Loading Discriminative Feature Representations in Hidden Layer

Daw-Ran Liou*, Yang-En Chen, Cheng-Yuan Liou
Dept. of Computer Science and Information Engineering
National Taiwan University
Taipei, Taiwan
*Correspondent: dawran6@gmail.com

***Abstract*—This work explores the neural features that are trained by decreasing a discriminative energy. It directly resolves the unfaithful representation problem and the ambiguous internal representation problem in various back-propagation training algorithms for MLP. It also indirectly overcomes the premature saturation problem.**

***Keywords*—Multilayer perceptron; deep learning; Boltzmann machine; ambiguous internal representation; unfaithful representation; classification; image restoration**

I. Introduction

The optimal spaced codes can hardly be accomplished by any learning algorithms for reduced Boltzmann machine [19]. This limits its applications in classification of noisy patterns and resolving blurred images. This work studies a discriminate energy to separate the internal representations of the hidden layer where the Euclidean distance between every two representations is enlarged as possible. Each representation is isolated as far as possible from all others in the layer hypercube space. When the representations of certain patterns can be isolated within a Euclidean radius, one can discriminate these patterns from all other patterns using a single neuron in the succeeding hidden layer. Such representations will exhaust the space uniformly and have tolerance for noisy patterns. The layered network is developed, layer after layer as those for deep networks, as an adjustable kernel to separate multiple classes as much as possible. Also, this energy directly resolves the ambiguous internal representation problem [8], which causes back-propagation learning algorithms to be inefficient. By employing this energy along with its learning algorithm simultaneously, multilayer networks can be trained for various tasks.

The ambiguous internal representation problem (or the unfaithful representation problem) [8] [11] is severe for any BP learning algorithms [15] [18] [16]. Patterns will be misclassified when patterns of different classes are mapped to a same internal representation in any hidden layer. This representation is called the ambiguous internal representation (AIR). When an AIR exists in a hidden layer, it is impossible to classify these patterns no matter how many neurons or layers are added on to its succeeding layers. The AIR will cause premature saturation [4]. One phenomenon is that the output error remains a significantly high constant for an unpredictable period during training. One may follow the instruction in [8] to construct a multilayer network forwardly, layer after layer, without the occurrence of AIR. Note that the tiling construction for the multilayer network [11] does not have the AIR problem. It is a feed-forward construction with reduced hidden representations in each succeeding layer. Its internal representations carry different meanings from those obtained by various deep learning algorithms [19].

The deep neural network solved the internal representations in an unsupervised way. Optimal spaced hidden representations can hardly be reached by any learning algorithms for RBM [19]. This limits its applications in classification of noisy patterns and resolving blurred images. To improve the resolution of these representations for different patterns and resolve the AIR problem, this work studied a discriminative energy to construct a network forwardly with distances among distinct representations as large as possible.

One way to do this is to separate these representations as much as possible on each hidden layer from lower layers to upper layers so that each class has its own isolated representation in each layer. The outputs of each hidden layer are in a hypercube space, and each output is close to a corner of the hypercube. The representations are the outputs of their corresponding patterns and are distributed at certain hypercube corners. These representations must be separated so that different classes have different representations. One may use all the corners freely to achieve this separation. We impose a requirement to isolate each representation. We require the basin radius [9] of each representation to be as large as possible. This means that the Euclidean distance between a representation and its closest neighbor representation is as large as possible. All representations are allowed to evolve within the hypercube space and compete for basins under this requirement. We will develop its training process in the next section. Note that when the internal representations are given, one can use the algorithms in [9] to enlarge their basins for each hidden layer.

Another way to fully use all corners is to require that the topographic structure of these representations in the hypercube resembles that of the patterns in the input space. This is somewhat similar to the method in [6] [12]. That method transforms the patterns into new representations on a grid plane according to the nonlinear mapping of a trained multilayer network. These new representations keep the property that similar patterns have near representations in the plane. It is expected that the topography [3] of patterns can be preserved on the plane. That method combines both unsupervised learning and supervised learning to force the patterns to be mapped on a plane according to their geometric topography. Its goal is to construct a perfect topographic mapping on a grid plane such that one can handle many recognition and classification tasks on this plane. To preserve topology, we may modify the method in [12] for each layer's hypercube instead of the grid plane. This is not our goal. We will develop internal representations within each hidden layer's hypercube space which will facilitate operations of the network. Our goal is to resolve the AIR problem by developing a self-organization evolution to separate the internal representations as much as possible. This is also the goal of the transformation kernels used in the support vector machine [1] [2], which employs artificial kernels to transform difficult patterns into high-dimensional representations and then attempts to construct an optimal hyperplane to separate these representations in the high-dimensional space. This high-dimensional space is not a hypercube, and its representations are not allowed to evolve in this hypercube freely. This is because this machine uses fixed and limited mapping kernels. We use the layered network as an adjustable and flexible kernel which can be trained by patterns. We formulate a simple case in the next section to show the idea. An extended case is also included in the next section. We then illustrate the trained neural features in the third section. Discussions are included in the last section.

II. Separable Internal Representation Method

Single-layer perceptrons

We formulate the separable internal representation (SIR) method used to solve the AIR problem and derive its algorithm for the single-layer perceptron. Assume that the values of input units can only be -1 or 1. Consider the discriminative (or repellence) energy [5] [6] [10] [13] [14] [17],

$$E^{rep} = -\frac{1}{2}\sum_{p_1}^{p}\sum_{p_2}^{p}\left(d\left(\mathbf{y}^{(p_1)},\mathbf{y}^{(p_2)}\right)\right)^2 = \sum_{p_1}^{p}\sum_{p_2}^{p} E_{p_1p_2}$$

$$E^{rep} = -\frac{1}{2}\sum_{p_1}^{p}\sum_{p_2}^{p}\sum_{m=1}^{M}\left(y_m^{(p_1)} - y_m^{(p_2)}\right)^2 \qquad \text{(Eq. 1)}$$

where $\mathbf{y}^{(p_1)}(\mathbf{y}^{(p_2)})$ is an M-dimensional output representation corresponding to the p_1 (p_2) pattern. M is the number of neurons. This repellence energy will force the representations to evolve in an M-dimensional hypercube space. Instead of the Hamming distance, the Euclidean distance is used in the function d to ease the derivation. Consider P input patterns $\{\mathbf{x}^{(1)},\mathbf{x}^{(2)},\dots,\mathbf{x}^{(P)}\}$, where the r^{th} pattern $\mathbf{x}^{(r)} = [x_1^{(r)} x_2^{(r)} \dots x_N^{(r)}]^t$ is an N-tuple bipolar binary row vector. In this case, each pattern has its own class. Therefore, there are P classes. The vector $\mathbf{y}^{(r)}$ is the output vector of the hidden neurons corresponding to the input pattern $\mathbf{x}^{(r)}$. The goal is to maximize the distance between every pair of output representations such that each representation is isolated from all others as far as possible. The balance of all distances is indicated by the extreme value of the energy.

To achieve this goal, we reduce the energy $\boldsymbol{E}$ by means of the gradient descent rule. The algorithm for adjusting the weights to decrease this energy is described below.

By differentiation, the gradient descent of each individual energy E_{p1p2} is

$$\frac{\partial E_{p1p2}}{\partial w_{ij}} = -\sum_{m=1}^{M}\left(y_m^{(p_1)} - y_m^{(p_2)}\right)\left(\frac{\partial y_m^{(p_1)}}{\partial w_{ij}} - \frac{\partial y_m^{(p_2)}}{\partial w_{ij}}\right)$$
$$= -\frac{1}{2}\left(y_i^{(p_1)} - y_i^{(p_2)}\right) \times\left\{\left(\left(1-y_i^{(p_1)}\right)^2\right)x_j^{(p_1)} - \left(\left(1-y_i^{(p_2)}\right)^2\right)x_j^{(p_2)}\right\}$$

where

$$y_i = f(net_i), net_i = \sum_{j=1}^{N} w_{ij}x_j \text{ , and}$$

$$f(net_i) = \tanh(0.5net_i) = \frac{1-\exp(-net_i)}{1+\exp(-net_i)}$$

Note that

$$\frac{\partial(net_m^{(p_1)})}{\partial w_{ij}} = \frac{\partial(net_m^{(p_2)})}{\partial w_{ij}} = 0, \text{for } m \neq i$$

The updating equations for the weights are

$$w_{ij} \leftarrow w_{ij} - \eta\frac{\partial E}{\partial w_{ij}} \qquad \text{(Eq. 2)}$$

where η is a positive learning constant. The threshold values $w_{i(N+1)}$ are updated in exactly the same way as those for the weights. Their updating equations are

$$w_{i(N+1)} \leftarrow w_{i(N+1)} + \frac{\eta}{2}\left(y_i^{(p_1)} - y_i^{(p_2)}\right)\left(\left(y_i^{(p_1)}\right)^2 - \left(y_i^{(p_2)}\right)^2\right)$$

and the fixed input is of value $x_{N+1} = -1$.

The initial weights are set as $w_{ij} = 0, \forall i \neq j$ and $w_{ij} = 1, \forall i = j$. These are orthogonal weights. All the patterns will map to themselves using these weights. We then feed patterns one by one into the network and save their corresponding output vectors in an array. We calculate the Euclidean distance between every pair of output vectors. We use a square matrix $[D]$ to store these distances. The value of its entry $[D]_{rs}$ is the distance between the output vector $\mathbf{y}^{(r)}$ and the output vector $\mathbf{y}^{(s)}$ for the r th pattern and the s th pattern. Thus, the distance matrix $[D]$ is symmetric and has zeros in all its diagonal entries. Among all the pairs of output vectors, we select one pair that has the minimum distance. Then we use this pair of output vectors (indexed as p_1 and p_2) together with their corresponding patterns in Equation 2 to increase their distance.

For the next iteration, we feed all the patterns into the network again. We update the distance matrix $[D]$ to increase the minimum distance. We repeat this procedure until the minimum distance cannot be increased or it is greater than a predetermined value. See one example in [5].

We can extend this algorithm to the case of noisy patterns. Assume the patterns belong to classes $\mathbf{X}_1, \mathbf{X}_2, \dots \mathbf{X}_K$, where class $\mathbf{X}_k$ contains P_k patterns $\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(P_K)}\}$. The goal is to maximize the distance between every pair of output vectors that belong to different classes and minimize the distance between every pair of output vectors that belong to the same class. To achieve this goal, we need an algorithm that can provide the attraction force for the same class patterns. This can be done by reversing the sign of the energy, Equation 1. We include the algorithm below. The energy and the attraction force [10] [13] [14] [17], are

$$E^{att} = \frac{1}{2}\sum_{p_k^1=1}^{P_k}\sum_{p_k^2=1}^{P_k}\left(d\left(\mathbf{y}^{(p_k^1)}, \mathbf{y}^{(p_k^2)}\right)\right)^2 = \sum_{p_k^1=1}^{P_k}\sum_{p_k^2=1}^{P_k} E_{p_k^1 p_k^2} \qquad \text{(Eq. 3)}$$

and

$$\frac{\partial E_{p_k^1 p_k^2}}{\partial w_{ij}} = \left\{\left(1-\left(y_i^{(p_k^1)}\right)^2\right)x_j^{(p_k^1)} - \left(1-\left(y_i^{(p_k^2)}\right)^2\right)x_j^{(p_k^2)}\right\} \times \left(y_i^{(p_k^1)} - y_i^{(p_k^2)}\right) \qquad \text{(Eq. 4)}$$

where

$$E_{p_k^1 p_k^2} = \frac{1}{2}\left(d\left(\mathbf{y}^{(p_k^1)}, \mathbf{y}^{(p_k^2)}\right)\right)^2 = \frac{1}{2}\sum_{i=1}^{M}\left(y_i^{(p_k^1)} - y_i^{(p_k^2)}\right)^2.$$

To minimize E^{att}, we update the weights using the method of steepest descent as follows:

$$w_{ij} \leftarrow w_{ij} - \eta\frac{\partial E_{p_k^1 p_k^2}}{\partial w_{ij}} \qquad \text{(Eq. 5)}$$

for $k = 1, \dots, K$. The thresholds are adjusted in a similar way as that for E^{rep}.

The procedure for operating this attraction algorithm is similar as the former one. We randomly pick a pair of patterns from class $\mathbf{X}_k$. We use these two patterns as input vectors (denoted as $\mathbf{x}^{(p_k^1)}$ and $\mathbf{x}^{(p_k^2)}$) and feed them into the network to obtain output responses (denoted as $\mathbf{y}^{(p_k^1)}$ and $\mathbf{y}^{(p_k^2)}$). We calculate the distances between every pair of output vectors, which are produced by patterns in the same class. We select the pair which has the maximum distance and use this pair of output vectors and their corresponding input patterns in Equation 5 to decrease the distance.

We employ a mixed strategy to operate the repellence force in Equation 2 and the attraction force in Equation 5 in a sequential mode. We randomly select two patterns from all classes. When these two patterns come from the same class, we use Equation 5 to pull them close together; when they come from different classes we use Equation 2 to push them far apart from each other. The network is trained until the following two conditions are satisfied: (1) The maximum distance among all the pairs of output vectors belonging to the same class is below a predetermined threshold. (2) The minimum distance among all the pairs of output vectors belonging to different classes exceeds a predetermined threshold. Otherwise, the training will continue until no more improvement in either the maximum or minimum distance can be achieved.

III. Simulations

Characters Recognition

In order to explore the trained feature of each hidden neuron obtained by the energy E^{rep}, we show an experiment with two character patterns, 'A' and 'B' in Fig. 1, which will be used as the input patterns. Each pattern is a class of its own. We construct a single-layer perceptron with 256 input units and 10 output neurons, abbreviated as 256-10 network. Each output neuron is fully connected with all input units. The two characters are saved as binary images of size of 16 pixels × 16 pixels and represented as two 16-by-16 matrices $[A]$ and $[B]$ respectively. Each matrix element is a real number in [-1,1]. In the matrix, a white pixel is represented by 1, and a black pixel is represented by -1. Let $\mathbf{x}^A$, or $\mathbf{x}^B$, denote the raw vector that contains all the elements of the matrix $[A]$, or $[B]$, where

$$\mathbf{x}^A \equiv \langle x_k^A | x_k^A = [A]_{ij}, k = 16(i-1)+j, \forall i,j = 1,\dots,16 \rangle.$$

The size of each image is 16-by-16 pixels, so the number of input units of the network, N, is 256. The number of output neurons of the network, M, is manually set to 10 in this experiment. To simplify the experiment, we set the values of all thresholds to zero. The activation function of each neuron of the network is set as a hyperbolic tangent function, which maps any real number to the interval [-1, 1]. The function of each neuron is

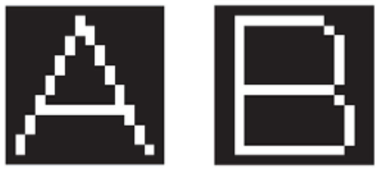

Fig. 1. Two character Images.

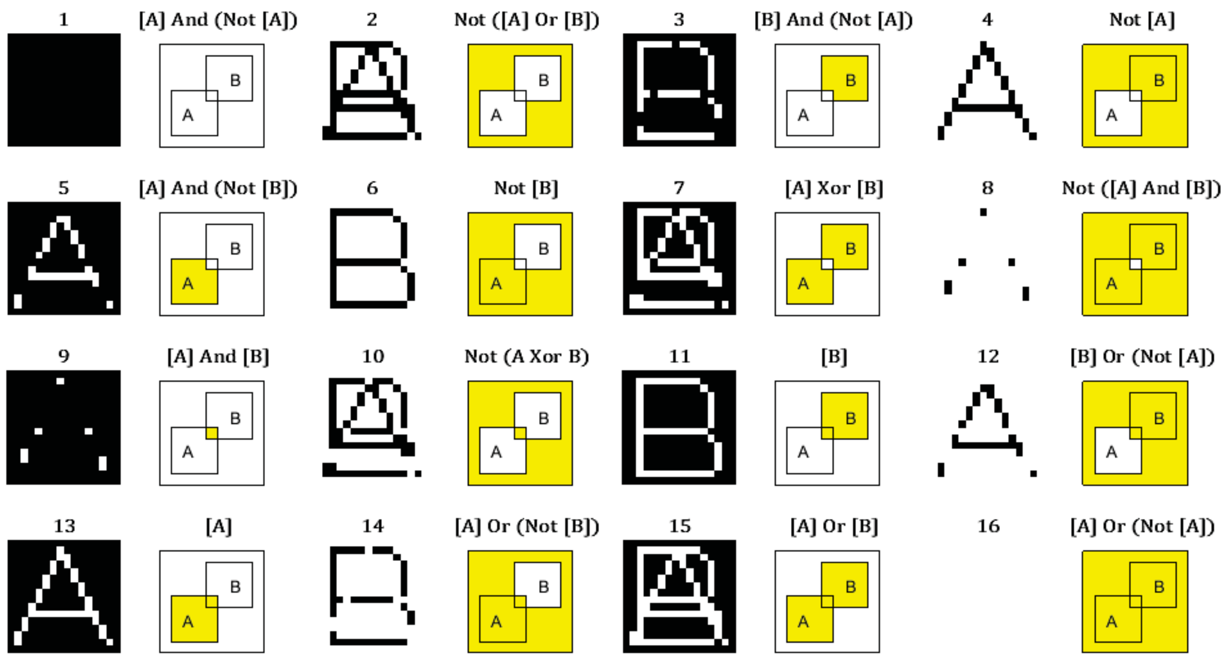

Fig. 2. 16 constructed weight features obtained from the sixteen logic functions that applied to the two matrices $[A]$ and $[B]$.

$$y_i^A = tanh\left(\sum_{j=1}^{N=256} w_{ij}x_j^A\right), \forall i = 1, \dots, M.$$

The energy is

$$E = E_{AB}^{rep} = \frac{1}{2}\sum_{i=1}^{M=10}(y_i^A - y_i^B)^2.$$

The training process will be stopped when there is no improvement in the energy value during 300 consecutive iterations.

TABLE I. ACTIVATION LEVELS OF NEURONS WITH WEIGHTS FROM THE SIXTEEN LOGIC FUNCTIONS

#	***Function of*** $[A]$ ***and*** $[B]$	*Pre-Activation: A*	*Pre-Activation: B*	*Pre-Activation: Difference*	*Sigmoid: A*	*Sigmoid: B*	*Sigmoid: Difference*
i	$\mathbf{w}_i$	$\mathbf{w}_i \cdot \mathbf{x}^A = \sum_{j=1}^{N} w_{ij}x_j^A$	$\mathbf{w}_i \cdot \mathbf{x}^B = \sum_{j=1}^{N} w_{ij}x_j^B$	$\mathbf{w}_i \cdot \mathbf{x}^A - \mathbf{w}_i \cdot \mathbf{x}^B$	y_i^A	y_i^B	$y_i^A - y_i^B$
1	$[A]$ And (Not $[A]$)	186	152	34	0.621	0.533	0.088
2	Not ($[A]$ Or $[B]$)	-166	-200	34	-0.571	-0.653	0.083
3	$[B]$ And (Not $[A]$)	96	242	-146	0.358	0.738	-0.379
4	Not $[A]$	-256	-110	-146	-0.762	-0.405	-0.357
5	$[A]$ And (Not $[B]$)	242	96	146	0.738	0.358	0.379
6	Not $[B]$	-110	-256	146	-0.405	-0.762	0.357
7	$[A]$ Xor $[B]$	152	186	-34	0.533	0.621	-0.088
8	Not ($[A]$ And $[B]$)	-200	-166	-34	-0.653	-0.571	-0.083
9	$[A]$ And $[B]$	200	166	34	0.653	0.571	0.083
10	Not ($[A]$ Xor $[B]$)	-152	-186	34	-0.533	-0.621	0.088
11	$[B]$	110	256	-146	0.405	0.762	-0.357
12	$[B]$ Or (Not $[A]$)	-242	-96	-146	-0.738	-0.358	-0.379
13	$[A]$	256	110	146	0.762	0.405	0.357
14	$[A]$ Or (Not $[B]$)	-96	-242	146	-0.358	-0.738	0.379
15	$[A]$ Or $[B]$	166	200	-34	0.571	0.653	-0.083
16	$[A]$ Or (Not $[A]$)	-186	-152	-34	-0.621	-0.533	-0.088

The energy will be improved when there exists significant difference, $y_i^A - y_i^B$, between the activation levels of each neuron while giving two different patterns to it. Its weights must be discriminative for these two patterns. We further analyze this discrimination using the weights that are obtained from the operation of logic functions defined on these two pattern matrices.

Since each element in the pattern matrix is a real number in $[-1,1]$. We define four basic logic functions on 16-by-16 matrices:

$$\begin{cases} \text{Not}: [-1,1]^{16\times16} \to [-1,1]^{16\times16} \\ \text{Not}([A]) = (\text{Not } [A]) = [R], \\ \textit{where } [R]_{ij} = -[A]_{ij}\ \forall i,j = 1,\dots,16 \end{cases};$$

$$\begin{cases} \text{Or}: [-1,1]^{16\times16} \times [-1,1]^{16\times16} \to [-1,1]^{16\times16} \\ \text{Or}([A],[B]) = ([A] \text{ Or } [B]) = [R], \\ \textit{where } [R]_{ij} = \max\left([A]_{ij},[B]_{ij}\right)\ \forall i,j = 1,\dots,16 \end{cases};$$

$$\begin{cases} \text{And}: [-1,1]^{16\times16} \times [-1,1]^{16\times16} \to [-1,1]^{16\times16} \\ \text{And}([A],[B]) = ([A] \text{ And } [B]) = [R], \\ \textit{where } [R]_{ij} = \min\left([A]_{ij},[B]_{ij}\right)\ \forall i,j = 1,\dots,16 \end{cases}; \text{ and}$$

$$\begin{cases} \text{Xor}: [-1,1]^{16\times16} \times [-1,1]^{16\times16} \to [-1,1]^{16\times16} \\ \text{Xor}([A],[B]) = ([A] \text{ Xor } [B]) = [R] \\ = \left([A] \text{ And } (\text{Not } [B])\right) \text{ Or } \left([B] \text{ And } (\text{Not } [A])\right) \end{cases}.$$

With these four basic functions, we can construct all 16 logic functions for the two matrices. All 16 logic functions are listed in TABLE I. The 16 constructed matrices obtained from the 16 functions are converted into images and plotted in Fig. 2. All 16 images are visually understandable. Each constructed matrix can be converted to the 256 weights of each neuron. Let $\mathbf{W}$ be a set that contains these 16 matrices. For each matrix in $\mathbf{W}$, we construct a neuron and convert this matrix to its 256 feature weights. Then, we inputted $\mathbf{x}^A$ and $\mathbf{x}^B$ respectively to each neuron, and recorded the activation levels of the neuron in TABLE I. In this table, $\mathbf{w}_i$ is a row vector, where $\mathbf{w}_i = [w_{i1}, w_{i2}, \dots, w_{iN}]$. The difference of activation levels between inputs $\mathbf{x}^A$ and $\mathbf{x}^B$ are also listed. Four neurons, {No. 3, 5, 12, and 14} or { $[B]$ And (Not $[A]$), $[A]$ And (Not $[B]$), $[B]$ Or (Not $[A]$), and $[A]$ Or (Not $[B]$) } produce large differences. These large differences are reasonable. They are the major differences between these two patterns. Using them as weights can greatly discriminate the two patterns. These four weight features are highly discriminative. Let $\mathbf{W}'$ be a collection of such highly discriminative weights, for example, these four weight matrices. We expect that decreasing the energy E^{rep} will generate certain highly discriminative weights that are similar to those in $\mathbf{W}'$. For two matrices, $[U]$ and $[V]$, the similarity $s([U],[V])$ is defined as

$$s([U],[V]) = \frac{\mathbf{x}^U}{|\mathbf{x}^U|} \cdot \frac{\mathbf{x}^V}{|\mathbf{x}^V|}.$$

For each neuron in the network, we calculate the similarities between its trained weights and the matrices in $\mathbf{W}$. The matrix in $\mathbf{W}$ which is the most similar to a neuron's weight matrix is called "the most similar function" of the neuron.

We did 5 experiments using the same 256-10 network and the same E^{rep} training process with five different settings for the initial weights of all 10 neurons. Their results are plotted in Fig. 3 to Fig. 8. In these figures, we use black for -1 and white for 1 in black-white images; black for 0, intensity of green for the values from 0 to 1, and intensity of red for the values from 0 to -1 in black-red-green figures. We discuss each of them in the following context.

For the first training, we set the initial weights with random numbers between [-1,1]. The result is shown in Fig. 3. We define the quantitative difference level of the i th neuron as $(y_i^A - y_i^B)^2/4$. We say that a neuron i "successively discriminates $\mathbf{x}^A$ and $\mathbf{x}^B$" only when this level is larger than 0.5. For the neurons that successively discriminate $\mathbf{x}^A$ and $\mathbf{x}^B$, "the most similar functions" of them are usually in $\mathbf{W}'$.

For the second training, we set the initial weights as the matrices in $\mathbf{W}'$, and its results are plotted in Fig. 4. All updates are unchanged during the training and all weights are exact the same as their initial weights. Every neuron generates the same activation levels when $\mathbf{x}^A$ and $\mathbf{x}^B$ are inputted. This is because that among all 256 pixels in [A] and [B], most of them are black pixels (-1) in the background. While multiplying with the weights of neurons, this black portion in the background always produces a sum with a very large value. So, the signs of the pre-activation values for $\mathbf{x}^A$ and $\mathbf{x}^B$ are always the same. Both of their pre-activation values have very large values, which will be mapped to -1 by the activation function, $y_i = f(net_i)$. So, any neuron

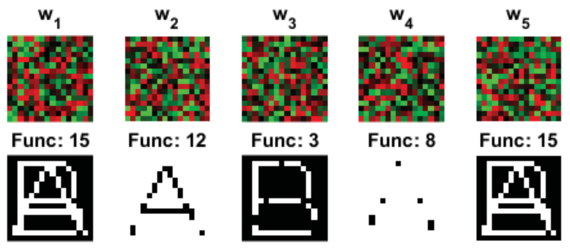

Fig. 3. Trained weights that are randomly initialized in [-1,1] and trained with SIR. Rows from the top are: trained wegihts, the most similar logic function. Their difference levels are 0.0, 1.0, 1.0, 0.0, and 0.0 respectively. The similarities to $[A]-[B]$ are -0.030, -0.060, -0.062, -0.020, -0.034.

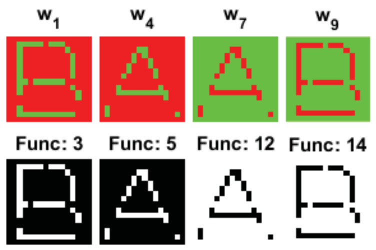

Fig. 4. Trained weights that are initialized as matrices in $\mathbf{W}'$ and trained with SIR. Rows from the top are: trained wegihts, the most similar logic function. Their difference levels are all 0.0.

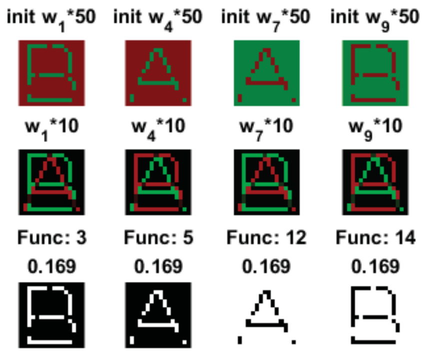

Fig. 5. Trained results by the weights that are initialized as those matrices in $\mathbf{W}'$ and scaled by 0.01. Rows from the top are: initial weights, trained wegihts, the most similar logic functions. Their difference levels are all 1.0. The similarities to $[A]-[B]$ are -0.996, 0.996, -0.996, and 0.996 respectively.

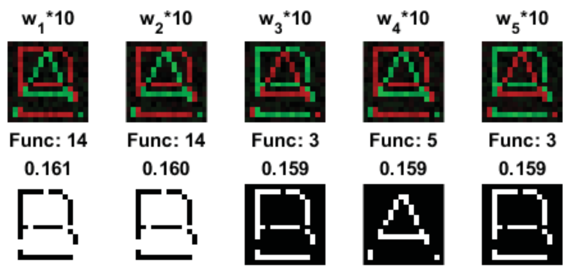

Fig. 6. Results obtained by the weights that are randomly initialized values in [-0.01,0.01]. Square of distnace between $\mathbf{y}^A$ and $\mathbf{y}^B$: 39.99. Rows from the top are: trained wegihts, the most similar logic function. Their difference levels are all 1.0. The similarities to $[A]-[B]$ are 0.986, 0.986, -0.987, 0.985, and -0.987 respectively.

initialized in this way does not discriminate $\mathbf{x}^A$ and $\mathbf{x}^B$ by the training process.

For the third training, we set the initial weights as those in the matrices $\mathbf{W}'$ and scaled them with a small number 0.01. The results are plotted in Fig. 5. Since we reduce the magnitudes of initial weights, $\mathbf{x}^A$ and $\mathbf{x}^B$ produce different levels and become separable during training. After training, all the weight matrices of the 10 neurons evolve to a single feature as those plotted in the first two images in Fig. 7. This is because the cost is maximized when all 10 neurons evolve to this most discriminative feature. This feature possesses a weight matrix formed by $\mathbf{w}_i = \mathbf{x}^A - \mathbf{x}^B$. The elements in the common parts of $[A]$ and $[B]$ are all 0 in $\mathbf{w}_i$. The elements that are different with opposite signs in $[A]$ and $[B]$ will be doubled. Using this feature $\mathbf{w}_i$ as the weights of a neuron, one can effectively distinguish $\mathbf{x}^A$ and $\mathbf{x}^B$ and map them into two distant codes.

In the fourth training, we set the initial weights with random real numbers between [-0.01, 0.01]. The results are shown in Fig. 6. All the 10 trained weight matrices are similar to the single feature $[A]-[B]$ or its negative version.

In the fifth training, we set the initial weights as $[A]-[B]$ or its negative version. The result is plotted in Fig. 7. The trained weights are exactly the same as the initial weights. The inputs $\mathbf{x}^A$ and $\mathbf{x}^B$ are mapped to two codes that

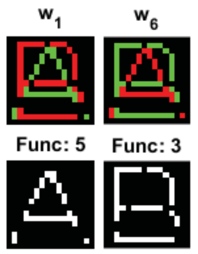

Fig. 7. Trained weights that are initialized as $[A]-[B]$ or $[B]-[A]$ and trained with SIR. Weights are not changed during training. Rows from the top are: trained wegihts, the most similar logic function. Their difference levels are all 1.0.

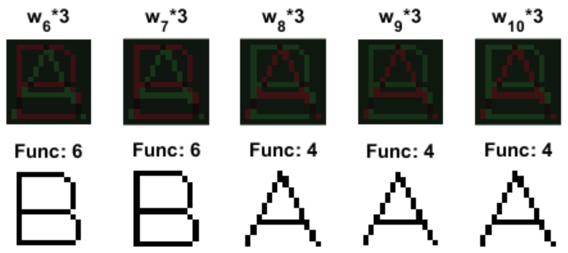

Fig. 8. Trained weights that are trained with Autoencoder [7]. Square of distance between $\mathbf{y}^A$ and $\mathbf{y}^B$: 33.47. Rows from the top are: trained wegihts, the most similar logic function.

are as far from each other as possible in the representation space [9]. This proves that $\mathbf{w}_i = \mathbf{x}^A - \mathbf{x}^B$ and its negative version are the best discriminative weights to distinguish $\mathbf{x}^A$ and $\mathbf{x}^B$. The SIR method can find the discriminative weights. To compare the results, Fig. 8 shows that the weight features trained with an auto-encoder [7] are similar to $[A]-[B]$ or its negative version. This is consistent with our study.

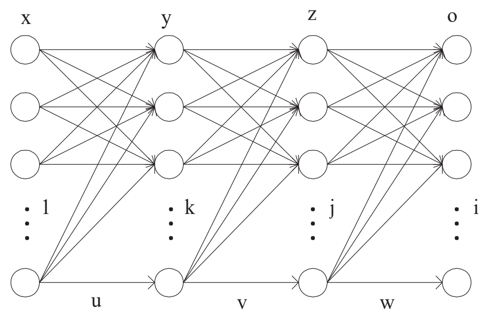

Fig.9. A Multilayer network.

The other approach to applying the single-layer method to the multilayer perceptron, Figure 9, is to extend this algorithm backwards to a deep bottom layer as the BP algorithm does. The reason for doing so is that we can take advantage of the nonlinear mapping ability of a multilayer perceptron to obtain ideal representations in the output layer. The derivation is similar to that for the BP algorithm. As before, we require that the distances between the output representations of different classes must be maximized. The weights between the output layer and the top hidden layer are adjusted based on the same updating rule used in Eq. 2. All the lower hidden layers are trained backwards. The local gradient of the upper layer is propagated to the next lower layer, and their weights are adjusted accordingly. The energy is

$$E^{rep} = -\frac{1}{2}\sum_{p_1=1}^{P}\sum_{p_2=1}^{P}\sum_{i=1}^{I}\left(o_i^{(p_1)} - o_i^{(p_2)}\right)^2. \quad \text{(Eq. 6)}$$

The local gradient $\boldsymbol{\delta}_{\mathrm{oi}}$ for the output neuron $\boldsymbol{o}_{\mathrm{i}}$ is defined as

$\boldsymbol{\delta}_{o_i} = \frac{\partial E}{\partial o_i}\frac{\partial o_i}{\partial net_i}$, where $\boldsymbol{o}_{\mathrm{i}}$ is obtained much as in Eq.2.

The local gradients for different input patterns p_1 and p_2 are

$$\boldsymbol{\delta}_{o_i}^{(p_1)} = \left(o_i^{(p_1)} - o_i^{(p_2)}\right)\left(\frac{1}{2}\left(1 - \left(o_i^{(p_1)}\right)\right)^2\right), \text{ and}$$

$$\boldsymbol{\delta}_{o_i}^{(p_2)} = \left(o_i^{(p_1)} - o_i^{(p_2)}\right)\left(\frac{1}{2}\left(1 - \left(o_i^{(p_2)}\right)\right)^2\right).$$

The local gradients for hidden neurons are obtained as

$$\delta_{z_j}^{(p_1)} = \frac{1}{2}\left(1 - \left(z_j^{(p_1)}\right)^2\right)\sum_r \delta_{o_r}^{(p_1)} w_{rj},$$

$$\delta_{z_j}^{(p_2)} = \frac{1}{2}\left(1 - \left(z_j^{(p_2)}\right)^2\right)\sum_r \delta_{o_r}^{(p_2)} w_{rj},$$

$$\delta_{y_k}^{(p_1)} = \frac{1}{2}\left(1 - \left(y_k^{(p_1)}\right)^2\right)\sum_r \delta_{z_r}^{(p_1)} v_{rk},$$

$$\delta_{y_k}^{(p_2)} = \frac{1}{2}\left(1 - \left(y_k^{(p_2)}\right)^2\right)\sum_r \delta_{z_r}^{(p_2)} v_{rk}.$$

The weights can be updated by the local gradient:

$$\Delta w_{ij} = \delta_{o_i}^{(p_1)} z_j^{(p_1)} - \delta_{o_i}^{(p_2)} z_j^{(p_2)},$$

$$\Delta v_{jk} = \delta_{z_j}^{(p_1)} y_k^{(p_1)} - \delta_{z_j}^{(p_2)} y_k^{(p_2)}, \text{ and}$$

$$\Delta u_{kl} = \delta_{y_k}^{(p_1)} x_l^{(p_1)} - \delta_{y_k}^{(p_2)} x_l^{(p_2)}.$$

We may reverse the sign of $\boldsymbol{E}^{\mathrm{rep}}$ to obtain the attraction energy. We omit its algorithm. We operate these two kind energies for every two patterns according to their class membership.

IV. Discussions

From the experiments above, we conclude that the best discriminative weight $\mathbf{w}_i$ to distinguish two patterns is the difference of the two input patterns, $\mathbf{w}_i = \mathbf{x}^A - \mathbf{x}^B$ or its negative version. In order to obtain other discriminative features, one may set the initial weights of the rest neurons with random numbers that are orthogonal to the trained features. New training patterns should not have components in those trained features.

Since it is tedious to do the same analyses for three or more patterns and many of their results are similar, we skip the discussions for them. The SIR method can pick the discriminative features among patterns and improve the energy value. Note that their common portions are gradually diminished and lost during the training process. Since these common portions are important in reconstruction of patterns, it is not recommended to use the energy E^{rep} in restorations, see the restoration example in [5].

The SIR method will exhaust the hidden space and maximize the utility of all neurons to accomplish highly separable representations of patterns. We can develop refined representations for patterns layer after layer or train a multilayer network backwardly to obtain such representations. We can use the network as an adjustable kernel to transform the patterns to a hypercube space with much isolated representations.

References

[1] B. Boser, I. Guyon, and V.N. Vapnik, “A training algorithm for optimal margin classifiers,” Fifth Annual Work'shop on computational Learning Theory, pp. 144-152, 1992.

[2] C. Cortes, and V.N. Vapnik, “Supprot vector networks,” Machine Learning, 20, pp. 273-297, 1995.

[3] T. Kohonen, “The self-organizing map,” Proceedings of the IEEE, 78, 1464-1480, 1990.

[4] Y. Lee, S. Oh, and M. Kim, “The effect of initial weights on premature saturation in back-propagation learning,” International Joint Conference on Neural Ñetworks, 1, pp. 765-770, 1991.

[5] C.-Y. Liou, H.-T. Chen, and J.-C. Huang, “Separation of internal representations of the hidden layer,” Proceedings of the International Computer Symposium, ICS, Chiayi, Taiwan, pp. 26-34, 2000.

[6] C.-Y. Liou and W.-C. Cheng, “Forced Accretion and Assimilation Based on Self-organizing Neural Network,” Self Organizing Maps - Applications and Novel Algorithm Design, Chapter 35 in Book edited by: Josphat Igadwa Mwasiagi, pp. 683-702, 2011.

[7] C.-Y. Liou, W.-C. Cheng, J.-W. Liou, D.-R. Liou, “Autoencoder for Words,” Neurocomputing, 139, pp. 84-96, 2014.

[8] C.Y. Liou, and W.J. Yu, “Ambiguous binary representation in multilayer neural networks,” IEEE International Conference on Neural Networks, 1, pp. 379-384, 1995.

[9] C.Y. Liou, and S.K. Yuan, “Error tolerant associative memory,” Biological Cybernetics, 81, pp. 331-342, 1999.

[10] J.C. Mao, and A.K. Jain, “Artificial neural networks for feature extraction and multivariate data projection,” IEEE Trans. on Neural Networks 6(2), pp. 296317, 1995.

[11] M. Mézard, and J.P. Nadal, “Learning in feed-forward layered networks: The tiling algorithm,” Journal of Physics A, 22, pp. 2191-2203, 1989.

[12] W. Pedrycz, and J. Waletzky, “Neural-network front ends in unsupervised learning,” IEEE Trans. on Neural Networks 8, pp. 390-401, 1997.

[13] B.D. Ripley, “Pattern recognition and neural networks,” Cambridge: Cambridge University Press, 1996.

[14] D.W. Ruck, S.K. Rogers, M. Kabrisky, M.E. Oxley, and B.W. Suter, “The multilayer perceptron as an approximation to a Bayes optimal discriminant function,” IEEE Trans. on Neural Networks, 1, No.4, pp. 296-298, 1990.

[15] D.E. Rumelhart, G.E. Hinton, and R.J. Williams, “Learning representations by back-propagating errors,” Nature (Longon), 323, 533-536, 1986.

[16] D.E. Rumelhart, and J.L. McClelland, “Parallel distributed processing: explorations,” the microstructure of cognition. vol. 1. Cambridge, MA: MIT Press, 1986.

[17] A.R. Webb, and D. Lowe, “The optimal internal representation of multilayer classifier networks performs nonlinear discriminant analysis,” Neural Networks, 3, pp. 367-37, 1990.

[18] P.J. Werbos, “Beyond regression: New tools for prediction and analysis in the behavioral sciences,” Ph.D. Thesis, Harvard University, 1974.

[19] G. E. Hinton, S. Osindero, and Y. Teh, “A fast learning algorithm for deep belief nets,” Neural Computation, 18, pp. 1527-1554, 2006.