Chapter 5

Separation of Internal Representations of the Hidden Layer

Abstract - We devise a method to separate the internal representations of the hidden layer where the Hamming distance between every two representations is required to be as large as possible. Each representation is isolated as far as possible from all others in the layer space. When the representations of certain patterns can be isolated within a Hamming radius, we can discriminate these patterns from all other patterns using a single neuron in the next upper layer. This space is a hypercube which is different from the grid plane used in a self-organizing map. Such representations will exhaust this hypercube uniformly and have tolerance for noisy patterns. This method directly resolves the ambiguous internal representation problem, which causes back-propagation learning\ to be inefficient. The layered network is developed as an adjustable kernel to separate multiple classes as much as possible. By employing this method along with the back-propagation learning algorithm, multilayer networks can be trained for various tasks.

1 Introduction

The multilayer networks (Rumelhart DE, and McClelland JL, 1986) have simple hierarchical architectures and are capable of pattern classification and recognition. Such networks consist of a set of sensory units that constitute the input layer, one or more hidden layers of computation units, and an output layer of computation units. The input signal propagates through the network in a forward direction from lower layers to upper layers. These networks are usually trained by the back propagation (BP) algorithm (Rumelhart DE, Hinton GE, and Williams RJ, 1986) (Werbos PJ, 1974). This algorithm is a supervised algorithm where we provide the desired output

for each input pattern during training. This algorithm measures the difference between the desired output and the actual output and adjusts the weights to reduce this difference. The ambiguous internal representation problem or the unfaithful representation problem (Liou C-Y, and Yu W-J, 1995) (Mézard M, and Nadal J-P, 1989) is severe for this algorithm. 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 lower hidden layer, it is impossible to classify these patterns no matter how many neurons or layers we add on to its upper layers. The AIR may cause premature saturation (Lee Y, Oh S, and Kim M, 1991). One phenomenon is that the output error remains a significantly high constant for an unpredictable period during training. One may follow the instruction in (Liou C-Y, and Yu W-J, 1995) and modify the algorithm in (Diamantaras K I, and Strintzis M G, 1998) to construct a multilayer network forwardly without the occurrence of AIR as done in (Chen J-L, 2000). The tiling construction for the multilayer network in (Mézard M, and Nadal J-P, 1989) does not have the AIR problem.

Following the instruction in (Liou C-Y, and Yu W-J, 1995), we will resolve this AIR problem by constructing a network forwardly with enlarged basins 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 such that each class has its own representation in each layer. The output of each hidden layer is in a hypercube space, and each output is a corner of this hypercube. The representations are the outputs of their corresponding patterns and are distributed at certain hypercube corners. These representations must be separated such that different classes have different representations. We may use all the corners freely to achieve this separation. We impose a requirement to isolate each representation. We require the basin of each representation to be as large as possible. This means that the distance between a representation and its closest neighbor representation is as large as possible. All representations are allowed to evolve in the hypercube and compete for basins under this requirement. We will develop this evolution in the next section. Note that when the internal representations are given, we can use the algorithms in (Liou C-Y, and Yuan S-K, 1999) to enlarge their basins for each neuron layer.

Another way to fully use the all the corners is to require that the topographic structure of these representations in the hypercube resemble that of the patterns in the input space. This is somewhat similar to the method in (Pedrycz W, and Waletzky J, 1997). 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 have the property that similar patterns have near representations in the

plane. It is expected that the topography (Kohonen T, 1990) of patterns can be perfectly 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 accomplish a perfect topographic mapping on a grid plane such that one can manipulate many recognition and classification tasks on this plane (or display). To preserve topology, we may modify the method in (Pedrycz W, and Waletzky J, 1997) for each layer's hypercube instead of the grid plane. This is not our goal. We will develop internal representations in each hidden layer's hypercube which will facilitate operation 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 (Boser B, Guyon I, and Vapnik VN, 1992) (Cortes C, and Vapnik VN, 1995), which employs inner-product 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 these 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 demonstrate the method. An extended case is also included in the next section. We then present applications in the third section. Discussion is included in the last section.

2 Separable Internal Representation Method

2.1 Single-layer perceptrons

We will now 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 distance (or repellence) energy (Ripley B D, 1996) (Mao Jianchang, and Jain Anil K, 1995) (Webb AR, and Lowe D, 1990) (Ruck DW, Rogers SK, Kabrisky M, Oxley ME, and Suter BW, 1990),

$$E^{rep} = -\frac{1}{2}\sum_{p_1}^{P}\sum_{p_2}^{P}(d(y^{(p_1)}, y^{(p2)}))^2$$

$$= \sum_{p_1}^{P} \sum_{p_2}^{P} E_{p_1 p_2}$$

$$E^{rep} = -\frac{1}{2} \sum_{p_1}^{P} \sum_{p_2}^{P} \sum_{m=1}^{M} (y_m^{(p_1)} - y_m^{(p2)})^2 \qquad (1)$$

where $y^{(p_1)}(y^{(p2)})$ is an *M*-dimensional output representation corresponding to the p_1th (p_2th) 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 as the distance function *d* to ease the derivation. Consider *P* input patterns $\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(p)}\}$, where the *r*th pattern $\mathbf{x}^{(r)} = \left[x_1^{(r)} x_2^{(r)} .. x_N^{(r)}\right]^t$ is an *N*-tuple bipolar binary 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. For example, to uniformly distribute three representations in a 3-cube as shown in Figure 1, **{ [-1 –1 –1]^t, [1 1 –1]^t, [-1 1 1]^t }** is one of the ideal solutions and **{ [1 1 1]^t, [1 -1 –1]^t, [-1 1 -1]^t }** is another. In this case, the balanced Hamming distance for all three representations is the same, which is 2 as shown in Figure 2.

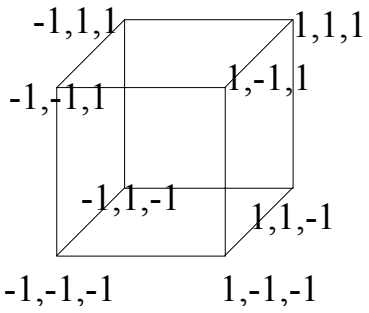


Figure 1: The 3-cube.

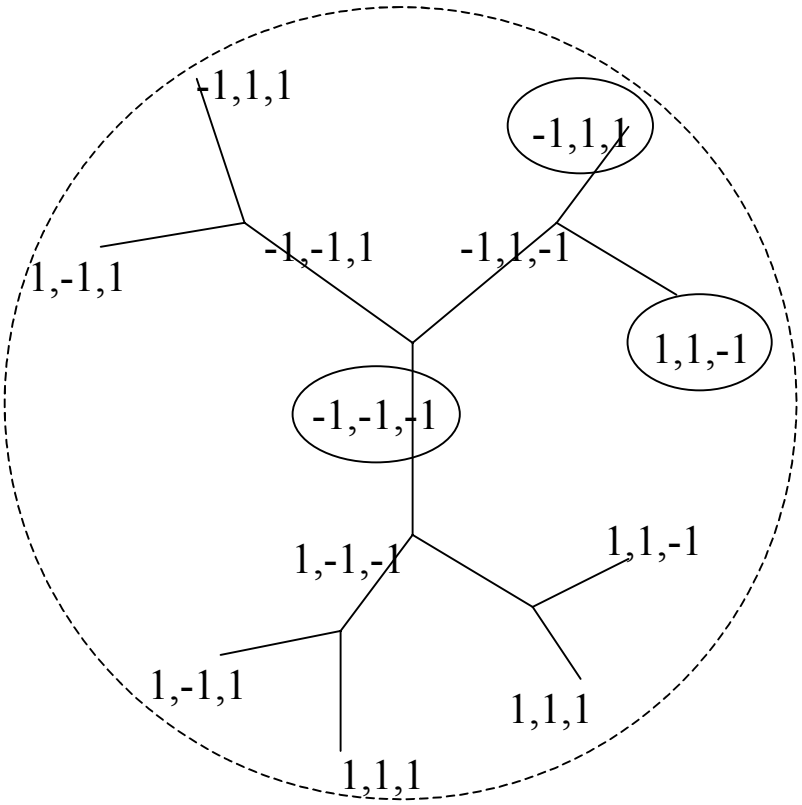


Figure 2: The Hamming distance is 2 between any two corners {(-1 –1 –1), (-1 1 1), (1 1 –1)}.

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 $\boldsymbol{E}_{\mathrm{p1p2}}$ is

$$\frac{\partial E_{p1p2}}{\partial w_{ij}} = -\sum_{m=1}^{M}\left(y_m^{(p1)} - y_m^{(p2)}\right)\left(\frac{\partial y_m^{(p1)}}{\partial w_{ij}} - \frac{\partial y_m^{(p2)}}{\partial w_{ij}}\right)$$

$$= -\sum_{m=1}^{M}\left(y_m^{(p1)} - y_m^{(p2)}\right)\left(\frac{\partial f(net_m^{(p1)})}{\partial (net_m^{(p1)})}\frac{\partial (net_m^{(p1)})}{\partial w_{ij}} - \frac{\partial f(net_m^{(p2)})}{\partial (net_m^{(p2)})}\frac{\partial (net_m^{(p2)})}{\partial w_{ij}}\right)$$

$$= -\frac{1}{2}\left(y_i^{(p1)} - y_i^{(p2)}\right)\left\{\left(1-\left(y_i^{(p1)}\right)^2\right)x_j^{(p1)} - \left(1-\left(y_i^{(p2)}\right)^2\right)x_j^{(p2)}\right\} \quad ,$$

where

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

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

Note that

$$\frac{\partial(net_m^{(p1)})}{\partial w_{ij}} = \frac{\partial(net_m^{(p2)})}{\partial w_{ij}} = 0; \quad for \quad m \neq i.$$

The updating equations for the weights are

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

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

$$w_{i(N+1)} \leftarrow w_{i(N+1)} + \frac{\eta}{2}(y_i^{(p1)} - y_i^{(p2)})((y_i^{(p1)})^2 - (y_i^{(p2)})^2),$$

and the fixed input is of value $\boldsymbol{x}_{\{N+1\}} = -1$.

The initial weights are set as $\boldsymbol{w}_{ij} = 0$ for all i≠j and $\boldsymbol{w}_{ij} = 1$ for 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 $\boldsymbol{D}$ to store these distances. The value of its entry $\boldsymbol{D}_{rs}$ is the distance between the output vector $\boldsymbol{y}^{(r)}$ and the output vector $\boldsymbol{y}^{(s)}$ (in response to the $\boldsymbol{r}$th pattern and the $\boldsymbol{s}$th pattern). Thus, the distance matrix $\boldsymbol{D}$ is symmetric and has zeros in all its diagonal entries. Among all the pairs of output vectors, we find one pair that has the minimum distance. Then we use this pair of output vectors (indexed as $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$) 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 $\boldsymbol{D}$ and increase the minimum distance. We repeat this procedure until the minimum distance cannot be increased or it is greater than a predetermined value.

We can extend this algorithm to the case of noisy patterns. Assume the patterns belong to classes $\boldsymbol{X}_1, \boldsymbol{X}_2, \ldots \boldsymbol{X}_K$, where class $\boldsymbol{X}_k$ contains $\boldsymbol{P}_k$ patterns $\{ \boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(Pk)} \}$. 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 function, Equation 1. We include the algorithm below.

The energy function and the attraction force (Ripley B D, 1996) (Mao Jianchang, and Jain Anil K, 1995) {cite} (Webb AR, and Lowe D, 1990) (Ruck DW, Rogers SK, Kabrisky M, Oxley ME, and Suter BW, 1990), are

$$E^{att} = \frac{1}{2}\sum_{p_k^1}^{P_k}\sum_{p_k^2}^{P_k}(d(y^{(p_k^i)}, y^{(p_k^2)}))^2 = \sum_{p_k^1}^{P_k}\sum_{p_k^2}^{P_k} E_{p_k^i p_k^2} \qquad (3)$$

and

$$\frac{\partial E_{p_k^1 p_k^2}}{\partial w_{ij}} = (y_i^{(p_k^1)} - y_i^{(p_k^2)})\{(1-(y_i^{(p_k^1)})^2)x_j^{(p_k^1)} - (1-(y_i^{(p_k^2)})^2)x_j^{(p_k^2)}\}, \quad (4)$$

where

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

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

$$w_{ij} \leftarrow w_{ij} - \eta \frac{\partial E_{p_k^1 p_k^2}}{\partial w_{ij}}, \quad (5)$$

for k = 1,...,$\boldsymbol{P}_{\text{k}}$. The thresholds are adjusted in a similar way as that for $\boldsymbol{E}^{\text{rep}}$.

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

2.2 Multilayer perceptrons

There are two ways to apply the above idea to a multilayer perceptron. Let us start with the one used to solve the AIR problem. The training process starts from the

bottom layer of the network. At the beginning, we focus on the input layer and the first hidden layer. They are viewed as a single-layer perceptron, and we use the SIR method for the single-layer perceptron to adjust the weights between these two layers. When the training is finished, we collect all the output responses of the first hidden layer as the first group of internal representations for the patterns. Then we shift to the second hidden layer. We train the weights between the first hidden layer and the second hidden layer using the collected first internal representations as inputs and using the SIR method for the single-layer perceptron. We then collect the second group of internal representations of the second hidden layer as inputs to train the weights between the second hidden layer and the 3rd layer. We proceed from the bottom hidden layer to the top layer until the output layer is completed. In this procedure the updating equations Equation 2 and Equation 5 are used repeatedly layer after layer with refined internal representations. We use forward training for the network and refine the representations layer after layer. After the training is finished, the representations of different classes are separated as much as possible, and representations of a class are clustered (or merged) together as closely as possible. This is different from the BP approach. This forward training is free from the premature saturation problem and the AIR problem. We may add other kinds of forces to obtain certain desired outputs gradually.

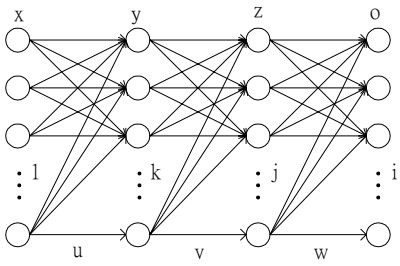


Figure 3: A Multilayer network.

The other approach to applying the single-layer method to the multilayer perceptron shown in Figure 3 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. We expect that a multilayer perceptron will have the potential to uniformly distribute the representations on hypercube corners and to map the fewest corners for each class of noisy patterns. 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 Equation 2. All the lower hidden layers are trained backwards without any free

evolution. The local gradient of the upper layer is propagated to the next lower layer, and their weights are adjusted accordingly. The energy function is

$$E^{rep} = -\frac{1}{2}\sum_{p_1=1}^{P}\sum_{p_2=1}^{P}(d(o^{(p_1)}, o^{(p2)}))^2$$

$$= \sum_{p_1=1}^{P}\sum_{p_2=1}^{P} E_{p_1 p_2}$$

$$E^{rep} = -\frac{1}{2}\sum_{p_1=1}^{P}\sum_{p_2=1}^{P}\sum_{i=1}^{I}(o_i^{(p_1)} - o_i^{(p2)})^2 \qquad (6)$$

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

$$\delta_{o_i} = \frac{\partial E}{\partial o_i}\frac{\partial o_i}{\partial net_i},$$

where $\boldsymbol{o}_{\mathrm{i}}$ is obtained much as in Equation (2). We calculate the local gradients for different input patterns p_1 and p_2. They are

$$\delta_{oi}^{(p1)} = (o_i^{(p1)} - o_i^{(p2)})(\frac{1}{2}(1-(o_i^{(p1)})^2)) \quad \textit{and}$$

$$\delta_{oi}^{(p2)} = (o_i^{(p1)} - o_i^{(p2)})(\frac{1}{2}(1-(o_i^{(p2)})^2)).$$

Accordingly, the local gradients for hidden neurons are obtained as

$$\delta_{zj}^{(p1)} = \frac{1}{2}(1-(z_j^{(p1)})^2\sum_r \delta_{or}^{(p1)} w_{rj},$$

$$\delta_{zj}^{(p2)} = \frac{1}{2}(1-(z_j^{(p2)})^2\sum_r \delta_{or}^{(p2)} w_{rj};$$

$$\delta_{yk}^{(p1)} = \frac{1}{2}(1-(y_k^{(p1)})^2\sum_r \delta_{zr}^{(p1)} v_{rk},$$

$$\delta_{yk}^{(p2)} = \frac{1}{2}(1-(y_k^{(p2)})^2\sum_r \delta_{zr}^{(p2)} v_{rk}.$$

The equations listed above show that the local gradients are the weighted sums of the local gradients of their connected upper layer. Then the weights can be updated by the local gradient:

$$\Delta w_{ij} = \delta_{oi}^{(p1)} z_j^{(p1)} - \delta_{oi}^{(p2)} z_j^{(p2)},$$

$$\Delta v_{jk} = \delta_{zj}^{(p1)} y_k^{(p1)} - \delta_{zj}^{(p2)} y_k^{(p2)},$$

$$\Delta u_{kl} = \delta_{yk}^{(p1)} x_l^{(p1)} - \delta_{yk}^{(p2)} x_l^{(p2)}.$$

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

3 Simulations

3.1 Characters Recognition

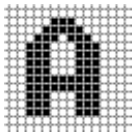
In this section we test the proposed method with experiments. The first experiment is recognition of characters. The pattern set contains 52 characters (A to Z and a to z). Each character is stored as a binary image of size of 16 pixels × 16 pixels as shown in Figure 4. Each pattern is a vector containing one image. Each pattern is a class of its own. We construct a single-layer perceptron with 256+1 input units and 256 output neurons. Each output neuron is fully connected with all input units and a 

Figure 4: A character image.

threshold unit. The training results are shown in Figure 5, Figure 6 and Figure 7. In Figure 5, we plot the sorted 52 minimum Hamming distances for all 52 characters where each minimum distance is the distance between one pattern and its closest pattern. For each output representation, we calculate the Hamming distances to all other 51 output vectors and record the minimum one. We plot the performances of the SIR method under different initial conditions. As shown in Figure 5, the minimum distances for all patterns are all less than 90 (the curve marked with -input-). The minimum distances of the output representations are all greater than 100. The performance curve marked with -output 1- is obtained by using the orthogonal initial weights. The curve marked with -output 2- is obtained by using the small random initial weights. We also use the multilayer perceptron as in Figure 3 to do this experiment and plot the performances in this figure. In this multilayer perceptron, each layer has 256 neurons. The performance curve marked with -output 3- is obtained by setting the orthogonal initial weights for all layers. The curve marked with -output 4- is obtained by setting the random initial weights. From this figure, the representations have larger distances than those of patterns. It will be relatively easier to separate these representations in the hidden layer space than separate the image patterns in the input space.

To see the distribution of these representations, we assume each representation shares equal number of basin corners, (2^{256}/52), in the hidden hypercube and these corners are connected neighbors in this hypercube. The output representation of each

pattern is considered as the center of these corners. Therefore, a center should be at a 222 Hamming distance to another center. The radius of the basin is less than 111 because $\sum_{i=0}^{111}\binom{256}{i} > 2^{256}/52$. Thus the distance between two centers is approximately 222. These kind round basins are ideal. The experiments show that we can separate the representations with a distance more than half the idea radius. To show the sizes of these trained basins, we plot the maximum Hamming distance between a representation and all others in Figure 6. We also plot the averaged Hamming distance for each representation in Figure 7. As shown in Figure 6, several maximum Hamming distances approach the ideal radius 222. With such well separated representations, one can restore noisy representations using the Hamming distance.

When we use the output representations obtained by this single layer perceptron as inputs to train the second hidden layer. Then use the output representations of the second hidden layer as inputs to train the third hidden layer. The output representations of the third hidden layer will approach to the performance curve, the -output 4-. In this case all layers have 256 neurons.

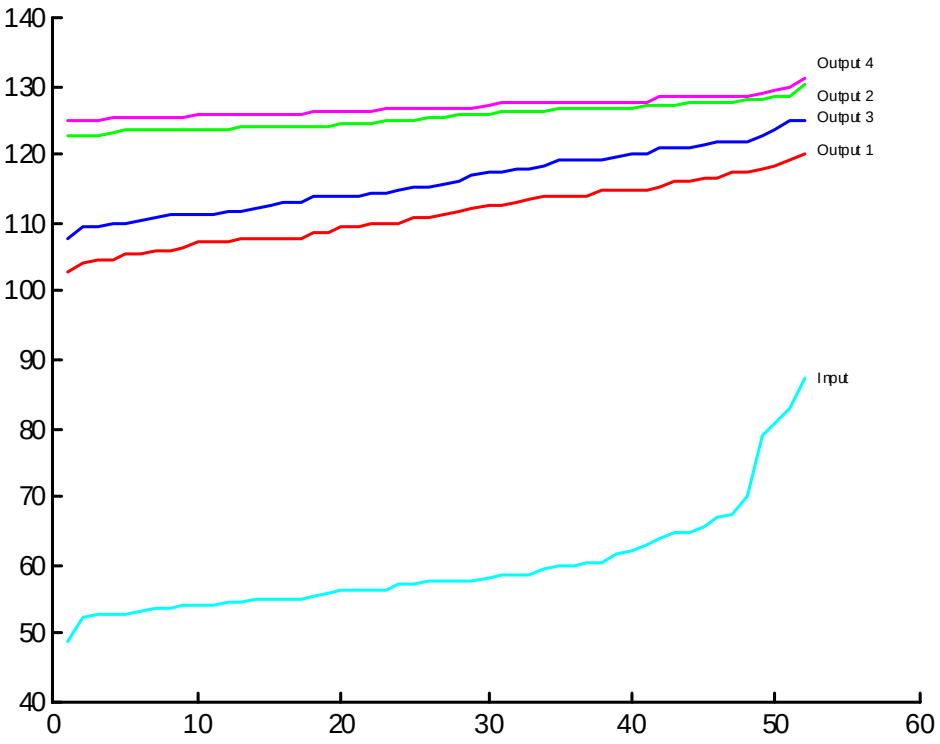


Figure 5: The minimum Hamming distances for the 52 representations by SIR method.

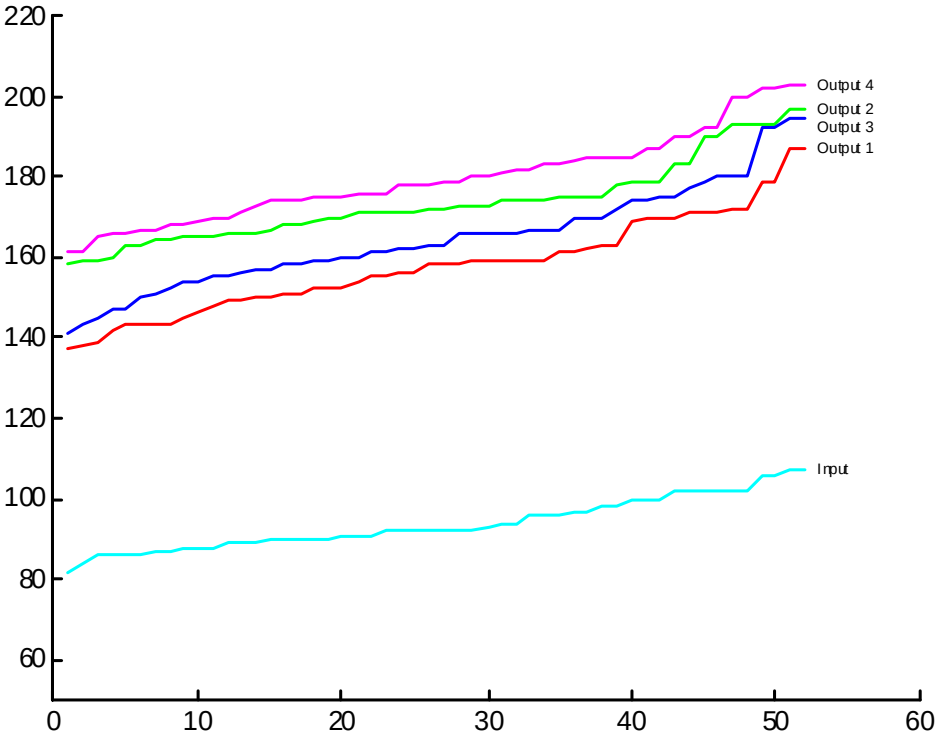


Figure 6: The maximum distances for the 52 representations by SIR method.

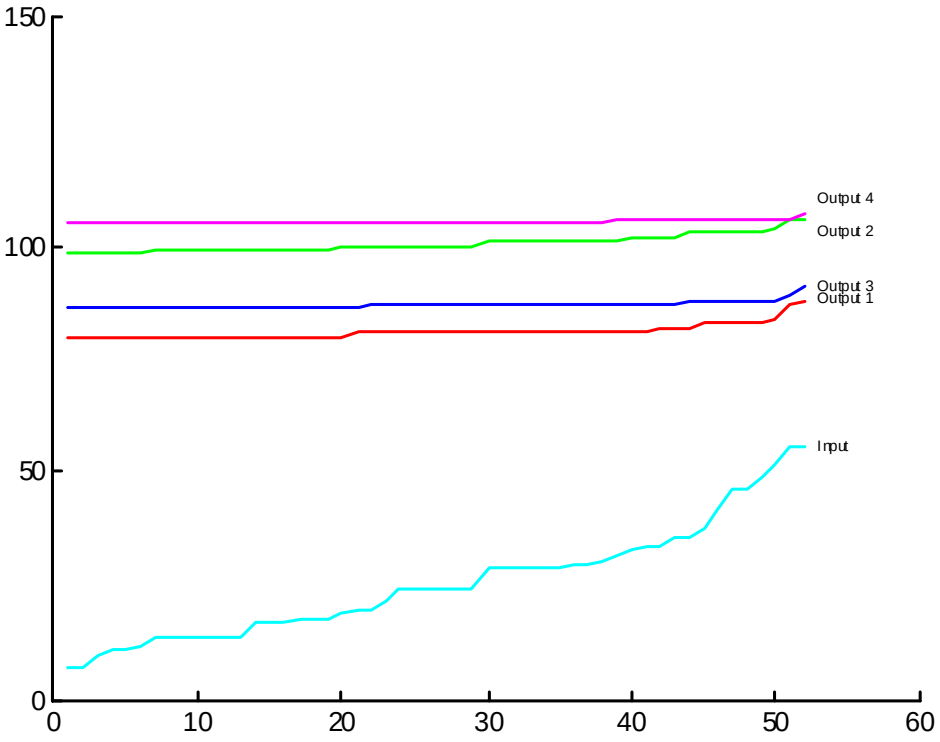


Figure 7: The averaged Hamming distances for the 52 representations by SIR

method.

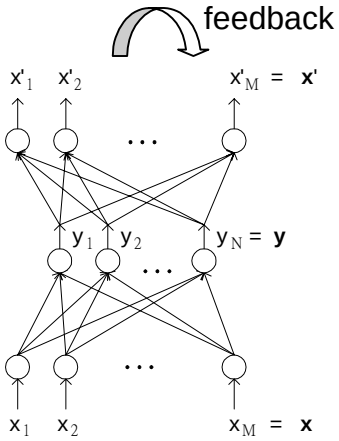


Figure 8: The recurrent network

3.2 Recurrent Associative Memory

In this test, we use the SIR method to develop the network shown in Figure 8 as an associative memory. There are three layers in this network, the input units, the hidden layer, and the output layer. This network is similar to the replicator network (Hecht-Nielsen R, 1995) with feedbacks. The response of the output layer will be send back to the input layer in the next iteration. There are only two layers with sigmoid function neurons. The input layer distributes signals to the hidden layer directly without any modification. The input layer and the hidden layer are used to develop highly separable international representations for the above 52 patterns to tolerate noisy patterns. The output layer is used to index these representations to their corresponding patterns. As an associative memory, the output will evolve to a stable state gradually. This stable state is the place where we store the pattern. Given a corrupted pattern (*search argument*), one corresponding stored pattern will be recalled through the association of this corrupted pattern and this memorization mechanism.

The training algorithm of this network is divided into two stages. In the first stage, we evolve the weights between the input units and the hidden layer by the SIR method. In the second stage, we train the weights between the hidden layer and the output layer by the BP algorithm using the 52 internal representations as inputs and their corresponding patterns as the desired outputs. In this case, each layer contains 256 neurons plus one fixed unit with value -1. All neurons in a layer are fully connected to the neurons of the next upper layer. In the first stage, we use small random numbers as initial weights to start the training of the weights between the input units and the hidden layer. The results of the SIR training are included in the

former section. We then save the 52 internal representations as inputs and their corresponding patterns as the desired outputs, {(y^p, x^p), P=1,..,52}, and use them to train the weights between the hidden layer and the output layer. In the second stage, all trained weights between the input units and the hidden layer must be fixed. In this stage, we only train the weights between the hidden units and the output layer using the BP algorithm:

$$v_{ki} \leftarrow v_{ki} + \eta \frac{1}{2}(x_k^p - x'_k)(1-(x'_k)^2 y_i^{(p)}.$$

This is the delta training rule for the bipolar continuous activation function. The desired output for the representation $\boldsymbol{y}^p$ is $\boldsymbol{x}^p$ and the network output is $\boldsymbol{x'}$. The training will stop when the network outputs are the same as the their corresponding patterns.

After training, we feed corrupted patterns to the network. The corrupted patterns are generated by randomly reversing 30% of the 256 image pixels. Successive responses of the output layer are recorded in Figure 9. Figure 9 shows the refined characters for the first five iterations. Most corrupted patterns will evolve to stable patterns within five iterations.

The SIR method has been used as the side direction method in (Liou C-Y, and Yang H-C, 1999) to solve the handprinted character recognition problem. This method has been used to solve the uniform resource placement problem (Chiu G-M, and Raghavendra C S, 1990) (Livingston M, and Stout Q F, 1988).

4 Discussions

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 use the network as an adjustable kernel to transform the patterns to a hypercube space with much isolated representations. We summarize the relations between the SIR method and other methods.

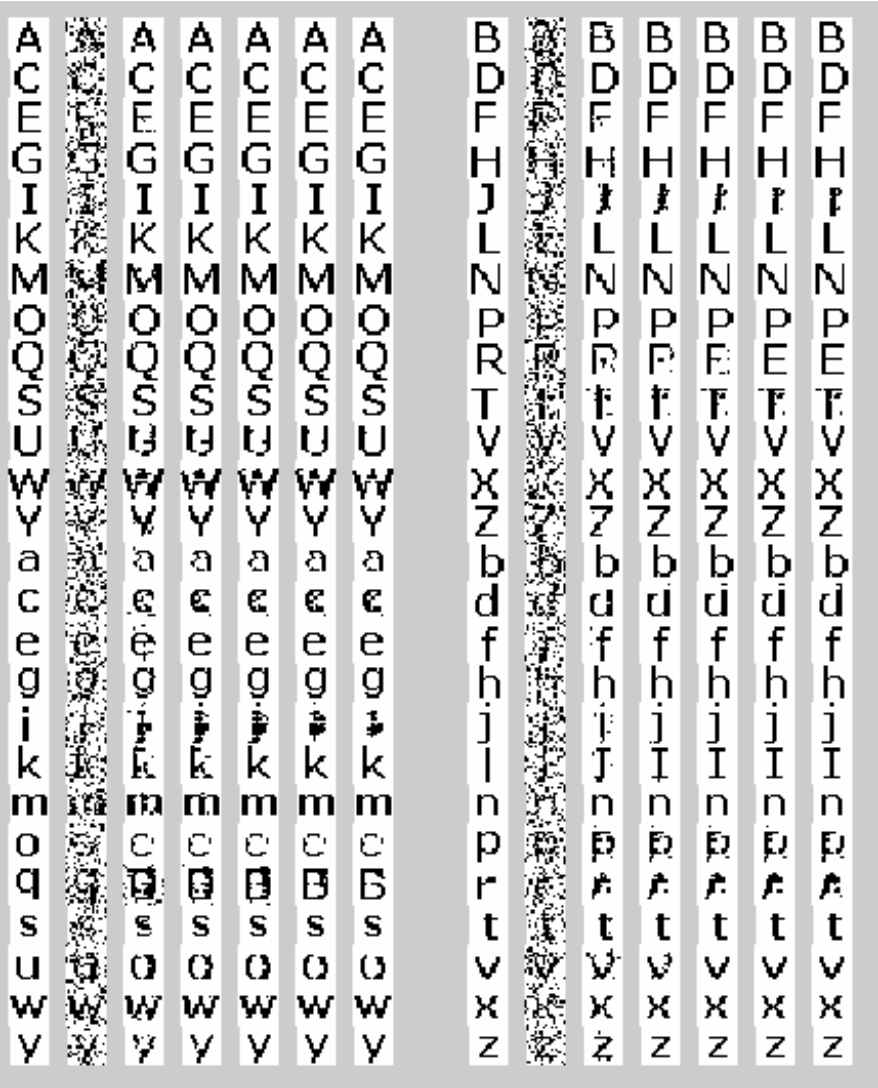


Figure 9: Evolutionary recall of characters. The training character is in the first column. The 30% corrupted character is in the second column. The recalled characters for the first five iterations are in the rest five columns.

4.1 Hebbian Learning

In the updating equations, the terms $(1-(y_i^{(p1)})^2)$ and $(1-(y_i^{(p2)})^2)$ have nonnegative values. These two terms are the derivatives of the activation function f(net_i))=tanh($0.5net_i$) . Their values approach 1 when $y_i^{(p1)}$ and $y_i^{(p2)}$ approach -1 or 1. If we substitute 1 for these terms, the learning rule becomes

$$w_{ij}(n+1) \leftarrow w_{ij}(n) + \eta\,(y_i^{(p1)}(n) - y_i^{(p2)}(n))(x_i^{(p1)}(n) - x_i^{(p2)}(n)) \tag{7}$$

which is in some similar to the Hebbian learning. The activation strength is proportional to the difference between two patterns and between postsynaptic responses (output vectors). The increase of the strength of a synapse is proportional to such differences on both sides of that synapse synchronously. To compare, a Hebbian learning form is

$$w_{ij}(n+1) \leftarrow w_{ij}(n) + \eta\ y_i(n)x_j(n) \qquad (8)$$

Another interesting form called the covariance hypothesis was introduced in (Sejnowski TJ, 1977) (Sejnowski TJ, 1997). According to this hypothesis, the learning applied to the synaptic weight $\boldsymbol{w}_{ij}$ is defined by

$$w_{ij}(n+1) \leftarrow w_{ij}(n) + \eta\ (y_i(n) - \bar{y}(n))(x_i(n) - \bar{x}(n)) \qquad (9)$$

where $\bar{x}(n)$ and $\bar{y}(n)$ denote the time-averaged values of and y_i, respectively. Comparing Equation 8 with the Equation 9, the differences between them are the presynaptic and postsynaptic reference thresholds, which determine the sign of synaptic modification. In Equation 7, instead of the time-averaged references, the presynaptic signal and the postsynaptic signal use the other signals as the references. From Equation 7 we see that the synaptic weight $\boldsymbol{w}_{ij}$ is enhanced when (a) the conditions $x_j^{(p1)} > x_j^{(p2)}$ and $y_i^{(p1)} > y_i^{(p2)}$ are satisfied or (b) the conditions $x_j^{(p1)} < x_j^{(p2)}$ and $y_i^{(p1)} < y_i^{(p2)}$ are satisfied.

4.2 Mutual Information

The proposed algorithms are based on the maximization the representations' distances among the different class representations and the minimization the distances among the same class representations. There is a similar network (Becker S, and Hinton GE, 1992) with two modules and a different goal. It maximizes the mutual information, $\mathbf{I}(\mathbf{Y}_a; \mathbf{Y}_b)$, where $\mathbf{Y}_a$ and $\mathbf{Y}_b$ are the output vectors corresponding to the input patterns $\mathbf{X}_a$ and $\mathbf{X}_b$. This mutual information is defined as

$$I = 0.5 \log \frac{V(Y_a + Y_b)}{V(Y_a - Y_b)}$$

where $\boldsymbol{V}$ is the variance over the responses of the training samples. This network is shown in Figure10. The goal of this network is to make the outputs $\mathbf{Y}_a$ and $\mathbf{Y}_b$ of the two modules to agree closely (i.e., to have a small expected squared difference)

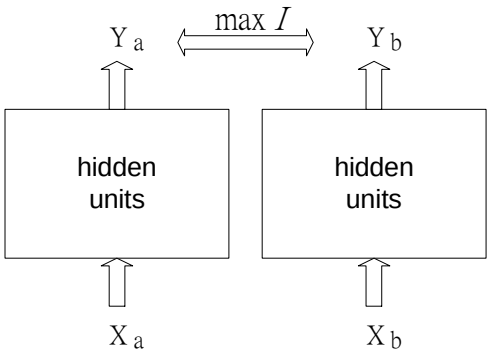


Figure 10: The two-module network.

corresponding to a closely related pair of input patterns $\mathbf{X}_a$ and $\mathbf{X}_b$. In the same time, the two modules cannot just produce constant outputs that is unaffected by the input patterns, otherwise, they convey no information. The outputs of these two modules should vary as the inputs are varied. If we replace this two-module network with a single-module network as shown in Figure 11 and confine the output responses in a hypercube space. We then train this network to maximize the object information function $I' = 0.5 \log V(Y_a - Y_b)$ for different class patterns and minimize this function for patterns in the same class. We obtain similar results as those using the SIR method.

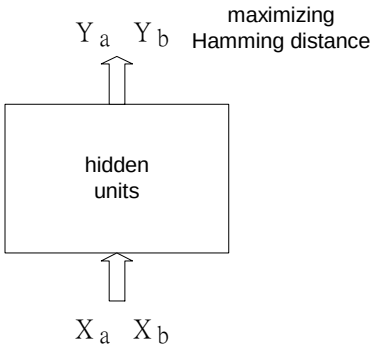


Figure 11: The single-module network

Note that this object function will weight frequent patterns. In our experiments all patterns have equal appearance (uniform probability distribution). The mean value of the vector $(y^{(p1)} - y^{(p2)})$ is zero. Assume each pattern of $\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(P)}\}$ has its own representation, $y^{(p1)} \neq y^{(p2)}$ for p1 ≠ p2, and equal probability of appearance, P^{-1}. Then $I' = 0.5 \log\{-2E^{rep}\} - \log P$. The information function, I', is coherent with the repellence energy or the attraction energy. These two energies also

have mutual information content.

Since $E^{rep} = -\frac{1}{2}\sum_{p1}^{P}\sum_{p2}^{P}\sum_{m=1}^{M}(y_m^{(p1)} - y_m^{(p2)})^2$, the SIR method tends to maximize (or minimize) the variance of each neuron's output difference, $V(y_m^{(p1)} - y_m^{(p2)})$, evenly for all pairs of different class patterns (or same class patterns). All neurons will be devoted to these class patterns. All neurons are sensitivitive to these patterns only. Any unknown pattern will be included in one of these patterns' representations. In other words, these representations exhaust the pattern space.

Several advanced methods further develop this two-module network with modified object functions to accomplish various tasks (Ukrainec A, and Haykin S, 1992) (Ukrainec AM, and Haykin S, 1996) (Schmidhuber J, and Prelinger D, 1993). The method in (Schmidhuber J, and Prelinger D, 1993) uses a similar object function as Equation 1 to match the outputs of the two modules.

The differences between support vector machine and the SIR method. As for a support vector machine, there is only one hyperplane to answer the 'yes/no' question. A multi-class classification task (polychotomy) must be decomposed into a set of simpler two-class classification tasks (dichotomies). Each dichotomy is implemented using one such machine independently. The outputs of these dichotomizes are reconstructed in classification. Advanced techniques have been developed in decomposition of polychotomy into dichotomies and reconstruction of the outputs. The SIR method attempts to simultaneously divide the whole representation space for all classes with multiple hyperplanes (neurons). The SIR method use the internal space of a perceptron and develop it as a single monolithic classifier, where each internal dichotomy (hyperplane) learns in a way dependent of each other. This SIR learning will exhaust the hidden layer space and maximize the utility of all neurons to accomplish highly separated representations in each layer. Such representations have large basins and facilitate the operations of error correction (Chiu G-M, and Raghavendra C S, 1990) (Livingston M, and Stout Q F, 1988). One can follow the SIR method and develop refined representations for patterns layer after layer feedforwardly or train a multilayer network backwardly to obtain such representations. We use the network as an adjustable kernel to transform the patterns to a hypercube space with much separated representations.