Bidirectional Representation and Backpropagation Learning

Olaoluwa Adigun and Bart Kosko
Department of Electrical Engineering
Signal and Image Processing Institute
University of Southern California

Abstract— *The backpropagation learning algorithm extends to bidirectional training of multilayer neural networks. The bidirectional operation gives a form of backward chaining or backward inference from a network output. We first prove that a fixed three-layer network of threshold neurons can exactly represent any finite permutation function and its inverse. The forward pass gives the function value. The backward pass through the same network gives the inverse value. We then derive and test a bidirectional version of the backpropagation algorithm that can learn bidirectional mappings or their approximations.*

Keywords: bidirectional associative memory, backpropagation learning, function representation, backward inference

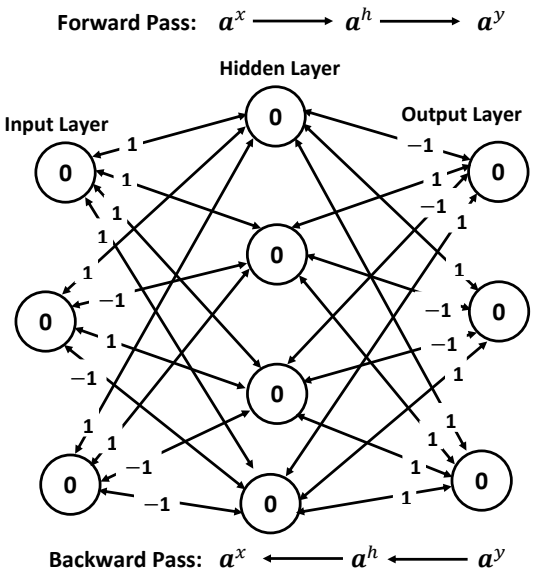

Fig. 1: *Bidirectional Representation of a Permutation Function. This 3-layer bidirectional threshold network exactly represents the invertible 3-bit bipolar permutation function f in Table 1. The forward pass feeds the input vector x to the input layer and passes it through the weighted links and the hidden layer of threshold neurons (each with zero threshold) to the output layer. The backward pass sends the output bit vector y back through the same weighted links and threshold neurons. The network computes $y = f(x)$ on the forward pass and the inverse value $f^{-1}(y)$ on the backward pass.*

1. Bidirectional Backpropagation

We show that bidirectional backpropagation (B-BP) training endows a multilayered neural network $N\colon \mathbb{R}^n \to \mathbb{R}^p$ with a form of backward inference. The forward pass gives the usual predicted neural output $N(x)$ given a vector input x. The output vector value $y = N(x)$ in effect answers the *what-if* question that x poses: What would we observe if x occurred? What would be the effect? Then the backward pass answers the *why* question that y poses: Why did y occur? What type of input would cause y? Feedback convergence to a resonating bidirectional fixed-point attractor [1], [2] gives a long-term or equilibrium answer to both the what-if and why questions.

This bidirectional approach to neural learning applies to big data because the BP algorithm [3], [4], [5] scales linearly with training data. BP has time complexity $O(n)$ for n training samples because the forward pass is $O(1)$ while the backward pass is $O(n)$. So the new B-BP algorithm still has only $O(n)$ complexity. This linear scaling does not hold in general for most machine-learning algorithms. An example is the quadratic complexity $O(n^2)$ of support-vector kernel methods [6].

We present the bidirectional results in two parts. The first part proves that there exist fixed-weight multilayer threshold networks that can exactly represent some invertible functions. Theorem 1 shows that this holds for all finite bipolar (or binary) permutation functions. Figure 1 shows such a bidirectional 3-layer network of zero-threshold neurons. It exactly represents the 3-bit permutation function f in Table 1 where $\{-,-,+\}$ denotes $\{-1,-1,1\}$. So f is a self-bijection that rearranges the 8 vectors in the bipolar hypercube $\{-1,1\}^3$. The forward pass converts the input bipolar vector $(1,1,1)$ into the output bipolar vector $(-1,-1,1)$. The backward pass converts $(-1,-1,1)$ into $(1,1,1)$ over the *same* fixed synaptic connection weights. These same weights and neurons convert the other 7 input vectors in the first column of Table 1 to the corresponding 7 output vectors in the second column and conversely.

Theorem 1 requires 2^n hidden neurons to represent a permutation function on the bipolar hypercube $\{-1,1\}^n$. Using so many hidden neurons is neither practical nor necessary. The representation in Figure 1 uses only 4 hidden neurons. It is just one example of a representation that uses fewer than 8 hidden neurons. We seek instead an efficient learning

algorithm that can learn bidirectional representations (or at least approximations) from sample data.

The second part extends the BP algorithm to just this bidirectional case. This takes some care because training the same weights in one direction tends to overwrite or undo the BP training in the other direction. The B-BP algorithm solves this problem by minimizing a joint error. It found representations of the permutation in Table 1 that needed only 3 hidden neurons.

The learning approximation also improves by adding more hidden neurons. Figure 2 shows the effect of training with 100 hidden neurons. Figure 3 shows how the B-BP training error falls off as the number of hidden neurons grows when learning the 5-bit permutation in Table 2.

2. Bidirectional Function Representation of Bipolar Permutations

This section proves that there exists multilayered neural networks that can exactly bidirectionally represent some invertible functions. We first define the network variables. The proof uses threshold neurons while the B-BP algorithm uses soft-threshold logistic sigmoids for hidden neurons and uses identity activations for input and output neurons.

A bidirectional neural network is a multilayer network $N\colon X \to Y$ that maps the input space X to the output space Y *and conversely* through the same set of weights. The backward pass uses the transpose matrices of the weight matrices that the forward pass uses. Such a network is a bidirectional associative memory or BAM [1], [2].

The forward pass sends input vector x through weight matrix $\mathbf{W}$ from the input layer to the hidden layer and then on through matrix $\mathbf{U}$ to the output layer. The backward pass sends the output y from the output layer back through the hidden layer to the input layer. Let I, J, and K denote the respective number 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.

Table 1: 3-Bit Bipolar Permutation Function f.

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

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

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

where weight w_{ji} connects the i^{th} input neuron to the j^{th} hidden neuron, a_i^x is the activation of the i^{th} input neuron, and b_j^h is the bias term of the j^{th} hidden neuron. The activation a_j^h of the j^{th} hidden neuron is a bipolar threshold:

$$a_j^h(o_j^h) = \begin{cases} -1 & \textit{if } o_j^h \leq 0 \\ 1 & \textit{if } o_j^h > 0\ . \end{cases} \tag{2}$$

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 (13) and to fire with a slightly different condition.

The input o_k^y to the k^{th} output neuron from the hidden layer is also affine:

$$o_k^y = \sum_{j=1}^{J} u_{kj} a_j^h + b_k^y \tag{3}$$

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

$$a_k^y(o_k^y) = \begin{cases} -1 & \text{if } o_k^y \leq 0 \\ 1 & \text{if } o_k^y > 0\ . \end{cases} \tag{4}$$

The forward pass of an input bipolar vector $\mathbf{x}$ from Table 1 through the network in Figure 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 j^{th} hidden neuron is

$$o_j^{hb} = \sum_{k=1}^{K} u_{kj} y^k + b_j^h \tag{5}$$

where y^k is the output of the k^{th} output neuron. The backward-pass activation of the j^{th} hidden neuron a_j^{hb} is

$$a_j^{hb}(o_j^{hb}) = \begin{cases} -1 & \textit{if } o_j^{hb} \leq 0 \\ 1 & \textit{if } o_j^{hb} > 0\ . \end{cases} \tag{6}$$

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

$$o_i^{xb} = \sum_{j=1}^{J} w_{ji} a_j^{hb} + b_i^x \tag{7}$$

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

$$a_i^x(o_i^{xb}) = \begin{cases} -1 & \textit{if } o_i^{xb} \leq 0 \\ 1 & \textit{if } o_i^{xb} > 0\ . \end{cases} \tag{8}$$

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 $f\colon \{-1,1\}^n \to \{-1,1\}^n$ is a permutation. Then there exists a 3-layer bidirectional neural network $N\colon \{-1,1\}^n \to \{-1,1\}^n$ that exactly represents f in the sense that $N(x) = f(x)$ and $N^{-1}(x) = f^{-1}(x)$ for all x.*

Proof: The proof strategy picks weight matrices $\mathbf{W}$ and $\mathbf{U}$ so that only one hidden neuron fires on both the forward and the backward pass. So we structure the network such that any input vector $\mathbf{x}$ fires only one hidden neuron on the forward pass and such that 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 $\{-1,1\}^n$ into itself. The bipolar hypercube contains the 2^n input bipolar column vectors $\mathbf{x_1}, \mathbf{x_2}, \dots, \mathbf{x_{2^n}}$. It likewise contains the 2^n output bipolar vectors $\mathbf{y_1}, \mathbf{y_2}, \dots, \mathbf{y_{2^n}}$. The network will use 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 output layer. Define $\mathbf{W}$ so that each row lists all 2^n bipolar input vectors and define $\mathbf{U}$ so that each column lists all 2^n transposed bipolar output vectors:

$$\mathbf{W} = \begin{bmatrix} \vdots & \vdots & \vdots & \vdots & \vdots \\ \mathbf{x_1} & \mathbf{x_2} & \vdots & \vdots & \mathbf{x_{2^n}} \\ \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

$$\mathbf{U} = \begin{bmatrix} \cdots & \mathbf{y_1}^T & \cdots \\ \cdots & \mathbf{y_2}^T & \cdots \\ \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots \\ \cdots & \mathbf{y_{2^n}}^T & \cdots \end{bmatrix}$$

We now show that this arrangement fires 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 every neuron has zero bias.

Pick a bipolar input vector $\mathbf{x}_m$ for the forward pass. Then the input activation vector $\mathbf{a}^x(\mathbf{x}_m) = (a_1^x(x_m^1), \dots, a_n^x(x_m^n))$ equals the input bipolar vector $\mathbf{x}_m$ because the input activations (8) 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 (1). It has the matrix-vector form

$$\mathbf{o}^h = \mathbf{W}^T \mathbf{a}^x \tag{9}$$
$$= \mathbf{W}^T \mathbf{x}_m \tag{10}$$
$$= (o_1^h,\ o_2^h, ...,\ o_n^h, ...,\ o_{2^n}^h)^T \tag{11}$$
$$= (\mathbf{x}_1^T \mathbf{x}_m,\ \mathbf{x}_2^T \mathbf{x}_m, \dots,\ \mathbf{x}_j^T \mathbf{x}_m, ...,\ \mathbf{x}_{2^n}^T \mathbf{x}_m)^T \tag{12}$$

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

The input o_j^h to the j^{th} neuron of the hidden layer obeys $o_j^h = n$ when $j = m$ and $o_j^h < n$ 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 $\mathbf{x}_j^T \mathbf{x}_m$ is maximum when both vectors have the same direction. This occurs when $j = m$. The inner product is otherwise less than n.

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:

$$a_j^h(o_j^h) = \begin{cases} 1 & \text{if } o_j^h \geq n, \\ 0 & \text{if } o_j^h < n\ . \end{cases} \tag{13}$$

Then the hidden layer activation $\mathbf{a}^h$ is the *unit* bit vector $(0,\ 0,\ ...,\ 1,\ ...,\ 0)^T$ 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 $\{-1,1\}$ are distinct and so exactly one of these 2^n vectors achieves the maximum inner-product value $n = \mathbf{x}_m^T \mathbf{x}_m$.

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

$$\mathbf{o}^y = \mathbf{U}^T\ \mathbf{a}^h \tag{14}$$
$$= \sum_{j=1}^{J} \mathbf{y}_j\ a_j^h \tag{15}$$
$$= \mathbf{y}_m \tag{16}$$

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

$$\mathbf{a}^y(o_j^y) = \begin{cases} 1 & \text{if } o_j^y \geq 0 \\ -1 & \text{if } o_j^y < 0\ . \end{cases} \tag{17}$$

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

$$\mathbf{a}^y = \mathbf{y}_m\ . \tag{18}$$

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

Consider next the backward pass over N.

The backward pass propagates the output vector $\mathbf{y}_m$ from the output layer to the input layer through the hidden layer. The hidden layer input $\mathbf{o}^h$ has the form (5) and so

$$\mathbf{o}^h = \mathbf{U}\ \mathbf{y}_m \tag{19}$$

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

The input o_j^h of the j^{th} neuron in the hidden layer o_j^h equals the inner product of $\mathbf{y}_j$ and $\mathbf{y}_m$. So $o_j^h = n$ when $j = m$ and $o_j^h < n$ 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^h is maximum when vectors $\mathbf{y}_m$ and $\mathbf{y}_j$ lie in the same direction. The activation $\mathbf{a}^h$ for the hidden layer has the same components as (13). So the hidden-layer activation $\mathbf{a}^h$ again equals the unit bit vecgtor $(0,\ 0,\ ...,\ 1,\ ...,\ 0)^T$ where $a_j^h = 1$ when $j = m$ and $a_j^h = 0$ when $j \neq m$.

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

$$\mathbf{o}^x = \mathbf{W}\ \mathbf{a}^h \tag{20}$$

$$= \sum_{j=1}^{J} \mathbf{x}_j\ \mathbf{a}^h \tag{21}$$

$$= \mathbf{x}_m\ . \tag{22}$$

The i^{th} input neuron has a threshold activation that is the same as

$$\mathbf{a}^x(o_i^x) = \begin{cases} 1 & if\ o_i^x \geq 0 \\ -1 & if\ o_i^x < 0 \end{cases} \tag{23}$$

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

$$\mathbf{a}^x = \mathbf{o}^x \tag{24}$$

$$= \mathbf{x}_m\ . \tag{25}$$

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

.

3. Bidirectional BP Learning

We now develop the new bidirectional BP algorithm for learning bidirectional function representations or approximations. Bidirectional BP training minimizes both the error function for the forward pass and the backward pass. The forward-pass error E_f is the error at the output layer. The backward-pass error E_b is the error at the input layer. Bidirectional BP training combines these two errors.

The forward pass sends the input vector $\mathbf{x}$ through the hidden layer to the ouput layer. We use only one hidden layer. There is no loss of generality in using any finite number of them. The hidden-layer input values o_j^h are the same as (1). The j^{th} hidden activation a_j^h is the *bipolar* logistic that shifts and scales the ordinary logistic:

$$a_j^h(o_j^h) = \frac{2}{1 + e^{-2o_j^h}} - 1 \tag{26}$$

and (3) gives the input o_k^y to the k^{th} output neuron. The hidden activations can also be logistic or any other sigmoidal function. The activation for an output neuron is the identity function:

$$a_k^y = o_k^y \tag{27}$$

where a_k^y is the activation of k^{th} output neuron. The error function for the forward pass was the squared error E_f

$$E_f = \frac{1}{2}\sum_{k=1}^{K}(y_k - a_k^y)^2\ . \tag{28}$$

where y_k is the value of k^{th} 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 $\mathbf{y}$ from the output layer to the input layer through the hidden layer. The input to j^{th} hidden neuron o_j^h is the same as (5). The activation a_j^h for j^{th} hidden neuron is:

$$a_j^h = \frac{2}{1 + e^{-2o_j^h}} - 1\ . \tag{29}$$

The input o_i^x for the i^{th} input neuron is the same as (7). The activation at the input layer is the identity function:

$$a_i^x = o_i^x\ . \tag{30}$$

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

The backward-pass error E_b is

$$E_b = \frac{1}{2}\sum_{i=1}^{I}(x_i - a_i^x)^2\ . \tag{31}$$

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

$$\frac{\partial a_j^h}{\partial o_j^h} = \frac{\partial}{\partial o_j^h}\Big(\frac{2}{1 + e^{-2o_j^h}} - 1\Big) \tag{32}$$

$$= \frac{4e^{-2o_j^h}}{(1 + e^{-2o_j^h})^2} \tag{33}$$

$$= \frac{2}{1 + e^{-2o_j^h}}\Big[2 - \frac{2}{1 + e^{-2o_j^h}}\Big] \tag{34}$$

$$= (a_j^h + 1)(1 - a_j^h)\ . \tag{35}$$

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 backward pass. Then the partial derivative of E_f with respect to weight u_{kj} is

$$\frac{\partial E_f}{\partial u_{kj}} = \frac{1}{2}\frac{\partial}{\partial u_{kj}}\sum_{k=1}^{K}(y_k - a_k^y)^2 \tag{36}$$

$$= \frac{\partial E_f}{\partial a_k^y}\ \frac{\partial a_k^y}{\partial o_k^y}\ \frac{\partial o_k^y}{\partial u^{kj}} \tag{37}$$

$$= (y_k - a_k^y) \times 1 \times a_k^y\ . \tag{38}$$

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

$$\frac{\partial E_f}{\partial w_{ji}} = \frac{1}{2}\frac{\partial}{\partial w_{ji}}\sum_{k=1}^{K}(y_k - a_k^y)^2 \tag{39}$$

$$= \Big(\sum_{k=1}^{K}\frac{\partial E_f}{\partial a_k^y}\ \frac{\partial a_k^y}{\partial o_k^y}\ \frac{\partial o_k^y}{\partial a_j^h}\Big)\ \frac{\partial a_j^h}{\partial o_j^h}\ \frac{\partial o_j^h}{\partial w_{ji}} \tag{40}$$

$$= \sum_{k=1}^{K}(y_k - a_k^y)u_{kj} \times a_j^{h'} \times x_i\ . \tag{41}$$

The partial derivative of E_f with respect to the bias b_k^y of the k^{th} output neuron is

$$\frac{\partial E_f}{\partial b_k^y} = \frac{1}{2}\frac{\partial}{\partial b_k^y}\sum_{k=1}^{K}(y_k - a_k^y)^2 \tag{42}$$

$$= \frac{\partial E_f}{\partial a_k^y}\ \frac{\partial a_k^y}{\partial o_k^y}\ \frac{\partial o_k^y}{\partial b_k^y} \tag{43}$$

$$= (y_k - a_k^y) \times 1 \times 1\ . \tag{44}$$

The partial derivative of E_f with respect to the bias b_j^h of the j^{th} hidden neuron is

$$\frac{\partial E_f}{\partial b_j^h} = \frac{1}{2}\frac{\partial}{\partial b_j^h}\sum_{k=1}^{K}(y_k - a_k^y)^2 \tag{45}$$

$$= \Big(\sum_{k=1}^{K}\frac{\partial E_f}{\partial a_k^y}\ \frac{\partial a_k^y}{\partial o_k^y}\ \frac{\partial o_k^y}{\partial a_j^h}\Big)\ \frac{\partial a_j^h}{\partial o_j^h}\ \frac{\partial o_j^h}{\partial b_j^h} \tag{46}$$

$$= \sum_{k=1}^{K}(y_k - a_k^y)u_{kj} \times a_j^{h'} \times 1\ . \tag{47}$$

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

$$\frac{\partial E_b}{\partial w_{ji}} = \frac{1}{2}\frac{\partial}{\partial w_{ji}}\sum_{k=1}^{K}(x_i - a_i^x)^2 \tag{48}$$

$$= \frac{\partial E_b}{\partial a_i^x}\ \frac{\partial a_i^x}{\partial o_i^x}\ \frac{\partial o_i^x}{\partial w^{ji}} \tag{49}$$

$$= (x_i - a_i^x) \times 1 \times a_i^x\ . \tag{50}$$

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

$$\frac{\partial E_b}{\partial u_{kj}} = \frac{1}{2}\frac{\partial}{\partial u_{kj}}\sum_{i=1}^{I}(x_i - a_i^x)^2 \tag{51}$$

$$= \Big(\sum_{i=1}^{I}\frac{\partial E_b}{\partial a_i^x}\ \frac{\partial a_i^x}{\partial o_i^x}\ \frac{\partial o_i^x}{\partial a_j^{hb}}\Big)\ \frac{\partial a_j^{hb}}{\partial o_j^{hb}}\ \frac{\partial o_j^{hb}}{\partial u_{kj}} \tag{52}$$

$$= \sum_{i=1}^{I}(x_i - a_i^x)w_{ji} \times a_j^{hb'} \times y_k\ . \tag{53}$$

The partial derivative of E_b with respect to the bias b_i^x of the i^{th} input neuron is:

$$\frac{\partial E_b}{\partial b_i^x} = \frac{1}{2}\frac{\partial}{\partial b_i^x}\sum_{i=1}^{I}(x_i - a_i^x)^2 \tag{54}$$

$$= \frac{\partial E_b}{\partial a_i^x}\ \frac{\partial a_i^x}{\partial o_i^x}\ \frac{\partial o_i^x}{\partial b_i^x} \tag{55}$$

$$= (x_i - a_i^x) \times 1 \times 1\ . \tag{56}$$

The partial derivative of E_b with respect to the bias b_j^h of the j^{th} hidden neuron is

$$\frac{\partial E_b}{\partial b_j^h} = \frac{1}{2}\frac{\partial}{\partial b_j^h}\sum_{i=1}^{I}(x_i - a_i^x)^2 \tag{57}$$

$$= \Big(\sum_{i=1}^{I}\frac{\partial E_b}{\partial a_i^x}\ \frac{\partial a_i^x}{\partial o_i^x}\ \frac{\partial o_i^x}{\partial a_j^{hb}}\Big)\ \frac{\partial a_j^{hb}}{\partial o_j^{hb}}\ \frac{\partial o_j^{hb}}{\partial b_j^h} \tag{58}$$

$$= \sum_{i=1}^{I}(x_i - a_i^x)w_{ji} \times a_j^{hb'} \times 1\ . \tag{59}$$

Bidirectional BP training minimizes the *joint* error E of the forward and backward passes. The joint error E sums the forward error E_f and backward error E_b:

$$E = E_f + E_b\ . \tag{60}$$

Then the partial derivative of E with respect to u_{kj} is

$$\frac{\partial E}{\partial u_{kj}} = \frac{\partial E_f}{\partial u_{kj}} + \frac{\partial E_b}{\partial u_{kj}} \tag{61}$$

$$= (y_k - a_k^y)a_k^y + \sum_{i=1}^{I}(x_i - a_i^x)w_{ji}a_j^{hb'}y_k \tag{62}$$

from (36) and (51) . The partial derivative of the joint error E with respect to the weight w_{ji} is

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E_f}{\partial w_{ji}} + \frac{\partial E_b}{\partial w_{ji}} \tag{63}$$

$$= \sum_{k=1}^{K}(y_k - a_k^y)u_{kj}a_j^{h'}x_i \tag{64}$$

$$+ (x_i - a_i^x)a_i^x \tag{65}$$

from (39) and (48) .

The partial derivative of E with respect to b_j^h gives

$$\frac{\partial E}{\partial b_j^h} = \frac{\partial E_f}{\partial b_j^h} + \frac{\partial E_b}{\partial b_j^h} \tag{66}$$

$$= \sum_{k=1}^{K}(y_k - a_k^y)u_{kj} \times a_j^{h'} \tag{67}$$

$$+ \sum_{i=1}^{I}(x_i - a_i^x)w_{ji} \times a_j^{hb'}\ . \tag{68}$$

from (45) and (57) .

The error for the input neuron bias is E_b only because $\mathbf{x} = \mathbf{o}^x$ for the forward pass. The error for the output neuron bias is E_f only for output neuron bias because $\mathbf{y} = \mathbf{o}^y$ for the backward pass. Then

$$\frac{\partial E}{\partial b_i^x} = \frac{\partial E_b}{\partial b_i^x} = x_i - a_i^x \tag{69}$$

$$\frac{\partial E}{\partial b_k^y} = \frac{\partial E_f}{\partial b_k^y} = y_k - a_k^y \ . \tag{70}$$

Then B-BP training updates the parameters as

$$u_{kj}^{(n+1)} = u_{kj}^{(n)} - \eta \frac{\partial E}{\partial u_{kj}} \tag{71}$$

$$w_{ji}^{(n+1)} = w_{ji}^{(n)} - \eta \frac{\partial E}{\partial w_{ji}} \tag{72}$$

$$b_i^{x(n+1)} = b_i^{x(n)} - \eta \frac{\partial E}{\partial b_i^x} \tag{73}$$

$$b_j^{h(n+1)} = b_j^{h(n)} - \eta \frac{\partial E}{\partial b_j^h} \tag{74}$$

$$b_k^{y(n+1)} = b_k^{y(n)} - \eta \frac{\partial E}{\partial b_k^y} \tag{75}$$

where η is the learning rate. The partial derivatives are from (61)−(70). Algorithm 1 summarizes the B-BP algorithm.

4. Simulation Results

We tested the bidirectional BP algorithm on a 5-bit permutation functions in 3-layer networks with different numbers of hidden neurons. The B-BP algorithm produced either an exact representation or approximation. We report results for learning a permutation function from the 5-bit bipolar vector space $\{-1, 1\}^n$. The hidden neurons used bipolar logistic activations. The input and output neurons used identity activations. Table 2 displays the the permutation test function that mapped $\{-1, 1\}^5$ to itself. We compared the forward and backward forms unidirectional BP with bidirectional BP. We also tested to see whether adding more hidden neurons improved network approximation accuracy.

The simulations used 18,000 samples for network training and 2,000 separate samples for testing. Forward-pass of (standard) BP used E_f as its error while backward-pass BP used E_b as its error. Bidirectional BP combined both E_f and E_b for its joint error. We computed testing error for forward pass and backward pass. Each plotted error value averaged 20 runs.

Figure 2 shows the results of running the three types of BP learning on a 3-layer network with 100 hidden neurons. The training error falls along both directions as the training progresses. This in not the case for the unidirectional cases of forward BP and backward BP training. Forward training and backward training perform well only for function approximation in their preferred direction and not in the opposite direction.

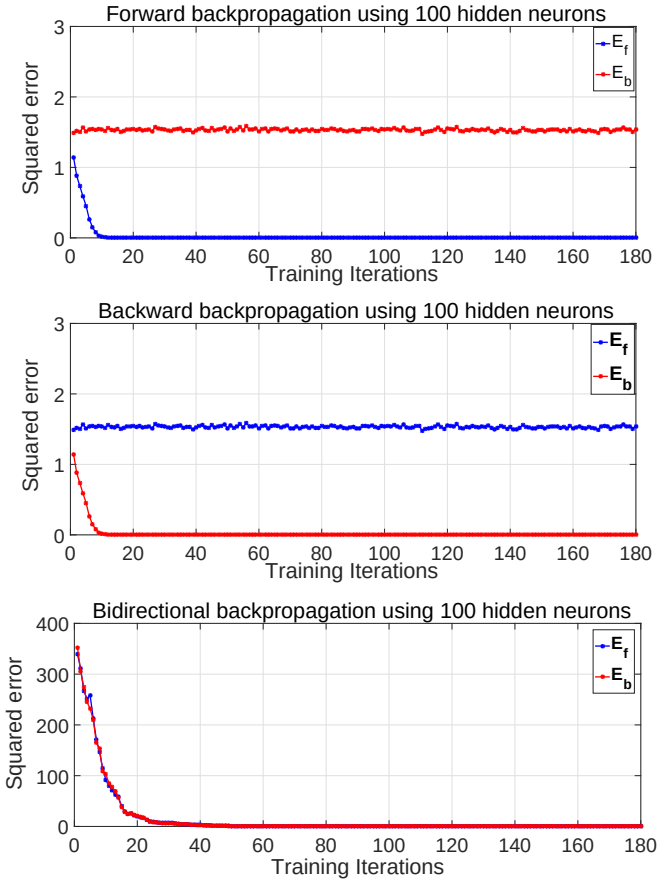

Fig. 2: *Training-set squared error using 100 hidden neurons with forward BP training, backward BP training, and bidirectional BP training. Forward BP tuned the network with respect to E_f only. Backward BP training tuned it respect to E_b only. Bidirectional BP training combined E_f and E_b to update the network parameters.*

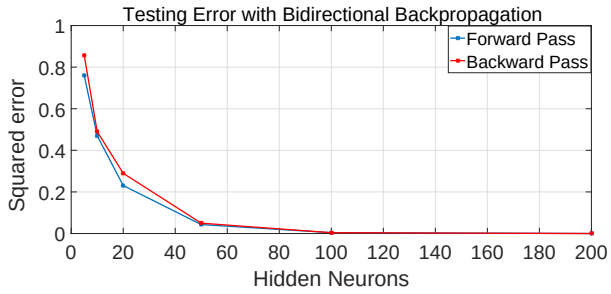

Fig. 3: *B-BP training error for the 3-bit permutation in Table 2 with different numbers of hidden neurons. The two curves describe the training error for the forward and backward passes through the 3-layer network. Each test used 2000 samples. The number of hidden neurons varied from 5, 10, 20, 50, 100, to 200.*

Table 3 shows the forward-pass test errors for learning 3-layer 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

Table 2: 5-Bit Bipolar Permutation Function.

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

Table 3: Forward-Pass Testing Error E_f

	Backpropagation errors		
Hidden Neurons	Forward	Backward	Bidirectional
5	0.6032	1.2129	0.7600
10	0.1732	1.4259	0.4700
20	0.0068	1.4031	0.2307
50	1.5×10^{-4}	1.5811	0.0430
100	2.0×10^{-6}	1.4681	0.0043
200	5.0×10^{-8}	1.6061	4.0×10^{-6}

Table 4: Backward-Pass Testing Error E_b

	Backpropagation errors		
Hidden Neurons	Forward	Backward	Bidirectional
5	1.2070	0.6016	0.8574
10	1.4085	0.1584	0.4900
20	1.2384	0. 0023	0.2895
50	1.3157	1.2×10^{-4}	0.0498
100	1.4681	3.6×10^{-6}	0.0039
200	1.7837	8.7×10^{-8}	9.0×10^{-6}

bidirectional BP. The forward-pass error for forward BP fell significantly as the number of hidden neurons grew. The forward-pass error of backward BP decreased slightly as the number of hidden neurons grew and gave the worst performance. Bidirectional BP performed well on the test set. Its forward-pass error also fell significantly as the number of hidden neurons grew. Table 4 shows similar error-versus-hidden-neuron results for the backward-pass error

The two tables jointly show that the unidirectional forms of BP performed well only in one direction while the B-BP algorithm performed well in both directions. Hence we propose using only the B-BP algorithm for learning bidirectional function representations or approximations.

Data: $\boldsymbol{T}$ input vectors $\{\boldsymbol{x}_1,\ldots,\ \boldsymbol{x}_T\}$, $\boldsymbol{T}$ target vectors $\{\boldsymbol{y}_1,\ldots,\ \boldsymbol{y}_T\}$ such that $f(\boldsymbol{x}_i) = \boldsymbol{y}_i$. Number of hidden neurons $\boldsymbol{J}$. Batch size $\boldsymbol{S}$ and number of epochs $\boldsymbol{R}$. Choose the learning rate $\boldsymbol{\eta}$.

Result: Bidirectional network for function f.

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

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

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

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

- Pick input vector $\boldsymbol{x}$ and its corresponding target vector $\boldsymbol{y}$.
- Compute hidden activation $\boldsymbol{a}^h$ and output activation $\boldsymbol{a}^y$ for forward pass.
- Compute hidden activation $\boldsymbol{a}^h$ and output activation $\boldsymbol{a}^y$ for backward pass.
- Compute the derivatives with respect to $\boldsymbol{W}$ and $\boldsymbol{U}$: $\boldsymbol{\nabla_W}E$ and $\boldsymbol{\nabla}_U E$.
- Compute the derivatives with respect to the bias weights: $\boldsymbol{\nabla_{b^x}}E$, $\boldsymbol{\nabla_{b^y}}E$, and $\boldsymbol{\nabla_{b^y}}E$
- Compute change in weights: $\Delta\boldsymbol{W}=\Delta\boldsymbol{W}+\boldsymbol{\nabla_W}E$ and $\Delta\boldsymbol{W}=\Delta\boldsymbol{W}+\boldsymbol{\nabla}_U E$
- Compute change in bias weights: $(\Delta\boldsymbol{b^x}=\Delta\boldsymbol{b^x}+\boldsymbol{\nabla_{b^x}}E)$, $(\Delta\boldsymbol{b^y}=\Delta\boldsymbol{b^y}+\boldsymbol{\nabla_{b^x}}E)$ and $(\Delta\boldsymbol{b^z}=\Delta\boldsymbol{b^z}+\boldsymbol{\nabla_{b^z}}E)$.

End

Update: $\boldsymbol{W^{(r+1)}=\ W^{(r)}-\frac{\eta}{L}\Delta U}$

$\boldsymbol{U^{(r+1)}=\ U^{(r)}-\frac{\eta}{L}\Delta U}$

$\boldsymbol{b^{x(r+1)}=\ b^{x(r)}-\frac{\eta}{L}\Delta b^x}$

$\boldsymbol{b^{y(r+1)}=\ b^{y(r)}-\frac{\eta}{L}\Delta b^y}$

$\boldsymbol{b^{z(r+1)}=\ b^{z(r)}-\frac{\eta}{L}\Delta b^z}$

End

Algorithm 1: *The Bidirectional BP Algorithm*

5. Conclusions

We have shown that a bidirectional multilayer network can exactly represent bipolar or binary permutation mappings if the network uses enough threshold or sigmoidal neurons. The proof requires an exponential number of hidden neurons for exact representations but much simpler representation exist in general. The new B-BP algorithm allows bidirectional learning of sampled functions. It can often find efficient bidirectional representations or approximations with a smaller set of hidden neurons.

References

[1] B. Kosko, "Bidirectional associative memories," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 18, no. 1, pp. 49–60, 1988.

[2] B. Kosko, *Neural Networks and Fuzzy Systems: A Dynamical Systems Approach to Machine Intelligence*. Prentice Hall, 1991.

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

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

[5] M. Jordan and T. Mitchell, "Machine learning: trends, perspectives, and prospects," *Science*, vol. 349, pp. 255–260, 2015.

[6] S. Y. Kung, *Kernel methods and machine learning*. Cambridge University Press, 2014.