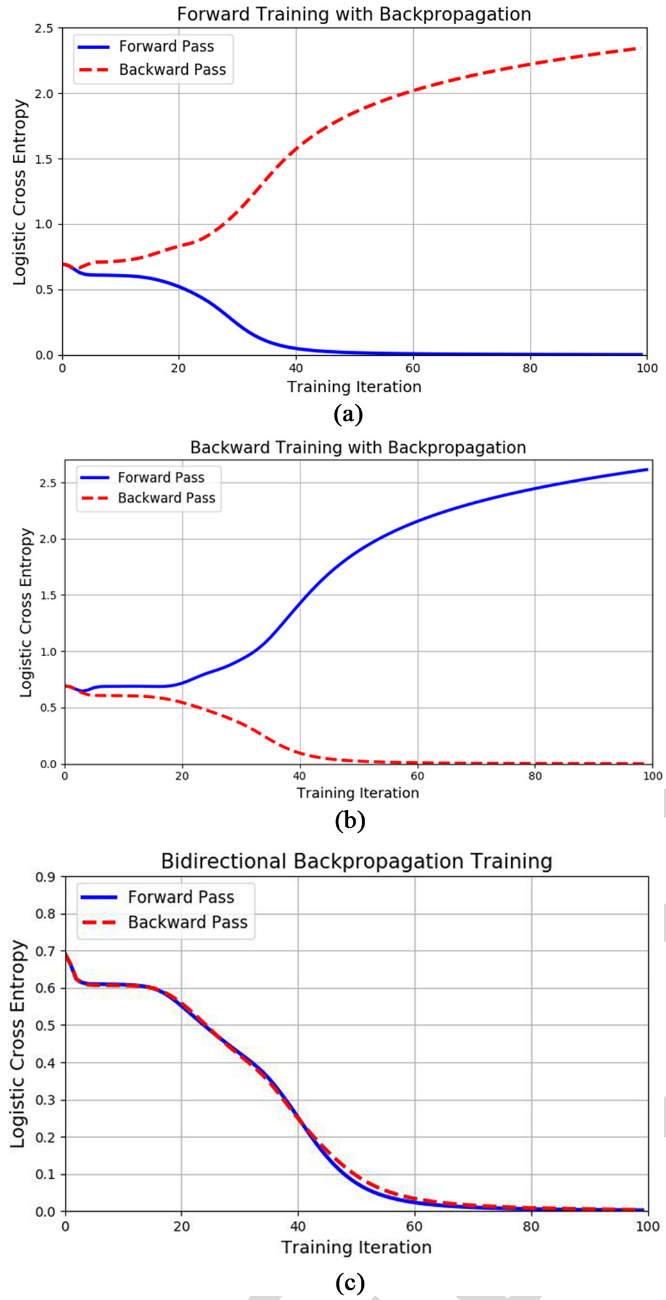

Fig. 4. Logistic-cross-entropy learning for double classification using 100 hidden neurons with forward BP training, backward BP training, and B-BP training. The trained network represents the 5-bit permutation function in Table II. (a) Forward BP tuned the network with respect to logistic cross entropy for the forward pass using E_f only. (b) Backward BP training tuned the network with respect to logistic cross entropy for the backward pass using E_b only. (c) B-BP training summed the logistic cross entropies for both the forward-pass error term E_f and the backward-pass error term E_b to update the network parameters.

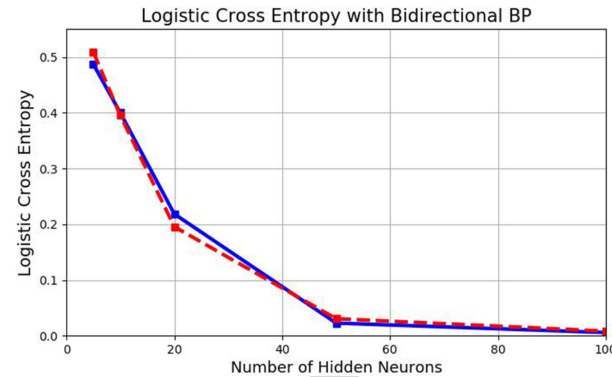

Fig. 5. B-BP training error for the 5-bit permutation in Table II using different numbers of hidden neurons. Training used the double-classification B-BP algorithm. The two curves describe the logistic cross entropy for the forward and backward passes through the 3-layer network. Each test used 640 samples. The number of hidden neurons increased from 5, 10, 20, 50, to 100.

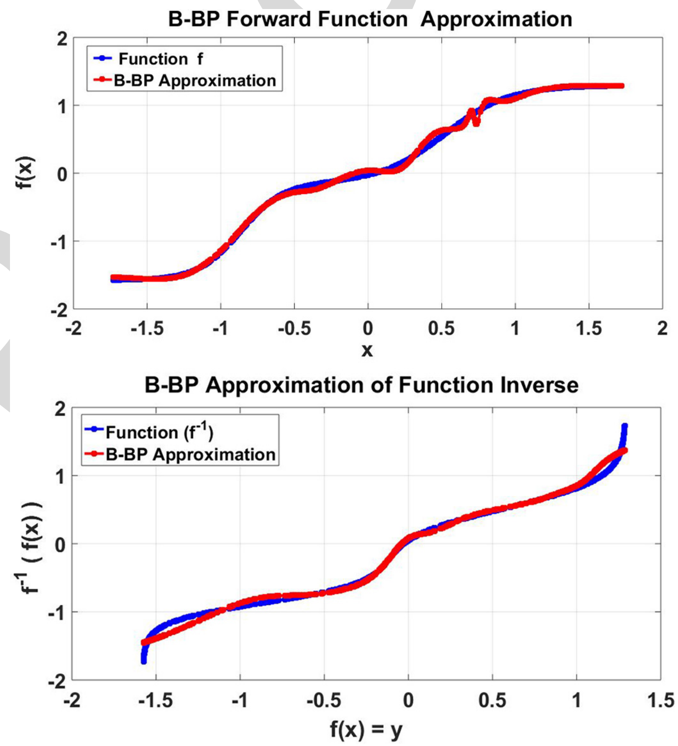

Fig. 6. B-BP double-regression approximation of the invertible function $f(x) = 0.5\sigma(6x+3) + 0.5\sigma(4x-1.2)$ using a deep 8-layer network with six hidden layers. The function σ denotes the bipolar logistic function in (1). Each hidden layer contained ten bipolar logistic neurons. The input and output layers each used a single neuron with an identity activation function. The forward pass approximated the forward function f. The backward pass approximated the inverse function f^{-1}.

Fig. 4 shows the results of running the three types of BP learning for classification on a 3-layer network with 100 hidden neurons. The values of E_f and E_b decrease with an increase in the training iterations for B-BP. This was not the case for the unidirectional cases of forward BP and backward BP training. Forward and backward training performed well only for function approximation in their respective training direction. Neither performed well in the opposite direction.

Table III shows the forward-pass cross entropy E_f for learning 3-layer classification neural networks as the number of hidden neurons grows. We again compared the three forms of BP for the network training: two forms of unidirectional BP and B-BP. The forward-pass error for forward BP fell substantially as the number of hidden neurons grew. The forward-pass error of backward BP decreased slightly as the number of hidden neurons grew. It gave the worst performance. B-BP performed well on the test set. Its forward-pass error also

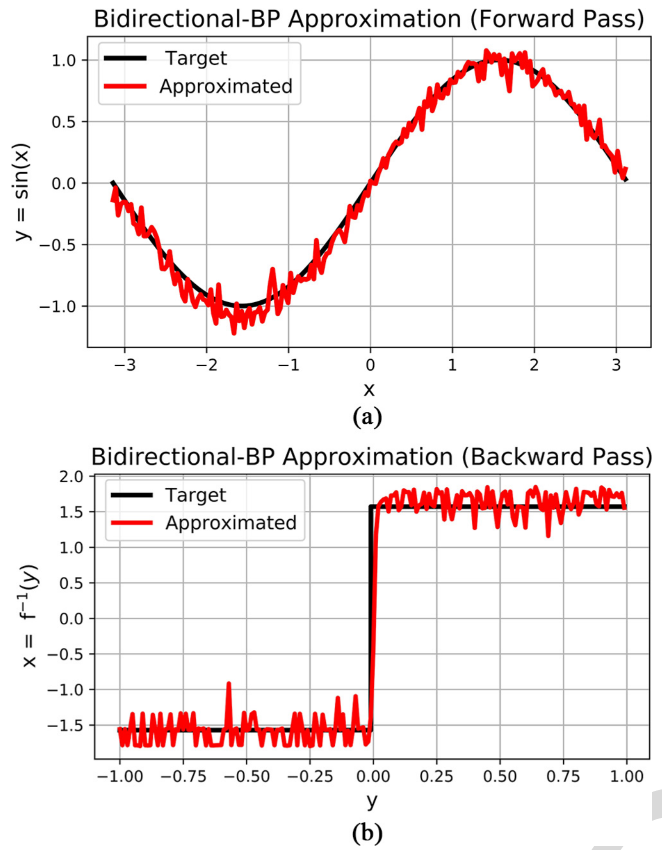

Fig. 7. B-BP double-regression learning of the noninvertible target function $f(x) = \sin x$. (a) Forward pass learned the function $y = f(x) = \sin x$. (b) Backward pass approximated the centroid of the values in the set-theoretic preimage $f^{-1}(\{y\})$ for y values in $(-1, 1)$. The two centroids were $-(\pi/2)$ and $(\pi/2)$.

TABLE IV
BACKWARD-PASS CROSS ENTROPY E_b

Hidden Neurons	Backpropagation Training		
	Forward	Backward	Bidirectional
5	2.9370	0.3572	0.4692
10	2.4920	0.1053	0.3198
20	4.6432	0.0149	0.0542
50	7.0921	0.0027	0.0040
100	7.1414	0.0013	0.0032

fell substantially as the number of hidden neurons grew. Table IV shows similar error-versus-hidden-neuron results for the backward-pass cross entropy E_b.

The two tables jointly show that the unidirectional forms of BP for regression performed well only in one direction. The B-BP algorithm performed well in both directions.

We tested the B-BP algorithm for double regression with the invertible function $f(x) = 0.5\sigma(6x + 3) + 0.5\sigma(4x - 1.2)$ for values of $x \in [-1.5, 1.5]$. We used a deep 8-layer network with 6 hidden layers for this approximation. Each hidden layer had 10 bipolar logistic neurons. There was only a single identity neuron in the input and output layers. The error functions E_f and E_b were ordinary squared error. Fig. 6 compares the B-BP approximation with the target function for both the forward pass and the backward pass.

Algorithm 1 B-BP Algorithm

Data: T input vectors $\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(T)}\}$ and T corresponding output vectors $\{\mathbf{y}^{(1)}, \mathbf{y}^{(2)}, \ldots, \mathbf{y}^{(T)}\}$ such that $f(\mathbf{x}^{(l)}) = \mathbf{y}^{(l)}$. Number of hidden neurons J. Batch size S and number of epochs R. Choose the learning rate η.

Result: Bidirectional neural network representation for function f.

Initialize: Randomly select the initial weights $W^{(0)}$ and $U^{(0)}$. Randomly pick the bias weights for input, hidden, and output neurons $\{b^{x(0)}, b^{h(0)}, b^{y(0)}\}$.

<u>while</u> epoch $r: 0 \longrightarrow R$ do

Select S random samples from the training dataset.

Initialize: $\Delta W = 0, \Delta U = 0, \Delta b^x = 0, \Delta b^h = 0, \Delta b^y = 0.$

FORWARD TRAINING

<u>while</u> *batch_size* $l: 1 \longrightarrow S$

- Randomly pick input vector $\mathbf{x}^{(l)}$ and its corresponding output vector $\mathbf{y}^{(l)}$
- Compute hidden layer input $\mathbf{o}^h$ and the corresponding hidden activation $\mathbf{a}^h$
- Compute output layer input $\mathbf{o}^y$ and the corresponding output activation $\mathbf{a}^y$
- Compute the forward error E_f
- Compute the following derivatives: $\nabla_W E_f, \nabla_U E_f, \nabla_{b^h} E_f$, and $\nabla_{b^y} E_f$
- Update: $\Delta W = \Delta W + \nabla_W E_f$; $\Delta b^h = \Delta b^h + \nabla_{b^h} E_f$
  $\Delta U = \Delta U + \nabla_U E_f$; $\Delta b^y = \Delta b^y + \nabla_{b^y} E_f$

End

BACKWARD TRAINING

<u>while</u> *batch_size* $l: 1 \longrightarrow S$

- Pick input vector $\mathbf{x}^{(l)}$ and its corresponding output vector $\mathbf{y}^{(l)}$.
- Compute hidden layer input $\mathbf{o}^{hb}$ and hidden activation $\mathbf{a}^{hb}$.
- Compute input $\mathbf{o}^{xb}$ at the input layer and input activation $\mathbf{a}^{xb}$.
- Compute the backward error E_b
- Compute the following derivatives: $\nabla_W E_b, \nabla_U E_b, \nabla_{b^h} E_b$, and $\nabla_{b^x} E_b$
- Update: $\Delta W = \Delta W + \nabla_W E_b$; $\Delta b^h = \Delta b^h + \nabla_{b^h} E_b$
  $\Delta U = \Delta U + \nabla_U E_b$; $\Delta b^x = \Delta b^x + \nabla_{b^x} E_b$

End

Update:

- $W^{(r+1)} = W^{(r)} - \eta \Delta W$
- $U^{(r+1)} = U^{(r)} - \eta \Delta U$
- $b^{x(r+1)} = b^{x(r)} - \eta \Delta b^x$
- $b^{h(r+1)} = b^{h(r)} - \eta \Delta b^h$
- $b^{y(r+1)} = b^{y(r)} - \eta \Delta b^y$

End

We also tested the B-BP double-regression algorithm on the *noninvertible* function $f(x) = \sin x$ for $x \in [-\pi, \pi]$. The forward mapping $f(x) = \sin x$ is a well-defined point function. The backward mapping $y = \sin^{-1}(f(x))$ is not. It defines instead a set-based pullback or preimage $f^{-1}(y) = f^{-1}(\{y\}) = \{x \in \mathbb{R} : f(x) = y\} \subset \mathbb{R}$. The B-BP-trained neural network tends to map each output point y to the centroid of its preimage $f^{-1}(y)$ on the backward pass because centroids minimize squared error and because backward-regression training uses squared error as its performance measure. Fig. 7 shows that forward regression learns the target function $\sin x$ while backward regression approximates the centroids $-(\pi/2)$ and $(\pi/2)$ of the two preimage sets.

V. CONCLUSION

Unidirectional BP learning extends to B-BP learning if the algorithm uses the appropriate joint error function for both forward and backward passes. This bidirectional extension applies to classification networks as well as to regression networks and to their combinations. Most classification networks can easily acquire a backward-inference capability if they include a backward-regression step in their training. So most networks simply ignore this inverse property of their weight structure.

Theorem 1 shows that a bidirectional multilayer threshold network can exactly represent a permutation mapping if the hidden layer contains an exponential number of hidden threshold neurons. An open question is whether these bidirectional networks can represent an arbitrary invertible mapping with far fewer hidden neurons. A simpler question holds for the weaker case of uniform approximation of invertible mappings.

Another open question deals with noise: to what extent does carefully injected noise speed B-BP convergence and accuracy? There are two bases for this question. The first is that the likelihood structure of BP implies that BP is itself a special case of the expectation–maximization algorithm [19]. The second basis is that appropriate noise can boost the EM family of hill-climbing algorithms on average because such noise makes signals more probable on average [21], [23].

References

[1] 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.

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

[3] C. M. Bishop, *Pattern Recognition and Machine Learning*. New York, NY, USA: Springer, 2006.

[4] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, May 2015.

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

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

[7] B. Kosko, *Neural Networks and Fuzzy Systems: A Dynamical Systems Approach to Machine Intelligence*. Englewood Cliffs, NJ, USA: Prentice-Hall, 1991.

[8] S. Y. Kung, *Kernel Methods and Machine Learning*. Cambridge, U.K.: Cambridge Univ. Press, 2014.

[9] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Math. Control Signals Syst.*, vol. 2, no. 4, pp. 303–314, 1989.

[10] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Netw.*, vol. 2, no. 5, pp. 359–366, 1989.

[11] B. Kosko, "Fuzzy systems as universal approximators," *IEEE Trans. Comput.*, vol. 43, no. 11, pp. 1329–1333, Nov. 1994.

[12] F. Watkins, "The representation problem for additive fuzzy systems," in *Proc. Int. Conf. Fuzzy Syst. (IEEE FUZZ)*, 1995, pp. 117–122.

[13] B. Kosko, "Generalized mixture representations and combinations for additive fuzzy systems," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, 2017, pp. 3761–3768.

[14] B. Kosko, "Additive fuzzy systems: From generalized mixtures to rule continua," *Int. J. Intell. Syst.*, vol. 33, no. 8, pp. 1573–1623, 2017.

[15] J.-N. Hwang, J. J. Choi, S. Oh, and R. J. Marks, "Query-based learning applied to partially trained multilayer perceptrons," *IEEE Trans. Neural Netw.*, vol. 2, no. 1, pp. 131–136, Jan. 1991.

[16] E. W. Saad, J. J. Choi, J. L. Vian, and D. C. Wunsch, "Query-based learning for aerospace applications," *IEEE Trans. Neural Netw.*, vol. 14, no. 6, pp. 1437–1448, Nov. 2003.

[17] E. W. Saad and D. C. Wunsch, "Neural network explanation using inversion," *Neural Netw.*, vol. 20, no. 1, pp. 78–93, 2007.

[18] Y. Yang, Y. Wang, and X. Yuan, "Bidirectional extreme learning machine for regression problem and its learning effectiveness," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 23, no. 9, pp. 1498–1505, Sep. 2012.

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

[20] S. Kullback and R. A. Leibler, "On information and sufficiency," *Ann. Math. Stat.*, vol. 22, no. 1, pp. 79–86, 1951.

[21] O. Osoba and B. Kosko, "The noisy expectation-maximization algorithm for multiplicative noise injection," *Fluctuation Noise Lett.*, vol. 15, no. 4, 2016, Art. no. 1650007.

[22] R. V. Hogg, J. McKean, and A. T. Craig, *Introduction to Mathematical Statistics*. Boston, MA, USA: Pearson, 2013.

[23] O. Osoba, S. Mitaim, and B. Kosko, "The noisy expectation–maximization algorithm," *Fluctuation Noise Lett.*, vol. 12, no. 3, 2013, Art. no. 1350012.

Olaoluwa (Oliver) Adigun received the Bachelor of Science degree in electronic and electrical engineering from Obafemi Awolowo University, Ife, Nigeria. He is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, Signal and Image Processing Institute, University of Southern California, Los Angeles, CA, USA.

He has been an Intern with Google AI, Mountain View, CA, USA, and with the Machine Learning Group, Amazon, Seattle, WA, USA.

Mr. Adigun shared the Best Paper Award for his research on noise-boosted recurrent backpropagation at the 2017 International Joint Conference on Neural Networks.

Bart Kosko (M'85–SM'07–F'10) received the degrees in philosophy, economics, applied mathematics, electrical engineering, and law.

He is a Professor of Department of Electrical and Computer Engineering and Law and the Past Director of Signal and Image Processing Institute with the University of Southern California, Los Angeles, CA, USA, and a Licensed Attorney. He has published the textbooks entitled *Neural Networks and Fuzzy Systems* and *Fuzzy Engineering*, the trade books entitled *Fuzzy Thinking*, *Heaven in a Chip*, and *Noise*, the edited volume *Neural Networks and Signal Processing*, the co-edited volume *Intelligent Signal Processing*, the novel *Nanotime*, and the upcoming novel *Cool Earth*.

Dr. Kosko was a co-recipient of the Best Paper Award at the 2017 International Joint Conference on Neural Networks.

Bidirectional Backpropagation

Olaoluwa Adigun, *Member, IEEE*, and Bart Kosko, *Fellow, IEEE*

***Abstract*—We extend backpropagation (BP) learning from ordinary unidirectional training to bidirectional training of deep multilayer neural networks. This gives a form of backward chaining or inverse inference from an observed network output to a candidate input that produced the output. The trained network learns a bidirectional mapping and can apply to some inverse problems. A bidirectional multilayer neural network can exactly represent some invertible functions. We prove that a fixed three-layer network can always exactly represent any finite permutation function and its inverse. The forward pass computes the permutation function value. The backward pass computes the inverse permutation with the same weights and hidden neurons. A joint forward–backward error function allows BP learning in both directions without overwriting learning in either direction. The learning applies to classification and regression. The algorithms do not require that the underlying sampled function has an inverse. A trained regression network tends to map an output back to the centroid of its preimage set.**

***Index Terms*—Backpropagation (BP) learning, backward chaining, bidirectional associative memory, function approximation, function representation, inverse problems.**

I. BIDIRECTIONAL BACKPROPAGATION

WE EXTEND the familiar unidirectional backpropagation (BP) algorithm [1]–[5] to the bidirectional case. Unidirectional BP maps an input vector to an output vector by passing the input vector forward through the network's visible and hidden neurons and its connection weights. Bidirectional BP (B-BP) combines this forward pass with a backward pass through the *same* neurons and weights. It does not use two separate feedforward or unidirectional networks.

B-BP training endows a multilayered neural network $N : \mathbb{R}^n \to \mathbb{R}^p$ with a form of backward inference. The forward pass gives the usual predicted neural output $N(\mathbf{x})$ given a vector input $\mathbf{x}$. The output vector value $\mathbf{y} = N(\mathbf{x})$ answers the *what-if* question that $\mathbf{x}$ poses: What would we observe if $\mathbf{x}$ occurred? What would be the effect? The backward pass answers the *why* question that $\mathbf{y}$ poses: Why did $\mathbf{y}$ occur? What type of input would cause $\mathbf{y}$? Feedback convergence to a resonating bidirectional fixed-point attractor [6], [7] gives a long-term or equilibrium answer to both the what-if and why questions. This paper does not address the global stability of multilayered bidirectional networks.

Manuscript received May 18, 2017; revised September 2, 2017; accepted November 10, 2017. This paper was recommended by Associate Editor G.-B. Huang. *(Corresponding author: Bart Kosko.)*

The authors are 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 of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSMC.2019.2916096

Bidirectional neural learning applies to large-scale problems and big data because the BP algorithm scales linearly with training data. BP has time complexity $O(n)$ for n training samples because both the forward and backward passes have complexity $O(n)$. So the B-BP algorithm still has $O(n)$ complexity because $O(n) + O(n) = O(n)$. This linear scaling does not hold for most machine-learning algorithms. An example is the quadratic complexity $O(n^2)$ of support-vector kernel methods [8].

We first show that multilayer bidirectional networks have sufficient power to exactly represent permutation mappings. These mappings are invertible and discrete. We then develop the B-BP algorithms that can approximate these and other mappings if the networks have enough hidden neurons.

A neural network N exactly *represents* a function f just in case $N(\mathbf{x}) = f(\mathbf{x})$ for all input vectors $\mathbf{x}$. Exact representation is much stronger than the more familiar property of function approximation: $N(\mathbf{x}) \approx f(\mathbf{x})$. Feedforward multilayer neural networks can uniformly approximate continuous functions on compact sets [9], [10]. Additive fuzzy systems are also uniform function approximators [11]. But additive fuzzy systems have the further property that they can exactly represent any real function if it is bounded [12]. This exact representation needs only two fuzzy rules because the rules absorb the function into their fuzzy sets. This holds more generally for generalized probability mixtures because the fuzzy rules define the mixed probability densities [13], [14].

Figs. 1 and 2 show bidirectional 3-layer networks of zero-threshold neurons. Both networks exactly represent the 3-bit permutation function f in Table I where $\{-,-,+\}$ denotes $\{-1,-1,1\}$. So f is a self-bijection that rearranges the 8 vectors in the bipolar hypercube $\{-1,1\}^3$. This f is just one of the 8! or 40 320 permutation maps or rearrangements on the bipolar hypercube $\{-1,1\}^3$. The forward pass converts the input bipolar vector $(1,1,1)$ to the output bipolar vector $(-1,-1,1)$. The backward pass converts $(-1,-1,1)$ to $(1,1,1)$ over the *same* fixed synaptic connection weights. These same weights and neurons similarly convert the other 7 input vectors in the first column of Table I to the corresponding 7 output vectors in the second column and vice versa.

Theorem 1 states that a multilayer bidirectional network can exactly represent any finite bipolar or binary permutation function. This result requires a hidden layer with 2^n hidden neurons for an n-bit permutation function on the bipolar hypercube $\{-1,1\}^n$. Fig. 3 shows such a network. Using so many hidden neurons is not practical or necessary in most real-world cases. The exact bidirectional representation in Fig. 1 uses only 4 hidden threshold neurons to represent the 3-bit permutation function. This was the smallest hidden layer that we found

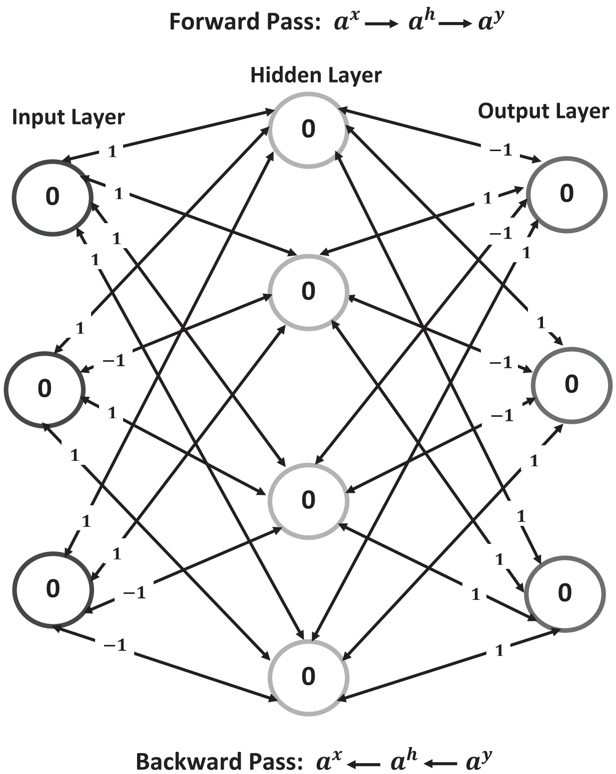

Fig. 1. Exact bidirectional representation of a permutation map. The 3-layer bidirectional threshold network exactly represents the invertible 3-bit bipolar permutation function f in Table I. The network uses four hidden neurons. The forward pass takes the input bipolar vector $\mathbf{x}$ at the input layer and feeds it forward through the weighted edges and the hidden layer of threshold neurons to the output layer. The backward pass feeds the output bipolar vector $\mathbf{y}$ back through the same weights and neurons. All neurons are bipolar and use zero thresholds. The bidirectional network computes $\mathbf{y} = f(\mathbf{x})$ on the forward pass. It computes the inverse value $f^{-1}(\mathbf{y})$ on the backward pass.

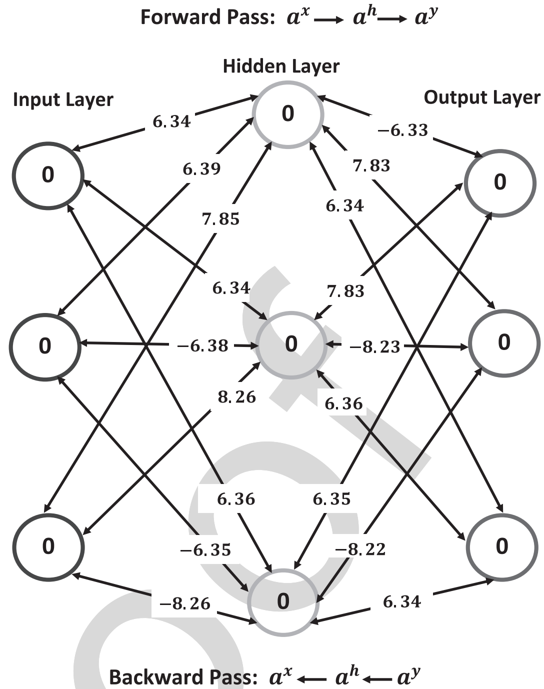

Fig. 2. Learned bidirectional representation of the 3-bit permutation in Table I. The bidirectional BP algorithm found this representation using the double-classification learning laws of Section III. It used only three hidden neurons. All the neurons were bipolar and had zero thresholds. Zero thresholding gave an exact representation of the 3-bit permutation.

through guesswork. Many other bidirectional representations also use fewer than 8 hidden neurons.

We seek instead a practical learning algorithm that can learn bidirectional approximations from sample data. Fig. 2 shows a learned bidirectional representation of the same 3-bit permutation in Table I. It uses only 3 hidden neurons. The B-BP algorithm tuned the neurons' threshold values as well as their connection weights. All the learned threshold values were near zero. We rounded them to zero to achieve the bidirectional representation with just 3 hidden neurons.

The rest of this paper derives the B-BP algorithm for regression and classification in both directions and for mixed classification–regression. This takes some care because training the weights in one direction tends to overwrite their BP training in the other direction. The B-BP algorithm solves this problem by minimizing a *joint* error function. The lone error function is cross entropy for unidirectional classification. It is squared error for unidirectional regression. Fig. 4 compares ordinary BP training and overwriting with B-BP training.

The learned approximation tends to improve if we add more hidden neurons. Fig. 5 shows that the B-BP training cross-entropy error falls as the number of hidden neurons grows when learning the 5-bit permutation in Table II.

Fig. 6 shows a deep 8-layer bidirectional approximation of the nonlinear function $f(x) = 0.5\sigma(6x + 3) + 0.5\sigma(4x - 1.2)$ and its inverse. The network used 6 hidden layers with 10 bipolar logistic neurons per layer. A bipolar logistic activation σ scales and translates an ordinary unit-interval-valued logistic

$$\sigma(x) = \frac{2}{1 + e^{-x}} - 1. \tag{1}$$

The final sections show that similar B-BP algorithms hold for training double-classification networks and mixed classification–regression networks. The B-BP learning laws are the same for regression and classification subject to these conditions: regression minimizes the squared error and uses identity output neurons. Classification minimizes the cross entropy and uses softmax output neurons. Both cases maximize the network likelihood or log-likelihood function. Logistic input and output neurons give the same B-BP learning laws if the network minimizes the bipolar cross entropy in (114). We call this *backpropagation invariance*.

B-BP learning also approximates noninvertible functions. The algorithm tends to learn the centroid of many-to-one functions. Suppose that the target function $f : \mathbb{R}^n \rightarrow \mathbb{R}^p$ is not one-to-one or injective. So it has no inverse f^{-1} point mapping. But it does have a *set-valued* inverse or preimage pullback mapping $f^{-1} : 2^{\mathbb{R}^p} \rightarrow 2^{\mathbb{R}^n}$ such that $f^{-1}(B) = \{x \in \mathbb{R}^n : f(x) \in B\}$ for any $B \subset \mathbb{R}^p$. Suppose that the n input training samples $x_1, \ldots, x_n$ map to the same output training sample $y : f^{-1}(\{y\}) = \{x_1, \ldots, x_n\}$. Then B-BP learning tends to map y to the centroid $\bar{x}$ of $f^{-1}(\{y\})$ because the centroid minimizes the mean-squared error of regression.

TABLE I
3-BIT BIPOLAR PERMUTATION FUNCTION f

Input x	Output t
$[+ + +]$	$[- - +]$
$[+ + -]$	$[- + +]$
$[+ - +]$	$[+ + +]$
$[+ - -]$	$[+ - +]$
$[- + +]$	$[- + -]$
$[- + -]$	$[- - -]$
$[- - +]$	$[+ - -]$
$[- - -]$	$[+ + -]$

Fig. 7 shows such an approximation for the noninvertible target function $f(x) = \sin x$. The forward regression approximates $\sin x$. The backward regression approximates the average or centroid of the two points in the preimage set of $y = \sin x$. Then $f^{-1}(\{y\}) = \sin^{-1}(y) = \{\theta, \pi - \theta\}$ for $0 < \theta < (\pi/2)$ if $0 < y < 1$. This gives the pullback's centroid as $(\pi/2)$. The centroid equals $-(\pi/2)$ if $-1 < y < 0$.

B-BP differs from earlier neural approaches to approximating inverses. Hwang *et al.* [15] developed an inverse algorithm for query-based learning in binary classification. Their BP-based algorithm is not bidirectional. It instead exploits the data-weight inner-product input to neurons. It holds the weights constant while it tunes the data for a given output. Saad *et al.* [16], [17] have applied this inverse algorithm to problems in aerospace and elsewhere. B-BP also differs from the more recent bidirectional extreme-learning-machine algorithm that uses a two-stage learning process but in a unidirectional network [18].

II. BIDIRECTIONAL EXACT REPRESENTATION OF BIPOLAR PERMUTATIONS

This section proves that there exist multilayered neural networks that can exactly bidirectionally represent some invertible functions. We first define the network variables. The proof uses threshold neurons. The B-BP algorithms below use soft-threshold logistic sigmoids for hidden neurons.

A bidirectional neural network is a multilayer network $N : X \to Y$ that maps the input space X to the output space Y *and conversely* through the same set of weights. The backward pass uses the matrix transposes of the weight matrices that the forward pass uses. Such a network is a bidirectional associative memory or BAM [6], [7]. The original BAM theorem [6] states that any *two*-layer neural network is globally bidirectionally stable for any sole rectangular weight matrix $\mathbf{W}$ with real entries.

The forward pass sends the input vector $\mathbf{x}$ through the weight matrix $\mathbf{W}$ that connects the input layer to the hidden layer. The result passes on through matrix $\mathbf{U}$ to the output layer. The backward pass sends the output $\mathbf{y}$ from the output layer back through the hidden layer to the input layer. Let $I, J,$ and K denote the respective numbers of input, hidden, and output neurons. Then the $I \times J$ matrix $\mathbf{W}$ connects the input layer to the hidden. The $J \times K$ matrix $\mathbf{U}$ connects the hidden layer to the output layer.

The hidden-neuron input o_j^h has the affine form

$$o_j^h = \sum_{i=1}^{I} w_{ij} a_i^x(x_i) + b_j^h \tag{2}$$

where weight w_{ij} connects the ith input neuron to the jth hidden neuron, a_i^x is the activation of the ith input neuron, and b_j^h is the bias of the jth hidden neuron. The activation a_j^h of the jth hidden neuron is a bipolar threshold

$$a_j^h\left(o_j^h\right) = \begin{cases} -1 & \text{if } o_j^h \le 0 \\ 1 & \text{if } o_j^h > 0. \end{cases} \tag{3}$$

The B-BP algorithm in the next section uses soft-threshold bipolar logistic functions for the hidden activations because such sigmoid functions are differentiable. The proof below also modifies the hidden thresholds to take on binary values in (14) and to fire with a slightly different condition.

The input o_k^y to the kth output neuron from the hidden layer is also affine

$$o_k^y = \sum_{j=1}^{J} u_{jk} a_j^h + b_k^y \tag{4}$$

where weight u_{jk} connects the jth hidden neuron to the kth output neuron. Term b_k^y is the additive bias of the kth output neuron. The output activation vector $\mathbf{a}^y$ gives the predicted outcome or target on the forward pass. The kth output neuron has bipolar threshold activation a_k^y

$$a_k^y\left(o_k^y\right) = \begin{cases} -1 & \text{if } o_k^y \le 0 \\ 1 & \text{if } o_k^y > 0. \end{cases} \tag{5}$$

The forward pass of an input bipolar vector $\mathbf{x}$ from Table I through the network in Fig. 1 gives an output activation vector $\mathbf{a}^y$ that equals the table's corresponding target vector $\mathbf{y}$. The backward pass feeds $\mathbf{y}$ from the output layer back through the hidden layer to the input layer. Then the backward-pass input o_j^{hb} to the jth hidden neuron is

$$o_j^{hb} = \sum_{k=1}^{K} u_{jk} a_k^y(y_k) + b_j^h \tag{6}$$

where y_k is the output of the kth output neuron. The term a_k^y is the activation of the kth output neuron. The backward-pass activation of the jth hidden neuron a_j^{hb} is

$$a_j^{hb}\left(o_j^{hb}\right) = \begin{cases} -1 & \text{if } o_j^{hb} \le 0 \\ 1 & \text{if } o_j^{hb} > 0. \end{cases} \tag{7}$$

The backward-pass input o_i^{xb} to the ith input neuron is

$$o_i^{xb} = \sum_{j=1}^{J} w_{ij} a_j^{hb} + b_i^x \tag{8}$$

where b_i^x is the bias for the ith input neuron. The input-layer activation $\mathbf{a}^x$ gives the predicted value for the backward pass. The ith input neuron has bipolar activation

$$a_i^{xb}\left(o_i^{xb}\right) = \begin{cases} -1 & \text{if } o_i^{xb} \le 0 \\ 1 & \text{if } o_i^{xb} > 0. \end{cases} \tag{9}$$

We can now state and prove the bidirectional representation theorem for bipolar permutations. The theorem also applies to binary permutations because the input and output neurons have bipolar threshold activations.

Theorem 1 (Exact Bidirectional Representation of Bipolar Permutation Functions): Suppose that the invertible function $f : \{-1, 1\}^n \to \{-1, 1\}^n$ is a permutation. Then there exists a 3-layer bidirectional neural network $N : \{-1, 1\}^n \to \{-1, 1\}^n$ that exactly represents f in the sense that $N(\mathbf{x}) = f(\mathbf{x})$ and that $N^{-1}(\mathbf{x}) = f^{-1}(\mathbf{x})$ for all $\mathbf{x}$. The hidden layer has 2^n threshold neurons.

Proof: The proof constructs weight matrices $\mathbf{W}$ and $\mathbf{U}$ so that exactly one hidden neuron fires on both the forward and the backward passes. Fig. 3 shows the proof technique for the special case of a 3-bit bipolar permutation. We structure the network so that an input vector $\mathbf{x}$ fires only one hidden neuron on the forward pass. The output vector $\mathbf{y} = \mathbf{N}(\mathbf{x})$ fires only the same hidden neuron on the backward pass.

The bipolar permutation f is a bijective map of the bipolar hypercube $\{-1, 1\}^n$ onto itself. The bipolar hypercube contains the 2^n input bipolar column vectors $\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_{2^n}}$. It likewise contains the 2^n output bipolar vectors $\mathbf{y_1}, \mathbf{y_2}, \ldots, \mathbf{y_{2^n}}$. The network uses 2^n corresponding hidden threshold neurons. So $J = 2^n$.

Matrix $\mathbf{W}$ connects the input layer to the hidden layer. Matrix $\mathbf{U}$ connects the hidden layer to the output layer. Define $\mathbf{W}$ so that its columns list all 2^n bipolar input vectors. Define $\mathbf{U}$ so that the columns of its transpose $\mathbf{U}^{\mathrm{T}}$ list all 2^n transposed bipolar output vectors:

$$\mathbf{W} = \begin{bmatrix}\mathbf{x_1} & \mathbf{x_2} & \ldots & \mathbf{x_{2^n}}\end{bmatrix}$$
$$\mathbf{U}^{\mathrm{T}} = \begin{bmatrix}\mathbf{y_1} & \mathbf{y_2} & \ldots & \mathbf{y_{2^n}}\end{bmatrix}.$$

We show next both that these weight matrices fire only one hidden neuron and that the forward pass of any input vector $\mathbf{x_n}$ gives the corresponding output vector $\mathbf{y_n}$. Assume that each neuron has zero bias.

Pick a bipolar input vector $\mathbf{x}_m$ for the forward pass. Then the input activation vector $\mathbf{a}^x(\mathbf{x}_m) = (a_1^x(x_m^1), \ldots, a_n^x(x_m^n))$ equals the input bipolar vector $\mathbf{x}_m$ because the input activations (9) are bipolar threshold functions with zero threshold. So $\mathbf{a}^x$ equals $\mathbf{x}_m$ because the vector space is bipolar $\{-1, 1\}^n$.

The hidden layer input $\mathbf{o}^h$ is the same as (2). It has the matrix-vector form

$$\mathbf{o}^h = \mathbf{W}^{\mathrm{T}}\mathbf{a}^x \tag{10}$$
$$= \mathbf{W}^{\mathrm{T}}\mathbf{x}_m \tag{11}$$
$$= \left(o_1^h, o_2^h, \ldots, o_n^h, \ldots, o_{2^n}^h\right)^{\mathrm{T}} \tag{12}$$
$$= \left(\mathbf{x}_1^{\mathrm{T}}\mathbf{x}_m, \mathbf{x}_2^{\mathrm{T}}\mathbf{x}_m, \ldots, \mathbf{x}_j^{\mathrm{T}}\mathbf{x}_m, \ldots, \mathbf{x}_{2^n}^{\mathrm{T}}\mathbf{x}_m\right)^{\mathrm{T}} \tag{13}$$

since o_j^h is the inner product of the bipolar vectors $\mathbf{x}_j$ and $\mathbf{x}_m$ from the definition of $\mathbf{W}$.

The input o_j^h to the jth neuron of the hidden layer obeys $o_j^h = n$ when $j = m$. It obeys $o_j^h < n$ when $j \neq m$. This holds because the vectors $\mathbf{x}_j$ are bipolar with scalar components in $\{-1, 1\}$. The magnitude of a bipolar vector in $\{-1, 1\}^n$ is $\sqrt{n}$. The inner product $\mathbf{x}_j^{\mathrm{T}}\mathbf{x}_m$ is a maximum when both vectors have the same direction. This occurs when $j = m$. The inner product is otherwise less than n. Fig. 3 shows a bidirectional neural network that fires just the sixth hidden neuron. The weights for the network in Fig. 3 are

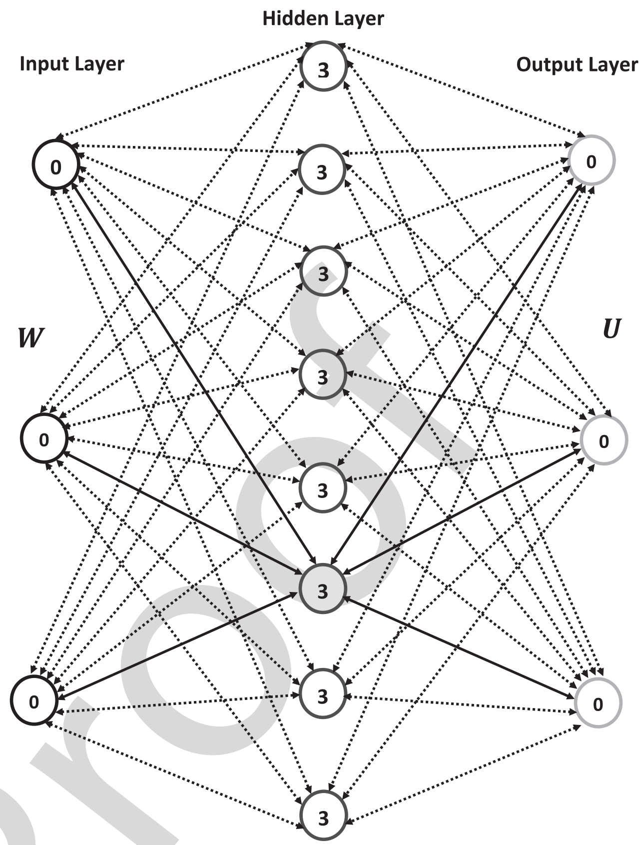

Fig. 3. Bidirectional network structure for the proof of Theorem 1. The input and output layers have n threshold neurons. The hidden layer has 2^n neurons with threshold values of n. The 8 fan-in 3-vectors of weights in $\mathbf{W}$ from the input to the hidden layer list the 2^3 elements of the bipolar cube $\{-1, 1\}^3$. So they list the eight vectors in the input column of Table I. The 8 fan-in 3-vectors of weights in $\mathbf{U}$ from the output to the hidden layer list the eight bipolar vectors in the output column of Table I. The threshold value for the sixth and highlighted hidden neuron is 3. Passing the sixth input vector $(-1, 1, -1)$ through $\mathbf{W}$ leads to the hidden-layer vector $(0, 0, 0, 0, 0, 1, 0, 0)$ of thresholded values. Passing this 8-bit vector through $\mathbf{U}$ produces after thresholding the sixth output vector $(-1, -1, -1)$ in Table I. Passing this output vector back through the transpose of $\mathbf{U}$ produces the same unit bit vector of thresholded hidden-unit values. Passing this vector back through the transpose of $\mathbf{W}$ produces the original bipolar vector $(-1, 1, -1)$.

$$\mathbf{W} = \begin{bmatrix} 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \end{bmatrix}$$
$$\mathbf{U}^T = \begin{bmatrix} -1 & -1 & 1 & 1 & -1 & -1 & 1 & 1 \\ -1 & 1 & 1 & -1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \end{bmatrix}.$$

Now comes the key step in the proof. Define the hidden activation a_j^h as a *binary* (not bipolar) threshold function where n is the threshold value

$$a_j^h\left(o_j^h\right) = \begin{cases} 1 & \text{if } o_j^h \geq n \\ 0 & \text{if } o_j^h < n. \end{cases} \tag{14}$$

Then the hidden-layer activation $\mathbf{a}^h$ is the *unit* bit vector $(0, 0, \ldots, 1, \ldots, 0)^{\mathrm{T}}$, where $a_j^h = 1$ when $j = m$ and where $a_j^h = 0$ when $j \neq m$. This holds because all 2^n bipolar vectors $\mathbf{x}_m$ in $\{-1, 1\}^n$ are distinct. So exactly one of these 2^n vectors achieves the maximal inner-product value $n = \mathbf{x}_m^{\mathrm{T}}\mathbf{x}_m$. So $a_j^h(o_j^h) = 0$ for $j \neq m$ and $a_m^h(o_m^h) = 1$. The bidirectional network in Fig. 3 represents the 3-bit bipolar permutation in Table I.

The input vector $\mathbf{o}^y$ to the output layer is

$$\mathbf{o}^y = \mathbf{U}^{\mathrm{T}}\mathbf{a}^h \tag{15}$$

$$= \sum_{j=1}^{J} \mathbf{y}_j \; a_j^h \tag{16}$$

$$= \mathbf{y}_m \tag{17}$$

where a_j^h is the activation of the jth hidden neuron. The activation $\mathbf{a}^y$ of the output layer is

$$\mathbf{a}^y\left(o_j^y\right) = \begin{cases} 1 & \text{if } \; o_j^y \geq 0 \\ -1 & \text{if } \; o_j^y < 0. \end{cases} \tag{18}$$

The output layer activation leaves $\mathbf{o}^y$ unchanged because $\mathbf{o}^y$ equals $\mathbf{y}_m$ and because $\mathbf{y}_m$ is a vector in $\{-1, 1\}^n$. So

$$\mathbf{a}^y = \mathbf{y}_m. \tag{19}$$

So the forward pass of an input vector $\mathbf{x}_m$ through the network yields the desired corresponding output vector $\mathbf{y}_m$ if $\mathbf{y}_m = f(\mathbf{x}_m)$ for the bipolar permutation map f.

Consider next the backward pass through the network N. The backward pass propagates the output vector $\mathbf{y}_m$ through the hidden layer back to the input layer. The hidden layer input $\mathbf{o}^{hb}$ has the same inner-product form as in (6):

$$\mathbf{o}^{hb} = \mathbf{U} \; \mathbf{y}_m \tag{20}$$

where $\mathbf{o}^{hb} = (\mathbf{y}_1^{\mathrm{T}}\mathbf{y}_m, \mathbf{y}_2^{\mathrm{T}}\mathbf{y}_m, \ldots, \mathbf{y}_j^{\mathrm{T}}\mathbf{y}_m, \ldots, \mathbf{y}_{2^n}^{\mathrm{T}}\mathbf{y}_m)^{\mathrm{T}}$.

The input o_j^{hb} of the jth neuron in the hidden layer equals the inner product of $\mathbf{y}_j$ and $\mathbf{y}_m$. So $o_j^{hb} = n$ when $j = m$. But now $o_j^{hb} < n$ when $j \neq m$. This holds because again the magnitude of a bipolar vector in $\{-1, 1\}^n$ is $\sqrt{n}$. The inner product o_j^{hb} is a maximum when vectors $\mathbf{y}_m$ and $\mathbf{y}_j$ lie in the same direction. The activation $\mathbf{a}^{hb}$ for the hidden layer has the same components as in (14). So the hidden-layer activation $\mathbf{a}^{hb}$ again equals the unit bit vector $(0, 0, \ldots, 1, \ldots, 0)^{\mathrm{T}}$ where $a_j^{hb} = 1$ when $j = m$ and $a_j^{hb} = 0$ when $j \neq m$.

Then the input vector $\mathbf{o}^{xb}$ for the input layer is

$$\mathbf{o}^{xb} = \mathbf{W} \; \mathbf{a}^{hb} \tag{21}$$

$$= \sum_{j=1}^{J} \mathbf{x}_j \; \mathbf{a}^{hb} \tag{22}$$

$$= \mathbf{x}_m. \tag{23}$$

The ith input neuron has a threshold activation that is the same as

$$a_i^{xb}\left(o_i^{xb}\right) = \begin{cases} 1 & \text{if } \; o_i^{xb} \geq 0 \\ -1 & \text{if } \; o_i^{xb} < 0 \end{cases} \tag{24}$$

where o_i^{xb} is the input of ith neuron in the input layer. This activation leaves $\mathbf{o}^{xb}$ unchanged because $\mathbf{o}^{xb}$ equals $\mathbf{x}_m$ and because the vector $\mathbf{x}_m$ lies in $\{-1, 1\}^n$. So

$$\mathbf{a}^{xb} = \mathbf{o}^{xb} \tag{25}$$

$$= \mathbf{x}_m. \tag{26}$$

So the backward pass of any target vector $\mathbf{y}_m$ yields the desired input vector $\mathbf{x}_m$ if $f^{-1}(\mathbf{y}_m) = \mathbf{x}_m$. This completes the backward pass and the proof. ■

III. Bidirectional Backpropagation Algorithms

A. *Double Regression*

We now derive the first of three B-BP learning algorithms. The first case is double regression where the network performs regression in both directions.

B-BP training minimizes both the forward error E_f and backward error E_b. B-BP alternates between backward training and forward training. Forward training minimizes E_f while holding E_b constant. Backward training minimizes E_b while holding E_f constant. E_f is the error at the output layer. E_b is the error at the input layer. Double regression uses squared error for both error functions.

The forward pass sends the input vector $\mathbf{x}$ through the hidden layer to the output layer. The network uses only one hidden layer for simplicity and with no loss of generality. The B-BP double-regression algorithm applies to any number of hidden layers in a deep network.

The hidden-layer input values o_j^h are the same as in (2). The jth hidden activation a_j^h is the binary logistic map

$$a_j^h\left(o_j^h\right) = \frac{1}{1 + e^{-o_j^h}} \tag{27}$$

where (4) gives the input o_k^y to the kth output neuron. The hidden activations can be logistic or any other sigmoidal function so long as they are differentiable. The activation for an output neuron is the identity function

$$a_k^y = o_k^y \tag{28}$$

where a_k^y is the activation of kth output neuron.

The error function E_f for the forward pass is squared error

$$E_f = \frac{1}{2}\sum_{k=1}^{K}\left(y_k - a_k^y\right)^2 \tag{29}$$

where y_k denotes the value of the kth neuron in the output layer. Ordinary unidirectional BP updates the weights and other network parameters by propagating the error from the output layer back to the input layer.

The backward pass sends the output vector $\mathbf{y}$ through the hidden layer to the input layer. The input to the jth hidden neuron o_j^{hb} is the same as in (6). The activation a_j^{hb} for the jth hidden neuron is

$$a_j^{hb} = \frac{1}{1 + e^{-o_j^{hb}}}. \tag{30}$$

The input o_i^x for the ith input neuron is the same as (8). The activation at the input layer is the identity function

$$a_i^{xb}\left(o_i^{xb}\right) = o_i^{xb}. \tag{31}$$

A nonlinear sigmoid (or Gaussian) activation can replace the linear function.

The backward-pass error E_b is also squared error

$$E_b = \frac{1}{2}\sum_{i=1}^{I}\left(x_i - a_i^x\right)^2. \tag{32}$$

The partial derivative of the hidden-layer activation in the forward direction is

$$\frac{\partial a_j^h}{\partial o_j^h} = \frac{\partial}{\partial o_j^h}\left(\frac{1}{1+e^{-o_j^h}}\right) \tag{33}$$

$$= \frac{e^{-o_j^h}}{\left(1+e^{-o_j^h}\right)^2} \tag{34}$$

$$= \frac{1}{1+e^{-o_j^h}}\left[1 - \frac{1}{1+e^{-o_j^h}}\right] \tag{35}$$

$$= a_j^h\left(1 - a_j^h\right). \tag{36}$$

Let $a_j^{h'}$ denote the derivative of a_j^h with respect to the inner-product term o_j^h. We again use the superscript b to denote the backward pass.

The partial derivative of E_f with respect to the weight u_{jk} is

$$\frac{\partial E_f}{\partial u_{jk}} = \frac{1}{2}\frac{\partial}{\partial u_{jk}}\sum_{k=1}^{K}\left(y_k - a_k^y\right)^2 \tag{37}$$

$$= \frac{\partial E_f}{\partial a_k^y}\,\frac{\partial a_k^y}{\partial o_k^y}\,\frac{\partial o_k^y}{\partial u_{jk}} \tag{38}$$

$$= \left(a_k^y - y_k\right)a_j^h. \tag{39}$$

The partial derivative of E_f with respect to w_{ij} is

$$\frac{\partial E_f}{\partial w_{ij}} = \frac{1}{2}\frac{\partial}{\partial w_{ij}}\sum_{k=1}^{K}\left(y_k - a_k^y\right)^2 \tag{40}$$

$$= \left(\sum_{k=1}^{K}\frac{\partial E_f}{\partial a_k^y}\,\frac{\partial a_k^y}{\partial o_k^y}\,\frac{\partial o_k^y}{\partial a_j^h}\right)\frac{\partial a_j^h}{\partial o_j^h}\,\frac{\partial o_j^h}{\partial w_{ij}} \tag{41}$$

$$= \sum_{k=1}^{K}\left(a_k^y - y_k\right)u_{jk}\;a_j^{h'}\;x_i \tag{42}$$

where $a_j^{h'}$ is the same as in (36). The partial derivative of E_f with respect to the bias b_k^y of the kth output neuron is

$$\frac{\partial E_f}{\partial b_k^y} = \frac{1}{2}\frac{\partial}{\partial b_k^y}\sum_{k=1}^{K}\left(y_k - a_k^y\right)^2 \tag{43}$$

$$= \frac{\partial E_f}{\partial a_k^y}\,\frac{\partial a_k^y}{\partial o_k^y}\,\frac{\partial o_k^y}{\partial b_k^y} \tag{44}$$

$$= a_k^y - y_k. \tag{45}$$

The partial derivative of E_f with respect to the bias b_j^h of the jth hidden neuron is

$$\frac{\partial E_f}{\partial b_j^h} = \frac{1}{2}\frac{\partial}{\partial b_j^h}\sum_{k=1}^{K}\left(y_k - a_k^y\right)^2 \tag{46}$$

$$= \left(\sum_{k=1}^{K}\frac{\partial E_f}{\partial a_k^y}\,\frac{\partial a_k^y}{\partial o_k^y}\,\frac{\partial o_k^y}{\partial a_j^h}\right)\frac{\partial a_j^h}{\partial o_j^h}\,\frac{\partial o_j^h}{\partial b_j^h} \tag{47}$$

$$= \sum_{k=1}^{K}\left(a_k^y - y_k\right)u_{jk}a_j^{h'} \tag{48}$$

where $a_j^{h'}$ is the same as in (36).

The partial derivative of the hidden-layer activation a_j^{hb} in the backward direction is

$$\frac{\partial a_j^{hb}}{\partial o_j^{hb}} = \frac{\partial}{\partial o_j^{hb}}\left(\frac{1}{1+e^{-o_j^{hb}}}\right) \tag{49}$$

$$= \frac{e^{-o_j^{hb}}}{\left(1+e^{-o_j^{hb}}\right)^2} \tag{50}$$

$$= \frac{1}{1+e^{-o_j^{hb}}}\left[1 - \frac{1}{1+e^{-o_j^{hb}}}\right] \tag{51}$$

$$= a_j^{hb}\left(1 - a_j^{hb}\right). \tag{52}$$

The partial derivative of E_b with respect to w_{ij} is

$$\frac{\partial E_b}{\partial w_{ij}} = \frac{1}{2}\frac{\partial}{\partial w_{ij}}\sum_{k=1}^{K}\left(x_i - a_i^{xb}\right)^2 \tag{53}$$

$$= \frac{\partial E_b}{\partial a_i^{xb}}\,\frac{\partial a_i^{xb}}{\partial o_i^{xb}}\,\frac{\partial o_i^{xb}}{\partial w_{ij}} \tag{54}$$

$$= \left(a_i^{xb} - x_i\right)a_j^{hb}. \tag{55}$$

The partial derivative of E_b with respect to u_{jk} is

$$\frac{\partial E_b}{\partial u_{jk}} = \frac{1}{2}\frac{\partial}{\partial u_{jk}}\sum_{i=1}^{I}\left(x_i - a_i^{xb}\right)^2 \tag{56}$$

$$= \left(\sum_{i=1}^{I}\frac{\partial E_b}{\partial a_i^{xb}}\,\frac{\partial a_i^{xb}}{\partial o_i^{xb}}\,\frac{\partial o_i^{xb}}{\partial a_j^{hb}}\right)\frac{\partial a_j^{hb}}{\partial o_j^{hb}}\,\frac{\partial o_j^{hb}}{\partial u_{jk}} \tag{57}$$

$$= \sum_{i=1}^{I}\left(a_i^{xb} - x_i\right)w_{ij}a_j^{hb'}y_k \tag{58}$$

where $a_j^{hb'}$ is the same as in (52).

The partial derivative of E_b with respect to the bias b_i^x of ith input neuron is

$$\frac{\partial E_b}{\partial b_i^x} = \frac{1}{2}\frac{\partial}{\partial b_i^x}\sum_{i=1}^{I}\left(x_i - a_i^{xb}\right)^2 \tag{59}$$

$$= \frac{\partial E_b}{\partial a_i^{xb}}\,\frac{\partial a_i^{xb}}{\partial o_i^{xb}}\,\frac{\partial o_i^{xb}}{\partial b_i^x} \tag{60}$$

$$= a_i^{xb} - x_i. \tag{61}$$

The partial derivative of E_b with respect to the bias b_j^h of jth hidden neuron is

$$\frac{\partial E_b}{\partial b_j^h} = \frac{1}{2}\frac{\partial}{\partial b_j^h}\sum_{i=1}^{I}\left(x_i - a_i^{xb}\right)^2 \quad (62)$$

$$= \left(\sum_{i=1}^{I}\frac{\partial E_b}{\partial a_i^{xb}}\frac{\partial a_i^{xb}}{\partial o_i^{xb}}\frac{\partial o_i^{xb}}{\partial a_j^{hb}}\right)\frac{\partial a_j^{hb}}{\partial o_j^{hb}}\frac{\partial o_j^{hb}}{\partial b_j^h} \quad (63)$$

$$= \sum_{i=1}^{I}\left(a_i^{xb} - x_i\right) w_{ij} a_j^{hb'} \quad (64)$$

where $a_j^{hb'}$ is the same as in (52).

The error function at the input layer is the backward-pass error E_b. The error function at the output layer is the forward-pass error E_f.

The above update laws for forward regression have the final form (for learning rate $\eta > 0$)

$$u_{jk}^{(n+1)} = u_{jk}^{(n)} - \eta\left(a_k^y - y_k\right)a_j^h \quad (65)$$

$$w_{ij}^{(n+1)} = w_{ij}^{(n)} - \eta\left(\sum_{k=1}^{K}\left(a_k^y - y_k\right)u_{jk}a_j^{h'}x_i\right) \quad (66)$$

$$b_j^{h(n+1)} = b_j^{h(n)} - \eta\left(\sum_{k=1}^{K}\left(a_k^y - y_k\right)u_{jk}a_j^{h'}\right) \quad (67)$$

$$b_k^{y(n+1)} = b_k^{y(n)} - \eta\left(a_k^y - y_k\right). \quad (68)$$

The dual update laws for backward regression have the final form

$$u_{jk}^{(n+1)} = u_{jk}^{(n)} - \eta\left(\sum_{i=1}^{I}\left(a_i^{xb} - x_i\right)w_{ij}a_j^{hb'}y_k\right) \quad (69)$$

$$w_{ij}^{(n+1)} = w_{ij}^{(n)} - \eta\left(a_i^{xb} - x_i\right)a_j^{hb} \quad (70)$$

$$b_i^{x(n+1)} = b_i^{x(n)} - \eta\left(a_i^{xb} - x_i\right) \quad (71)$$

$$b_j^{h(n+1)} = b_j^{h(n)} - \eta\left(\sum_{i=1}^{I}\left(a_i^{xb} - x_i\right)w_{ij}a_j^{hb'}\right). \quad (72)$$

B-BP training minimizes E_f while holding E_b constant. It then minimizes E_b while holding E_f constant. Equations (65)–(68) state the update rules for forward training. Equations (69)–(72) state the update rules for backward training. Each training iteration involves forward training and then backward training.

Algorithm 1 summarizes the B-BP algorithm. It shows how to combine forward and backward training in B-BP. Fig. 6 shows how double-regression B-BP approximates the invertible function $f(x) = 0.5\sigma(6x + 3) + 0.5\sigma(4x - 1.2)$ if $\sigma(x)$ denotes the bipolar logistic function in (1). The approximation used a deep 8-layer network with six layers of ten bipolar logistic neurons each. The input and output layer each contained only a single identity neuron.

B. Double Classification

We now derive a B-BP algorithm where the network's forward pass acts as a classifier network and so does its backward pass. We call this double classification.

We present the derivation in terms of cross entropy for the sake of simplicity. Our double-classification simulations used the slightly more general form of cross entropy in (114) that we call *logistic* cross entropy. The simpler cross-entropy derivation applies to softmax input neurons and output neurons (with implied 1-in-K coding). Logistic input and output neurons require logistic cross entropy for the same BP derivation because then the same final BP partial derivatives result.

The simplest double-classification network uses Gibbs or softmax neurons at both the input and output layers. This creates a winner-take-all structure at those layers. Then the kth softmax neuron in the output layer codes for the kth input pattern. The output layer represents the pattern as a K-length unit bit vector with a "1" in the kth slot and a "0" in the other $K - 1$ slots [3], [19]. The same 1-in-I binary encoding holds for the ith neuron at the input layer. The softmax structure implies that the input and output fields each compute a discrete probability distribution for each input.

Classification networks differ from regression networks in another key aspect: they do not minimize squared error. They instead minimize the *cross entropy* of the given target vector and the softmax activation values of the output or input layers [3]. Equation (79) states the forward cross entropy at the output layer if y_k is the desired or target value of the kth output neuron. Then a_k^y is its actual softmax activation value. The entropy structure applies because both the target vector and the input and output vectors are probability vectors. Minimizing the cross entropy maximizes the Kullback–Leibler divergence [20] and vice versa [19].

The classification BP algorithm depends on another optimization equivalence: minimizing the cross entropy is equivalent to maximizing the network's likelihood or log-likelihood [19]. We will establish this equivalence because it implies that the *BP learning laws have the same form for both classification and regression*. We will prove the equivalence for only the forward direction. It applies equally in the backward direction. The result unifies the BP learning laws. It also allows carefully selected noise to enhance the network likelihood because BP is a special case [19], [21] of the expectation–maximization algorithm for iteratively maximizing a likelihood with missing data or hidden variables [22].

Denote the network's forward probability density function as $p_f(\mathbf{y}|\mathbf{x}, \Theta)$. The vector Θ lists all parameters in the network. The input vector $\mathbf{x}$ passes through the multilayer network and produces the output vector $\mathbf{y}$. Then the network's forward likelihood $L_f(\Theta)$ is the natural logarithm of the forward network probability: $L_f(\Theta) = \ln p_f(\mathbf{y}|\mathbf{x}, \Theta)$.

We will show that $p_f(\mathbf{y}|\mathbf{x}, \Theta) = \exp\{-E_f(\Theta)\}$. So BP's forward pass computes the forward cross entropy as it maximizes the likelihood [19].

The key assumption is that output softmax neurons in a classifier network are independent because there are no intralayer connections among them. Then the network probability density $p_f(\mathbf{y}|\mathbf{x}, \Theta)$ factors into a product of K-many marginals [3]: $p_f(\mathbf{y}|\mathbf{x}, \Theta) = \prod_{k=1}^{K} p_f(y_k|\mathbf{x}, \Theta)$. This gives

$$L_f(\Theta) = \ln p_f(\mathbf{y}|\mathbf{x}, \Theta) \quad (73)$$

$$= \ln \prod_{k=1}^{K} p_f(y_k|\mathbf{x}, \Theta) \tag{74}$$

$$= \ln \prod_{k=1}^{K} \left(a_k^y\right)^{y_k} \tag{75}$$

$$= \sum_{k=1}^{K} y_k \ln a_k^y \tag{76}$$

$$= -E_f(\Theta) \tag{77}$$

from (79) since $\mathbf{y}$ is a 1-in-K-encoded unit bit vector. Then exponentiation gives $p_f(\mathbf{y}|\mathbf{x}, \Theta) = \exp\{-E_f(\Theta)\}$. Minimizing the forward cross entropy E_f is equivalent to maximizing the negative cross entropy $-E_f$. So minimizing E_f maximizes the forward network likelihood L and vice versa.

The third equality (75) holds because the kth marginal factor $p_f(y_k|\mathbf{x}, \Theta)$ in a classifier network equals the exponentiated softmax activation $(a_k^t)^{y_k}$. This holds because $y_k = 1$ if k is the correct class label for the input pattern $\mathbf{x}$ and $y_k = 0$ otherwise. This discrete probability vector defines an output categorical distribution. It is a single-sample multinomial.

We now derive the B-BP algorithm for double classification. The algorithm minimizes the error functions separately where $E_f(\Theta)$ is the forward cross entropy in (75) and $E_b(\Theta)$ is the backward cross entropy in (81). We first derive the forward B-BP classifier algorithm. We then derive the backward portion of the B-BP double-classification algorithm.

The forward pass sends the input vector $\mathbf{x}$ through the hidden layer or layers to the output layer. The input activation vector $\mathbf{a}^x$ is the vector $\mathbf{x}$.

We assume only one hidden layer for simplicity. The derivation applies to deep networks with any number of hidden layers. The input to the jth hidden neuron o_j^h has the same linear form as in (2). The jth hidden activation a_j^h is the same ordinary unit-interval-valued logistic function in (27). The input o_k^y to the kth output neuron is the same as in (4). The hidden activations can also be ReLU or hyperbolic tangents or many other functions.

The forward classifier's output-layer neurons use Gibbs or softmax activations

$$a_k^y = \frac{e^{\left(o_k^y\right)}}{\sum_{l=1}^{K} e^{\left(o_l^y\right)}} \tag{78}$$

where a_k^y is the activation of the kth output neuron. Then the forward error E_f is the cross entropy

$$E_f = -\sum_{k=1}^{K} y_k \ln \ a_k^y \tag{79}$$

between the binary target values y_k and the actual output activations a_k^y.

We next describe the backward pass through the classifier network. The backward pass sends the output target vector $\mathbf{y}$ through the hidden layer to the input layer. So the initial activation vector $\mathbf{a}^{\mathbf{y}}$ equals the target vector $\mathbf{y}$. The input to the jth neuron of the hidden layer o_j^{hb} has the same linear form as (6). The activation of the jth hidden neuron is the same as (30).

The backward-pass input to the ith input neuron is also the same as (8). The input activation is Gibbs or softmax

$$a_i^{xb} = \frac{e^{\left(o_i^{xb}\right)}}{\sum_{l=1}^{I} e^{\left(o_i^{xb}\right)}} \tag{80}$$

where a_i^{xb} is the backward-pass activation for the ith neuron of the input neuron. Then the backward error E_b is the cross entropy

$$E_b = -\sum_{i=1}^{I} x_i \ln \ a_i^{xb} \tag{81}$$

where x_i is the target value of the ith input neuron.

The partial derivatives of the hidden activation a_j^h and a_j^{hb} are the same as in (36) and (52).

The partial derivative of the output activation a_k^y for the forward classification pass is

$$\frac{\partial a_k^y}{\partial o_k^y} = \frac{\partial}{\partial o_k^y}\left(\frac{e^{\left(o_k^y\right)}}{\sum_{l=1}^{K} e^{\left(o_l^y\right)}}\right) \tag{82}$$

$$= \frac{e^{o_k^y}\left(\sum_{l=1}^{K} e^{\left(o_l^y\right)}\right) - e^{o_k^y} e^{o_k^y}}{\left(\sum_{l=1}^{K} e^{\left(o_l^y\right)}\right)^2} \tag{83}$$

$$= \frac{e^{o_k^y}\left(\sum_{l=1}^{K} e^{\left(o_l^y\right)} - e^{o_k^y}\right)}{\left(\sum_{l=1}^{K} e^{\left(o_l^y\right)}\right)^2} \tag{84}$$

$$= a_k^y\left(1 - a_k^y\right). \tag{85}$$

The partial derivative when $l \neq k$ is

$$\frac{\partial a_k^y}{\partial o_l^y} = \frac{\partial}{\partial o_l^y}\left(\frac{e^{\left(o_k^y\right)}}{\sum_{m=1}^{K} e^{\left(o_m^y\right)}}\right) \tag{86}$$

$$= \frac{-e^{o_k^y} e^{o_l^y}}{\left(\sum_{l=1}^{K} e^{\left(o_l^y\right)}\right)^2} \tag{87}$$

$$= -a_k^y \ a_l^y. \tag{88}$$

So the partial derivative of a_k^y with respect to o_l^k is

$$\frac{\partial a_k^y}{\partial o_l^y} = \begin{cases} -a_k^y \ a_l^y & \text{if} \quad l \neq k \\ a_k^y\left(1 - a_k^y\right) & \text{if} \quad l = k. \end{cases} \tag{89}$$

Denote this derivative as $a_k^{y'}$. The derivative $a_i^{xb'}$ of the backward classification pass has the same form because both sets of classifier neurons have softmax activations.

The partial derivative of the forward cross entropy E_f with respect to u_{jk} is

$$\frac{\partial E_f}{\partial u_{jk}} = -\frac{\partial}{\partial u_{jk}} \sum_{k=1}^{K} y_k \ln \ a_k^y \tag{90}$$

$$= \sum_{k=1}^{K}\left(\frac{\partial E_f}{\partial a_k^y} \frac{\partial a_k^y}{\partial o_k^y} \frac{\partial o_k^y}{\partial u_{jk}}\right) \tag{91}$$

$$= -\left(\frac{y_k}{a_k^y}\left(1 - a_k^y\right) a_k^y - \sum_{l \neq k}^{K} \frac{y_l}{a_l^y} a_k^y a_l^y\right) a_j^h \tag{92}$$

$$= \left(a_k^y - y_k\right) a_j^h. \tag{93}$$

The partial derivative of the forward cross entropy E_f with respect to the bias b_k^y of the kth output neuron is

$$\frac{\partial E_f}{\partial b_k^y} = \frac{\partial}{\partial b_k^y} \sum_{k=1}^{K} y_k \ln \ a_k^y \tag{94}$$

$$= \sum_{k=1}^{K} \left(\frac{\partial E_f}{\partial a_k^y} \ \frac{\partial a_k^y}{\partial o_k^y} \ \frac{\partial o_k^y}{\partial b_k^y} \right) \tag{95}$$

$$= -\left(\frac{y_k}{a_k^y} \big(1 - a_k^y\big) a_k^y - \sum_{l \neq k}^{K} \frac{y_l}{a_l^y} a_k^y a_l^y \right) \tag{96}$$

$$= a_k^y - y_k. \tag{97}$$

Equations (93) and (97) show that the derivatives of E_f with respect to u_{jk} and b_k^y for double classification are the same as for double regression in (39) and (45). The activations of the hidden neurons are the same as for double regression. So the derivatives of E_f with respect to w_{ij} and b_j^h are the same as the respective ones in (42) and (48).

The partial derivative of E_b with respect to w_{ij} is

$$\frac{\partial E_b}{\partial w_{ij}} = -\frac{\partial}{\partial w_{ij}} \sum_{i=1}^{I} x_i \ \ln \ a_i^{xb} \tag{98}$$

$$= \sum_{i=1}^{I} \left(\frac{\partial E_b}{\partial a_i^{xb}} \ \frac{\partial a_i^{xb}}{\partial o_i^{xb}} \ \frac{\partial o_i^{xb}}{\partial w_{ij}} \right) \tag{99}$$

$$= -\left(\frac{x_i}{a_i^{xb}} \Big(1 - a_i^{xb}\Big) a_i^{xb} - \sum_{l \neq i}^{I} \frac{x_l}{a_l^{xb}} a_i^{xb} a_l^{xb} \right) a_j^{hb} \tag{100}$$

$$= \Big(a_i^{xb} - x_i\Big) a_j^{hb}. \tag{101}$$

The partial derivative of E_b with respect to the bias b_i^x of the ith input neuron is

$$\frac{\partial E_b}{\partial b_i^x} = -\frac{\partial}{\partial b_i^{xb}} \sum_{i=1}^{I} x_i \ \ln \ a_i^{xb} \tag{102}$$

$$= \sum_{i=1}^{I} \left(\frac{\partial E_b}{\partial a_i^{xb}} \ \frac{\partial a_i^{xb}}{\partial o_i^{xb}} \ \frac{\partial o_i^{xb}}{\partial b_i^x} \right) \tag{103}$$

$$= -\left(\frac{x_i}{a_i^{xb}} \Big(1 - a_i^{xb}\Big) a_i^{xb} - \sum_{l \neq i}^{I} \frac{x_l}{a_l^{xb}} a_i^{xb} a_l^{xb} \right) \tag{104}$$

$$= a_i^{xb} - x_i. \tag{105}$$

Equations (101) and (105) likewise show that the derivatives of E_b with respect to w_{ij} and b_i^x for double classification are the same as for double regression in (53) and (59). The activations of the hidden neurons are the same as for double regression. So the derivatives of E_b with respect to u_{jk} and b_j^h are the same as the respective ones in (58) and (64).

B-BP training for double classification also alternates between minimizing E_f while holding E_b constant and minimizing E_b while holding E_f constant. The forward and backward errors are again cross entropies.

The update laws for forward classification have the final form

$$u_{jk}^{(n+1)} = u_{jk}^{(n)} - \eta \Big(\big(a_k^y - y_k\big) a_j^h \Big) \tag{106}$$

$$w_{ij}^{(n+1)} = w_{ij}^{(n)} - \eta \left(\sum_{k=1}^{K} \big(a_k^y - y_k\big) u_{jk} a_j^{h'} x_i \right) \tag{107}$$

$$b_j^{h(n+1)} = b_j^{h(n)} - \eta \left(\sum_{k=1}^{K} \big(a_k^y - y_k\big) u_{jk} a_j^{h'} \right) \tag{108}$$

$$b_k^{y(n+1)} = b_k^{y(n)} - \eta \big(a_k^y - y_k\big). \tag{109}$$

The dual update laws for backward classification have the final form

$$u_{jk}^{(n+1)} = u_{jk}^{(n)} - \eta \left(\sum_{i=1}^{I} \Big(a_i^{xb} - x_i\Big) w_{ij} a_j^{hb'} y_k \right) \tag{110}$$

$$w_{ij}^{(n+1)} = w_{ij}^{(n)} - \eta \Big(\Big(a_i^{xb} - x_i\Big) a_j^{hb} \Big) \tag{111}$$

$$b_i^{x(n+1)} = b_i^{x(n)} - \eta \Big(a_i^{xb} - x_i\Big) \tag{112}$$

$$b_j^{h(n+1)} = b_j^{h(n)} - \eta \left(\sum_{i=1}^{I} \Big(a_i^{xb} - x_i\Big) w_{ij} a_j^{hb'} \right). \tag{113}$$

The derivation shows that the update rules for double classification are the same as the update rules for double regression.

B-BP training minimizes E_f while holding E_b constant. It then minimizes E_b while holding E_f constant. Equations (106)–(109) are the update rules for forward training. Equations (110)–(113) are the update rules for backward training. Each training iteration involves first running forward training and then running backward training. Algorithm 1 again summarizes the B-BP algorithm.

The more general case of double classification uses logistic neurons at the input and output layer. Then the BP derivation requires the slightly more general *logistic* cross-entropy performance measure. We used the logistic cross-entropy $E_{\log}$ for double classification training because the input and output neurons were logistic (rather than softmax)

$$E_{\log} = -\sum_{k=1}^{K} y_k \ln a_k^y - \sum_{k=1}^{K} (1 - y_k) \ln\big(1 - a_k^y\big). \tag{114}$$

Partially differentiating $E_{\log}$ for logistic input and output neurons gives back the same B-BP learning laws as does differentiating cross entropy for softmax input and output neurons.

C. Mixed Case: Classification and Regression

We last derive the B-BP learning algorithm for the mixed case of a neural classifier network in the forward direction and a regression network in the backward direction.

This mixed case describes the common case of neural image classification. The user needs only add backward-regression training to allow the same classifier net to predict which image input produced a given output classification. Backward regression estimates this answer as the centroid of the inverse set-theoretic mapping or preimage. The B-BP algorithm achieves this by alternating between minimizing E_f and minimizing E_b. The forward error E_f is the same as the cross entropy in the double-classification network above. The backward error E_b is the same as the squared error in double regression.

The input space is likewise the I-dimensional real space $\mathbb{R}^I$ for regression. The output space uses 1-in-K binary encoding for classification. The output neurons of regression networks use identity functions as activations. The output neurons of classifier networks use softmax activations.

The forward pass sends the input vector $\mathbf{x}$ through the hidden layer to the output layer. The input activation vector $\mathbf{a}^x$ equals $\mathbf{x}$. We again consider only a single hidden layer for simplicity. The input o_j^h to the jth hidden neuron is the same as in (2). The activation a_j^h of the jth hidden layer is the ordinary logistic activation in (27). Equation (4) defines the input o_k^y to the kth output neuron. The output activation is softmax. So the output activation a_k^y is the same as in (78). The forward error E_f is the cross entropy in (79). The forward pass in this mixed case is the same as the forward pass for double classification. So (42), (48), (93), and (97) give the respective derivatives of the forward error E_f with respect to w_{ij}, b_j^h, u_{jk}, and b_y^k.

The backward pass propagates the 1-in-K vector $\mathbf{y}$ from the output through the hidden layer to the input layer. The output layer activation vector $\mathbf{a}^{\mathbf{y}}$ equals $\mathbf{y}$. The input o_j^{hb} to the jth hidden neuron for the backward pass is the same as in (6). Equation (30) gives the activation a_j^{hb} for the jth hidden unit in the backward pass. Equation (8) gives the input o_i^{xb} for the ith input neuron. The activation a_i^{xb} of the ith input neuron for the backward pass is the same as in (31). The backward error E_b is the squared error in (32).

The backward pass in this mixed case is the same as the backward pass for double regression. So (55), (58), (61), and (64) give the respective derivatives of the backward error E_b with respect to w_{ij}, b_i^x, u_{jk}, and b_j^h.

The update laws for forward classification–regression training have the final form

$$u_{jk}^{(n+1)} = u_{jk}^{(n)} - \eta\left(a_k^y - y_k\right)a_j^h \tag{115}$$

$$w_{ij}^{(n+1)} = w_{ij}^{(n)} - \eta\left(\sum_{k=1}^{K}\left(a_k^y - y_k\right)u_{jk}a_j^{h'}x_i\right) \tag{116}$$

$$b_j^{h(n+1)} = b_j^{h(n)} - \eta\left(\sum_{k=1}^{K}\left(a_k^y - y_k\right)u_{jk}a_j^{h'}\right) \tag{117}$$

$$b_k^{y(n+1)} = b_k^{y(n)} - \eta\left(a_k^y - y_k\right). \tag{118}$$

The update laws for backward classification–regression training have the final form

$$u_{jk}^{(n+1)} = u_{jk}^{(n)} - \eta\left(\sum_{i=1}^{I}\left(a_i^{xb} - x_i\right)w_{ij}a_j^{hb'}y_k\right) \tag{119}$$

$$w_{ij}^{(n+1)} = w_{ij}^{(n)} - \eta\left(a_i^{xb} - x_i\right)a_j^{hb} \tag{120}$$

$$b_i^{x(n+1)} = b_i^{x(n)} - \eta\left(a_i^{xb} - x_i\right) \tag{121}$$

TABLE II
5-BIT BIPOLAR PERMUTATION FUNCTION

Input x	Output t	Input x	Output t
[− − − − −]	[+ + − + +]	[+ − − − −]	[− + + + +]
[− − − − +]	[− − + − −]	[+ − − − +]	[− + − − −]
[− − − + −]	[− − − + −]	[+ − − + −]	[+ − − + −]
[− − − + +]	[+ + + − +]	[+ − − + +]	[− − + − +]
[− − + − −]	[+ + − + −]	[+ − + − −]	[− + − + +]
[− − + − +]	[+ − − + +]	[+ − + − +]	[+ + − − +]
[− − + + −]	[− + + − +]	[+ − + + −]	[+ + + + +]
[− − + + +]	[− − + + +]	[+ − + + +]	[− − + + −]
[− + − − −]	[+ − + + +]	[+ + − − −]	[+ + + − −]
[− + − − +]	[+ − − − +]	[+ + − − +]	[− + − + −]
[− + − + −]	[+ − + + −]	[+ + − + −]	[+ − − − −]
[− + − + +]	[− + + − −]	[+ + − + +]	[− − − + +]
[− + + − −]	[− + + + −]	[+ + + − −]	[− − − − −]
[− + + − +]	[+ + − − −]	[+ + + − +]	[− + − − +]
[− + + + −]	[+ − + − +]	[+ + + + −]	[+ + + + −]
[− + + + +]	[− − − − +]	[+ + + + +]	[+ − + − −]

TABLE III
FORWARD-PASS CROSS ENTROPY E_f

Hidden Neurons	Backpropagation Training		
	Forward	Backward	Bidirectional
5	0.4222	1.4534	0.4729
10	0.0881	1.8173	0.3045
20	0.0132	4.7554	0.0539
50	0.0037	4.4039	0.0034
100	0.0014	5.8473	0.0029

$$b_j^{h(n+1)} = b_j^{h(n)} - \eta\left(\sum_{i=1}^{I}\left(a_i^{xb} - x_i\right)w_{ij}a_j^{hb'}\right). \tag{122}$$

B-BP training minimizes E_f while holding E_b constant. It then minimizes the E_b while holding E_f constant. Equations (115)–(118) state the update rules for forward training. Equations (119)–(122) state the update rules for backward training. Algorithm 1 shows how forward learning combines with backward learning in B-BP.

IV. SIMULATION RESULTS

We tested the B-BP algorithm for double classification on a 5-bit permutation function. We used 3-layer networks with different numbers of hidden neurons. The neurons used bipolar logistic activations. The performance measure was the logistic cross entropy in (114). The B-BP algorithm produced either an exact representation or an approximation. The permutation function bijectively mapped the 5-bit bipolar vector space $\{-1, 1\}^5$ of 32 bipolar vectors onto itself. Table II displays the permutation test function. We compared the forward and backward forms of unidirectional BP with B-BP. We also tested whether adding more hidden neurons improved network approximation accuracy.

The forward pass of standard BP used logistic cross entropy as its error function. The backward pass did as well. B-BP summed the forward and backward errors for its joint error. We computed the test error for the forward and backward passes. Each plotted error value averaged 20 runs.

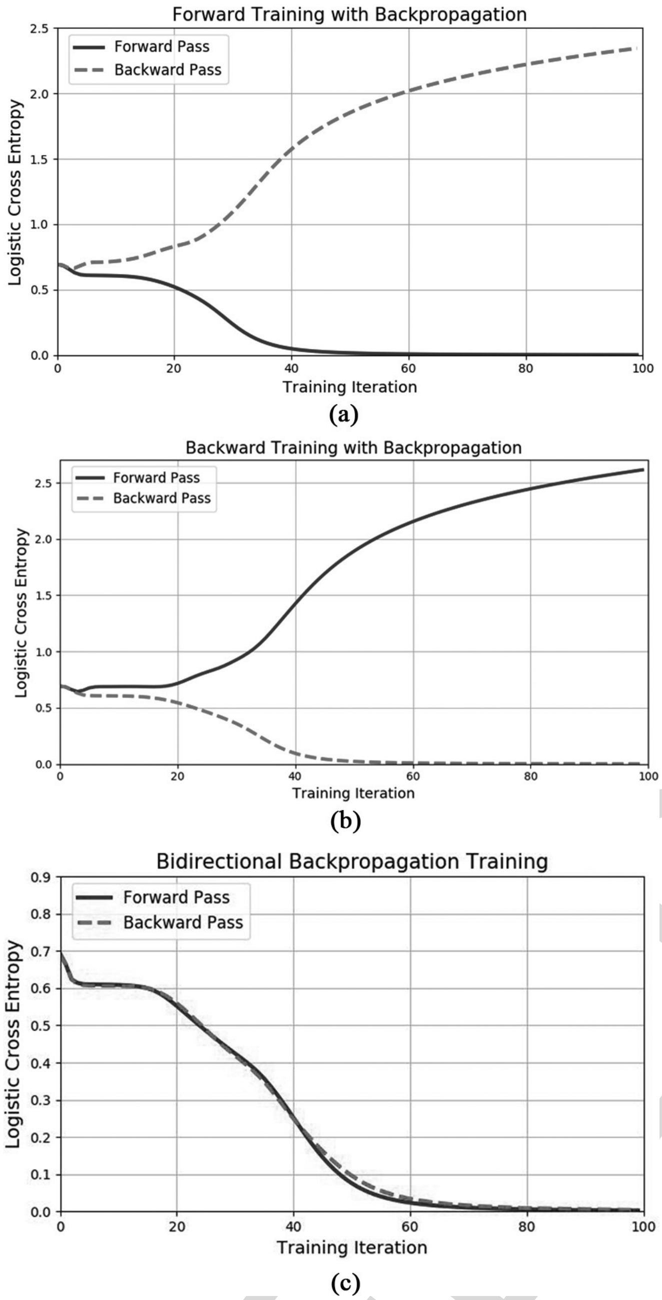

Fig. 4. Logistic-cross-entropy learning for double classification using 100 hidden neurons with forward BP training, backward BP training, and B-BP training. The trained network represents the 5-bit permutation function in Table II. (a) Forward BP tuned the network with respect to logistic cross entropy for the forward pass using E_f only. (b) Backward BP training tuned the network with respect to logistic cross entropy for the backward pass using E_b only. (c) B-BP training summed the logistic cross entropies for both the forward-pass error term E_f and the backward-pass error term E_b to update the network parameters.

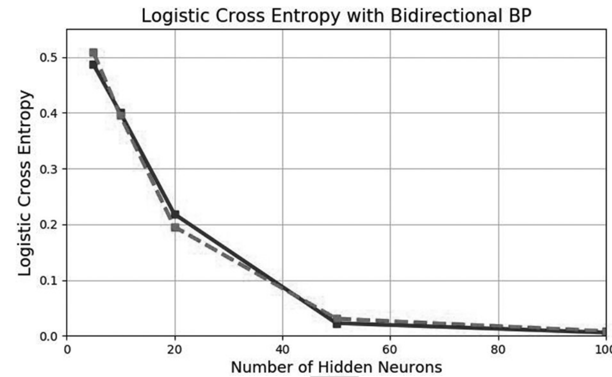

Fig. 5. B-BP training error for the 5-bit permutation in Table II using different numbers of hidden neurons. Training used the double-classification B-BP algorithm. The two curves describe the logistic cross entropy for the forward and backward passes through the 3-layer network. Each test used 640 samples. The number of hidden neurons increased from 5, 10, 20, 50, to 100.

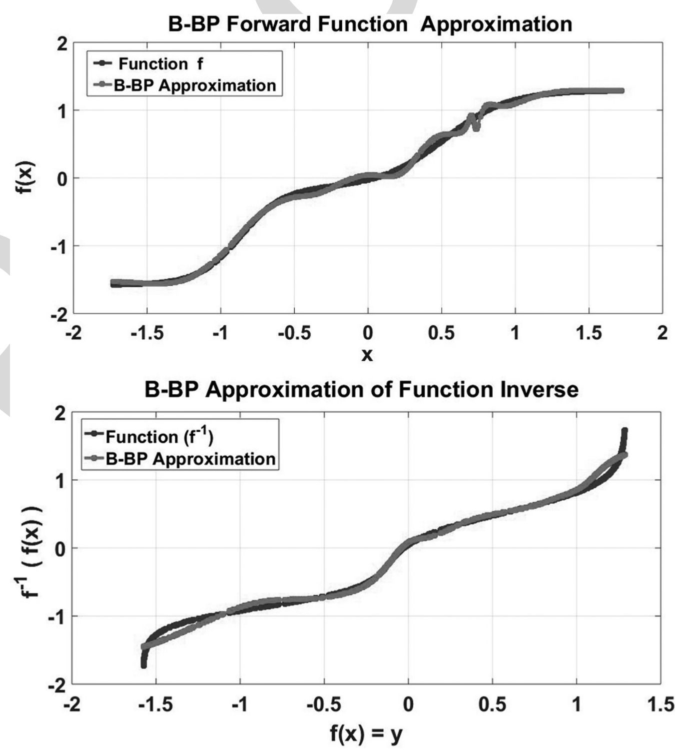

Fig. 6. B-BP double-regression approximation of the invertible function $f(x) = 0.5\sigma(6x + 3) + 0.5\sigma(4x - 1.2)$ using a deep 8-layer network with six hidden layers. The function σ denotes the bipolar logistic function in (1). Each hidden layer contained ten bipolar logistic neurons. The input and output layers each used a single neuron with an identity activation function. The forward pass approximated the forward function f. The backward pass approximated the inverse function f^{-1}.

Fig. 4 shows the results of running the three types of BP learning for classification on a 3-layer network with 100 hidden neurons. The values of E_f and E_b decrease with an increase in the training iterations for B-BP. This was not the case for the unidirectional cases of forward BP and backward BP training. Forward and backward training performed well only for function approximation in their respective training direction. Neither performed well in the opposite direction.

Table III shows the forward-pass cross entropy E_f for learning 3-layer classification neural networks as the number of hidden neurons grows. We again compared the three forms of BP for the network training: two forms of unidirectional BP and B-BP. The forward-pass error for forward BP fell substantially as the number of hidden neurons grew. The forward-pass error of backward BP decreased slightly as the number of hidden neurons grew. It gave the worst performance. B-BP performed well on the test set. Its forward-pass error also

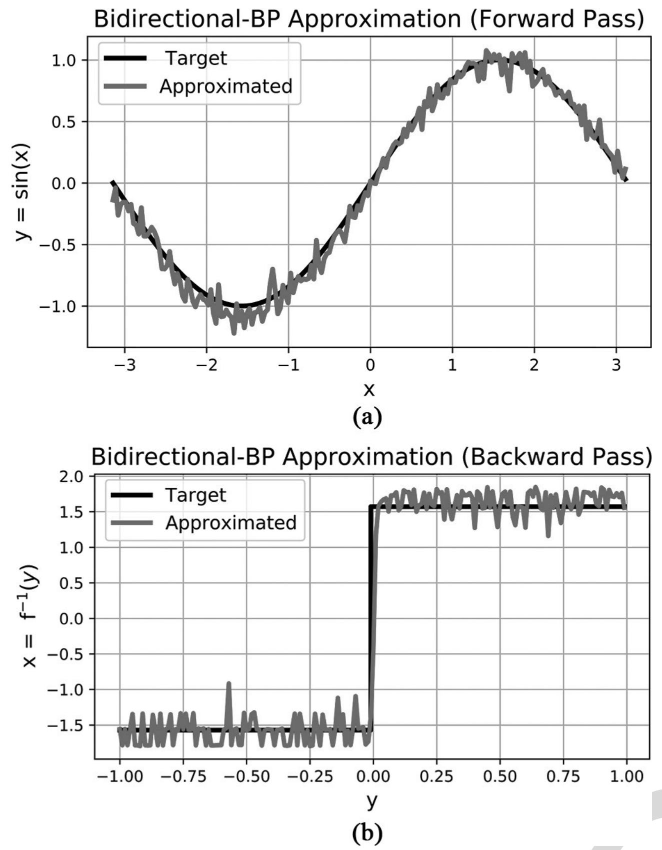

Fig. 7. B-BP double-regression learning of the noninvertible target function $f(x) = \sin x$. (a) Forward pass learned the function $y = f(x) = \sin x$. (b) Backward pass approximated the centroid of the values in the set-theoretic preimage $f^{-1}(\{y\})$ for y values in $(-1, 1)$. The two centroids were $-(\pi/2)$ and $(\pi/2)$.

TABLE IV
BACKWARD-PASS CROSS ENTROPY E_b

	Backpropagation Training		
Hidden Neurons	Forward	Backward	Bidirectional
5	2.9370	0.3572	0.4692
10	2.4920	0.1053	0.3198
20	4.6432	0.0149	0.0542
50	7.0921	0.0027	0.0040
100	7.1414	0.0013	0.0032

fell substantially as the number of hidden neurons grew. Table IV shows similar error-versus-hidden-neuron results for the backward-pass cross entropy E_b.

The two tables jointly show that the unidirectional forms of BP for regression performed well only in one direction. The B-BP algorithm performed well in both directions.

We tested the B-BP algorithm for double regression with the invertible function $f(x) = 0.5\sigma(6x + 3) + 0.5\sigma(4x - 1.2)$ for values of $x \in [-1.5, 1.5]$. We used a deep 8-layer network with 6 hidden layers for this approximation. Each hidden layer had 10 bipolar logistic neurons. There was only a single identity neuron in the input and output layers. The error functions E_f and E_b were ordinary squared error. Fig. 6 compares the B-BP approximation with the target function for both the forward pass and the backward pass.

Algorithm 1 B-BP Algorithm

Data: $\boldsymbol{T}$ input vectors $\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(T)}\}$ and $\boldsymbol{T}$ corresponding output vectors $\{\mathbf{y}^{(1)}, \mathbf{y}^{(2)}, \ldots, \mathbf{y}^{(T)}\}$ such that $f(\mathbf{x}^{(t)}) = \mathbf{y}^{(t)}$. Number of hidden neurons $\boldsymbol{J}$. Batch size $\boldsymbol{S}$ and number of epochs $\boldsymbol{R}$. Choose the learning rate $\boldsymbol{\eta}$.

Result: Bidirectional neural network representation for function $\boldsymbol{f}$.

Initialize: Randomly select the initial weights $\boldsymbol{W}^{(0)}$ and $\boldsymbol{U}^{(0)}$. Randomly pick the bias weights for input, hidden, and output neurons $\{\boldsymbol{b}^{x(0)}, \boldsymbol{b}^{h(0)}, \boldsymbol{b}^{y(0)}\}$.

while epoch $\boldsymbol{r} : 0 \longrightarrow \boldsymbol{R}$ **do**

Select $\boldsymbol{S}$ random samples from the training dataset.

Initialize: $\Delta\boldsymbol{W} = 0$, $\Delta\boldsymbol{U} = 0$, $\Delta\boldsymbol{b}^{x} = 0$, $\Delta\boldsymbol{b}^{h} = 0$, $\Delta\boldsymbol{b}^{y} = 0$.

FORWARD TRAINING

while ***batch_size*** $\boldsymbol{l} : 1 \longrightarrow \boldsymbol{S}$

- Randomly pick input vector $\mathbf{x}^{(l)}$ and its corresponding output vector $\mathbf{y}^{(l)}$
- Compute hidden layer input $\mathbf{o}^{h}$ and the corresponding hidden activation $\mathbf{a}^{h}$
- Compute output layer input $\mathbf{o}^{y}$ and the corresponding output activation $\mathbf{a}^{y}$
- Compute the forward error E_f
- Compute the following derivatives: $\nabla_{\boldsymbol{W}}E_f$, $\nabla_{\boldsymbol{U}}E_f$, $\nabla_{\boldsymbol{b}^h}E_f$, and $\nabla_{\boldsymbol{b}^y}E_f$
- Update : $\Delta\boldsymbol{W} = \Delta\boldsymbol{W} + \nabla_{\boldsymbol{W}}E_f$; $\Delta\boldsymbol{b}^{h} = \Delta\boldsymbol{b}^{h} + \nabla_{\boldsymbol{b}^h}E_f$
  $\Delta\boldsymbol{U} = \Delta\boldsymbol{U} + \nabla_{\boldsymbol{U}}E_f$; $\Delta\boldsymbol{b}^{y} = \Delta\boldsymbol{b}^{y} + \nabla_{\boldsymbol{b}^y}E_f$

End

BACKWARD TRAINING

while ***batch_size*** $\boldsymbol{l} : 1 \longrightarrow \boldsymbol{S}$

- Pick input vector $\mathbf{x}^{(l)}$ and its corresponding output vector $\mathbf{y}^{(l)}$.
- Compute hidden layer input $\mathbf{o}^{hb}$ and hidden activation $\mathbf{a}^{hb}$.
- Compute input $\mathbf{o}^{xb}$at the input layer and input activation $\mathbf{a}^{xb}$.
- Compute the backward error E_b
- Compute the following derivatives: $\nabla_{\boldsymbol{W}}E_b$, $\nabla_{\boldsymbol{U}}E_b$, $\nabla_{\boldsymbol{b}^h}E_b$, and $\nabla_{\boldsymbol{b}^x}E_b$
- Update : $\Delta\boldsymbol{W} = \Delta\boldsymbol{W} + \nabla_{\boldsymbol{W}}E_b$; $\Delta\boldsymbol{b}^{h} = \Delta\boldsymbol{b}^{h} + \nabla_{\boldsymbol{b}^h}E_b$
  $\Delta\boldsymbol{U} = \Delta\boldsymbol{U} + \nabla_{\boldsymbol{U}}E_b$; $\Delta\boldsymbol{b}^{x} = \Delta\boldsymbol{b}^{x} + \nabla_{\boldsymbol{b}^x}E_b$

End

Update:

- $\boldsymbol{W}^{(r+1)} = \boldsymbol{W}^{(r)} - \eta\Delta\boldsymbol{W}$
- $\boldsymbol{U}^{(r+1)} = \boldsymbol{U}^{(r)} - \eta\Delta\boldsymbol{U}$
- $\boldsymbol{b}^{x(r+1)} = \boldsymbol{b}^{x(r)} - \eta\Delta\boldsymbol{b}^{x}$
- $\boldsymbol{b}^{h(r+1)} = \boldsymbol{b}^{h(r)} - \eta\Delta\boldsymbol{b}^{h}$
- $\boldsymbol{b}^{y(r+1)} = \boldsymbol{b}^{y(r)} - \eta\Delta\boldsymbol{b}^{y}$

End

We also tested the B-BP double-regression algorithm on the *noninvertible* function $f(x) = \sin x$ for $x \in [-\pi, \pi]$. The forward mapping $f(x) = \sin x$ is a well-defined point function. The backward mapping $y = \sin^{-1}(f(x))$ is not. It defines instead a set-based pullback or preimage $f^{-1}(y) = f^{-1}(\{y\}) = \{x \in \mathbb{R} : f(x) = y\} \subset \mathbb{R}$. The B-BP-trained neural network tends to map each output point y to the centroid of its preimage $f^{-1}(y)$ on the backward pass because centroids minimize squared error and because backward-regression training uses squared error as its performance measure. Fig. 7 shows that forward regression learns the target function $\sin x$ while backward regression approximates the centroids $-(\pi/2)$ and $(\pi/2)$ of the two preimage sets.

V. CONCLUSION

Unidirectional BP learning extends to B-BP learning if the algorithm uses the appropriate joint error function for both forward and backward passes. This bidirectional extension applies to classification networks as well as to regression networks and to their combinations. Most classification networks can easily acquire a backward-inference capability if they include a backward-regression step in their training. So most networks simply ignore this inverse property of their weight structure.

Theorem 1 shows that a bidirectional multilayer threshold network can exactly represent a permutation mapping if the hidden layer contains an exponential number of hidden threshold neurons. An open question is whether these bidirectional networks can represent an arbitrary invertible mapping with far fewer hidden neurons. A simpler question holds for the weaker case of uniform approximation of invertible mappings.

Another open question deals with noise: to what extent does carefully injected noise speed B-BP convergence and accuracy? There are two bases for this question. The first is that the likelihood structure of BP implies that BP is itself a special case of the expectation–maximization algorithm [19]. The second basis is that appropriate noise can boost the EM family of hill-climbing algorithms on average because such noise makes signals more probable on average [21], [23].

References

[1] 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.

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

[3] C. M. Bishop, *Pattern Recognition and Machine Learning*. New York, NY, USA: Springer, 2006.

[4] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, May 2015.

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

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

[7] B. Kosko, *Neural Networks and Fuzzy Systems: A Dynamical Systems Approach to Machine Intelligence*. Englewood Cliffs, NJ, USA: Prentice-Hall, 1991.

[8] S. Y. Kung, *Kernel Methods and Machine Learning*. Cambridge, U.K.: Cambridge Univ. Press, 2014.

[9] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Math. Control Signals Syst.*, vol. 2, no. 4, pp. 303–314, 1989.

[10] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Netw.*, vol. 2, no. 5, pp. 359–366, 1989.

[11] B. Kosko, "Fuzzy systems as universal approximators," *IEEE Trans. Comput.*, vol. 43, no. 11, pp. 1329–1333, Nov. 1994.

[12] F. Watkins, "The representation problem for additive fuzzy systems," in *Proc. Int. Conf. Fuzzy Syst. (IEEE FUZZ)*, 1995, pp. 117–122.

[13] B. Kosko, "Generalized mixture representations and combinations for additive fuzzy systems," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, 2017, pp. 3761–3768.

[14] B. Kosko, "Additive fuzzy systems: From generalized mixtures to rule continua," *Int. J. Intell. Syst.*, vol. 33, no. 8, pp. 1573–1623, 2017.

[15] J.-N. Hwang, J. J. Choi, S. Oh, and R. J. Marks, "Query-based learning applied to partially trained multilayer perceptrons," *IEEE Trans. Neural Netw.*, vol. 2, no. 1, pp. 131–136, Jan. 1991.

[16] E. W. Saad, J. J. Choi, J. L. Vian, and D. C. Wunsch, "Query-based learning for aerospace applications," *IEEE Trans. Neural Netw.*, vol. 14, no. 6, pp. 1437–1448, Nov. 2003.

[17] E. W. Saad and D. C. Wunsch, "Neural network explanation using inversion," *Neural Netw.*, vol. 20, no. 1, pp. 78–93, 2007.

[18] Y. Yang, Y. Wang, and X. Yuan, "Bidirectional extreme learning machine for regression problem and its learning effectiveness," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 23, no. 9, pp. 1498–1505, Sep. 2012.

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

[20] S. Kullback and R. A. Leibler, "On information and sufficiency," *Ann. Math. Stat.*, vol. 22, no. 1, pp. 79–86, 1951.

[21] O. Osoba and B. Kosko, "The noisy expectation-maximization algorithm for multiplicative noise injection," *Fluctuation Noise Lett.*, vol. 15, no. 4, 2016, Art. no. 1650007.

[22] R. V. Hogg, J. McKean, and A. T. Craig, *Introduction to Mathematical Statistics*. Boston, MA, USA: Pearson, 2013.

[23] O. Osoba, S. Mitaim, and B. Kosko, "The noisy expectation–maximization algorithm," *Fluctuation Noise Lett.*, vol. 12, no. 3, 2013, Art. no. 1350012.

Olaoluwa (Oliver) Adigun received the Bachelor of Science degree in electronic and electrical engineering from Obafemi Awolowo University, Ife, Nigeria. He is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, Signal and Image Processing Institute, University of Southern California, Los Angeles, CA, USA.

He has been an Intern with Google AI, Mountain View, CA, USA, and with the Machine Learning Group, Amazon, Seattle, WA, USA.

Mr. Adigun shared the Best Paper Award for his research on noise-boosted recurrent backpropagation at the 2017 International Joint Conference on Neural Networks.

Bart Kosko (M'85–SM'07–F'10) received the degrees in philosophy, economics, applied mathematics, electrical engineering, and law.

He is a Professor of Department of Electrical and Computer Engineering and Law and the Past Director of Signal and Image Processing Institute with the University of Southern California, Los Angeles, CA, USA, and a Licensed Attorney. He has published the textbooks entitled *Neural Networks and Fuzzy Systems* and *Fuzzy Engineering*, the trade books entitled *Fuzzy Thinking*, *Heaven in a Chip*, and *Noise*, the edited volume *Neural Networks and Signal Processing*, the co-edited volume *Intelligent Signal Processing*, the novel *Nanotime*, and the upcoming novel *Cool Earth*.

Dr. Kosko was a co-recipient of the Best Paper Award at the 2017 International Joint Conference on Neural Networks.