An Efficient Learning Scheme for Pattern Classification Using Multi-layer Feed-forward Neural Networks **

Kanad Keeni[†], **Kenji Nakayama**[††], *and* **Hiroshi Shimodaira**[†††],

SUMMARY This study highlights on the subject of weight initialization in multi-layer feed-forward networks. Training data is analyzed and the notion of *critical point* is introduced for determining the initial weights and the number of hidden units. The proposed method has been applied to artificial data, publicly available cancer database and Devanagari characters. The experimental results of artificial data show that the proposed method takes almost 1/3 of the training time required for standard back propagation. In order to verify the effectiveness of the proposed method, standard back propagation where the learning starts with random initial weights has been applied to the cancer database and Devanagari characters. The experimental results indicate the the proposed weight initialization method results in better generalization.

key words: Neural networks, pattern classification, initial weights, decision boundary, critical points

1. Introduction

Neural networks architectures have sparked of great interest in recent years because of their intriguing learning capabilities. Several learning algorithms have been developed for training the networks and out of them Back Propagation [1] is probably most widely used. The reason for the popularity is the underlying simplicity and relative power of the algorithm. Its power derives from the fact that unlike its precursors, the perception learning rule [2], and the Widrow-Hoff learning rule [3], it can be employed for training nonlinear networks of arbitrary connectivity. Since such networks are often required for real-world applications, such a learning procedure is critical. However the standard back-propagation has the following drawbacks.

1. The learning procedure is time consuming. Training always starts from random initial weights and the weight adjustment procedure is very slow

2. It is not clear as to how to construct an appropriate network architecture. The number of hidden units for a particular problem can only be determined by trial and error

In case of 1., it is widely known that the initial weights of the network would largely effect the generalization performance. For example, two networks with same architecture, when trained with totally different initial weights would produce different results. Several researchers have designed systems in which weights are initialized so that the initial activity of the network corresponds to the successive rules, that may come from an expert. Wilson has proposed Fast BPN [4], where the initial parameters are determined by estimating the signal rank with general likelihood ratio test (GLRT) and the singular value decomposition (SVD) of the GLRT covariance matrix. However the disadvantage of their method is that the number of hidden units can not exceed the input feature dimension. In order to tackle 1, most of the researchers have mainly focused on improving the optimization procedure by dynamically adapting the learning rates [5] - [6]. On the other hand it has been shown in [7] that training data selection is also important.

In case of 2., the problem has been treated in various ways. One most common approach has been to start with a large number of hidden units and then prune the network once it is trained [8], [9]. However, pruning does not always improve generalization. Another strategy for finding a minimal architecture has been to add or remove units sequentially [10], [11].

It is also well known that neural networks do not make any assumption about the probability distribution functions of data and can solve complex problems with arbitrary decision boundary. Therefore, it is desirable to investigate the learning characteristics of the networks for finding an estimate about the decision boundary. However, there has been no significant research for finding initial weights and minimal network architecture by exploiting the characteristics of decision boundary.

In the present study, the above mentioned drawbacks of back propagation has been carefully investigated and a method has been proposed for determining the initial weights for input to hidden layer and estimation of hidden units automatically.

This paper is divided into 6 sections. The next section describes the pattern mapping characteristics of

[†]The author is with the Faculty of Information Systems and Quantitative Sciences of Nanzan University

[††]The author is with the Faculty Elec. Eng. and Comp. Science of Kanazawa University

[†††]The author is with the Faculty of Information Science of Japan Advanced Institute of Science and Technology

**A part of this paper was presented at the International Joint Conference on Neural Networks (IJCNN'99) and International Conference on Computational Intelligence and Multi-media Applications (ICCIMA'99)

Multi-layer Neural Networks (MLNN). The third section presents the automatic method for generating initial weights for input to hidden layer connections. The fourth section describes the database. Experimental results of artificial, real world data are provided in the Fifth section. Finally the last section is devoted to conclusion and further researches.

2. Pattern classification characteristics of MLNN

In any pattern classification system, pattern mapping or pattern classification is equivalent to dividing an N dimensional space where the patterns are distributed. In case of multi-layer neural networks (MLNN), this N dimensional space is divided by forming hyper-planes with the help of synaptic weights of nonlinear neurons. MLNNs do not make any assumption about the probability distribution functions of the training data and can solve complex problems with arbitrary decision boundary. The degree of freedom in placing the decision boundary is very high. Therefore, neural networks are considered to be a good choice for pattern classification tasks.

Another most important aspect of neural networks is learn-ability. In case of supervised learning, networks can find optimal synaptic weights through learning. However, since neural networks are nonlinear systems and gradient descent is used to find a set of weights that optimize the performance for a particular task, there is always a possibility of getting stuck in local minima. Therefore, global minima or optimal solution is not always guaranteed. Furthermore, the learning process is time consuming and it is highly dependent on the problem that is to be solved.

If we assume that there are no overlap among the distribution of training patterns, then pattern mapping can be categorized in the following classes.

1. $||\boldsymbol{X}_i - \boldsymbol{X}_j||$ is small $\bigwedge$ $||\boldsymbol{Y}_i - \boldsymbol{Y}_j||$ is small
2. $||\boldsymbol{X}_i - \boldsymbol{X}_j||$ is small $\bigwedge$ $||\boldsymbol{Y}_i - \boldsymbol{Y}_j||$ is large
3. $||\boldsymbol{X}_i - \boldsymbol{X}_j||$ is large $\bigwedge$ $||\boldsymbol{Y}_i - \boldsymbol{Y}_j||$ is small
4. $||\boldsymbol{X}_i - \boldsymbol{X}_j||$ is large $\bigwedge$ $||\boldsymbol{Y}_i - \boldsymbol{Y}_j||$ is large

Here, $\boldsymbol{X}_i$ and $\boldsymbol{X}_j$ belong to class ω_1 and ω_2, $\boldsymbol{Y}_i$ and $\boldsymbol{Y}_j$ are the corresponding output vectors, and $||\cdot||$ stands for the Euclidean norm. In case of 1., the problem is to map similar input vectors in a way such that the corresponding output vectors also become similar. In the second case, the input vectors are similar but they are to be mapped as different patterns in the output space. The third case implies that the input patterns that are far from each other in the input space are to be mapped as similar patterns in the output space. Finally, the 4th case means that the input patterns are far from each other in the input space and they are to be mapped as different patterns in the output space. Now, the pattern mapping of 1., 3., and 4. are not that difficult. However, in case of 2., the problem is to map the patterns that are very close in the input space, as different patterns in the output space. In this case even though the solution exists, due to the difficulty of the problem the training process would be time consuming. Therefore, the second type of pattern mapping results in very slow learning and the possibility of arriving at a local solution is very high.

For example, if we define the connection weight from the i'th input to the j'th hidden unit as w_{ij} then the total input and output of the j'th hidden unit can be defined as follows.

$$net_j = \sum_{i=1}^{n} w_{ij}x_i + \theta_j \tag{1}$$

$$O_j = \sigma(net_j) \tag{2}$$

$$\sigma(net_j) = \frac{1}{1+e^{-net_j}} \tag{3}$$

where, $\sigma(\cdot)$ is the activation function and θ_j is the bias. At the same time the total input to the k'th output unit and the corresponding output can be defined as follows.

$$net_k = \sum_{j=1}^{j} w_{jk}O_i + \theta_k \tag{4}$$

$$O_k = \sigma(net_k) \tag{5}$$

where, $\sigma(\cdot)$ is the same activation function as it was with the hidden layer.

As mentioned earlier the similar patterns play a critical role in learning. Suppose we have training patterns $\boldsymbol{x}_{1n}$ and $\boldsymbol{x}_{2\acute{n}}$ that are very close in the input space and the patterns belong to the class ω_1 and ω_2 respectively. In this case, the network output would become extremely sensitive. This is because the network output must change rapidly for a small change in the input.

Now, if the decision boundary is far from the patterns $\boldsymbol{x}_{1n}$ and $\boldsymbol{x}_{2\acute{n}}$, then the corresponding outputs would have the value $O_{1n} \cong O_{2\acute{n}} \cong 0$ or 1. However, during the learning process, as the decision boundary approaches $\boldsymbol{x}_{1n}$ and $\boldsymbol{x}_{2\acute{n}}$ the output of the corresponding patterns approach the same value, and the learning process becomes extremely slow. In this case, as the decision boundary moves close to the pair $\boldsymbol{x}_{1n}$, $\boldsymbol{x}_{2\acute{n}}$ or enters the region between the pair, the amount of weight correction becomes extremely small. To be specific, if we assume $O_{1n} \cong O_{2\acute{n}} \cong$ some value y then the amount of correction for the n'th pattern Δ_n would be as follows.

$$\Delta_n = \eta \delta_n O_{nj}, \tag{6}$$

$$\delta_n = (t_n - O_n)\acute{f}(net_n), \tag{7}$$

where, O_{nj} is the output of the j'th hidden unit. Now, as the patterns $\boldsymbol{x}_{1n}$ and $\boldsymbol{x}_{2\acute{n}}$ are similar, the output of the jth hidden unit would also become similar, that is

$$O_{1nj} \cong O_{2\acute{n}j}, \tag{8}$$

and

$$\acute{f}(net_{1n}) \cong \acute{f}(net_{2\acute{n}}). \tag{9}$$

In this case, the weight correction will be as follows.

$$\begin{aligned}\Delta_{1n} + \Delta_{2\acute{n}} = \eta O_{1nj}((t_{1n} - O_{1n}) \\ +(t_{2\acute{n}} - O_{2\acute{n}}))\acute{f}(net_{1n}).\end{aligned} \tag{10}$$

If it is assumed that the targets of the patterns are

$$t_{1n} = 1, t_{2\acute{n}} = 0, \tag{11}$$

and the output of the patterns are

$$O_{1n} = z, O_{2\acute{n}} = z - \epsilon, \tag{12}$$

then the weight correction would become as follows.

$$\begin{aligned}\Delta_{1n} + \Delta_{2\acute{n}} &= \eta O_{1nj}((t_{1n} - z) \\ &+(t_{2\acute{n}} - (z - \epsilon)))\acute{f}(net_{1n}) \\ &= ((t_{1n} + t_{2\acute{n}}) - 2z + \epsilon)\acute{f}(net_{1n}) \\ &= (1 - 2z + \epsilon)\acute{f}(net_{1n})\end{aligned} \tag{13}$$

Now at beginning of training, the decision boundary would be far from $\boldsymbol{x}_{1n}$ and $\boldsymbol{x}_{2\acute{n}}$ and in that case the correction of synaptic weights would not be small. However, during the training process, as the decision boundary moves towards $\boldsymbol{x}_{1n}$ and $\boldsymbol{x}_{2\acute{n}}$, because of the similarity of the patterns the output would approach the same value. The most critical situation would take place as z and $||\epsilon||$ approach the value 0.5 and 0 respectively. That is,

$$\begin{aligned}\Delta_{1n} + \Delta_{2\acute{n}} = \\ \lim_{z\to 0.5}\lim_{\epsilon\to 0}(1 - 2z + \epsilon)\acute{f}(net_{1n}) \cong 0\end{aligned} \tag{14}$$

Therefore, the correction of weights for these patterns would become very small and as a result the learning process would become extremely slow.

On the other hand, if the patterns $\boldsymbol{x}_{1n}$ and $\boldsymbol{x}_{2\acute{n}}$ are far from each other in the input space, even if the decision boundary moves towards them the activation of the corresponding outputs would not become the same at the same time. Hence, the weight correction will not become small.

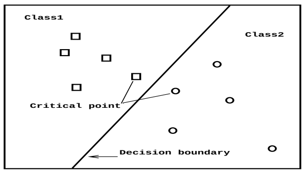


Fig. 1 Critical points and decision boundary

3. Estimation of decision boundary

Here, the decision rule is to select the class corresponding to the output neuron with the largest output. For the sake of simplicity, the number of output unit is set to two (two-class classification problem). However, the concept can be hopefully extended to multi-class classification problems. The decision boundary for a multi-layer feed-forward network is defined as follows.

Definition 1. The decision boundary between two classes in a feed-forward neural networks is the locus of points where both of the output neurons produce the same activation.

If we define the activation output unit i as $O_i(\boldsymbol{x})$ where $\boldsymbol{x}$ is an input vector and let $d(\boldsymbol{x}) = O_1(\boldsymbol{x}) - O_2(x)$, then the decision boundary can be defined as

$$\{\boldsymbol{x}|d(\boldsymbol{x}) = 0\} \tag{15}$$

Next, the notion of *Critical points* is introduced as follows.
Definition 2. The set of critical points contains pairs of data that satisfy the following condition:

$$\min_k(d(\boldsymbol{p}_i, \boldsymbol{q}_k)) = d(\boldsymbol{p}_i, \boldsymbol{q}_j), \tag{16}$$

$$\min_k(d(\boldsymbol{p}_k, \boldsymbol{q}_j)) = d(\boldsymbol{p}_i, \boldsymbol{q}_j) \tag{17}$$

where $d(\boldsymbol{p}_i, \boldsymbol{q}_k)$ denotes the Euclidean distance between the vector $\boldsymbol{p}_i$ and $\boldsymbol{q}_k$.

If we denote the samples in class ω_1 as $\boldsymbol{p}_i$ and samples in class ω_2 as $\boldsymbol{q}_j$ then for each sample in class ω_1 and class ω_2, the set of critical points $\boldsymbol{C}$ can be defined as

$$\begin{aligned}\boldsymbol{C} = \{(\boldsymbol{p}_i, \boldsymbol{q}_j)|\min_k(d(\boldsymbol{p}_i, \boldsymbol{q}_k)) = d(\boldsymbol{p}_i, \boldsymbol{q}_j), \\ \min_k(d(\boldsymbol{p}_k, \boldsymbol{q}_j)) = d(\boldsymbol{p}_i, \boldsymbol{q}_j), \boldsymbol{p}_i \in \omega_1, \boldsymbol{q}_j \in \omega_2\}\end{aligned} \tag{18}$$

A hyper-plane has to be placed in between the pair of critical points. If the coordinate of the pair of critical points $(\boldsymbol{p}_i, \boldsymbol{q}_k)$ are (x_i, y_k) and (u_i, v_k) then the ideal hyper plane must go through the point $\frac{(x_i+u_i)}{2}, \frac{(y_k+v_k)}{2}$ and the slope z of the straight line can be calculated from the following equation.

$$\frac{v_k - y_k}{u_i - x_i} \times z = -1 \tag{19}$$

In the present approach instead of starting from scratch the initial weights for the hidden units connections are calculated from these critical points pairs.

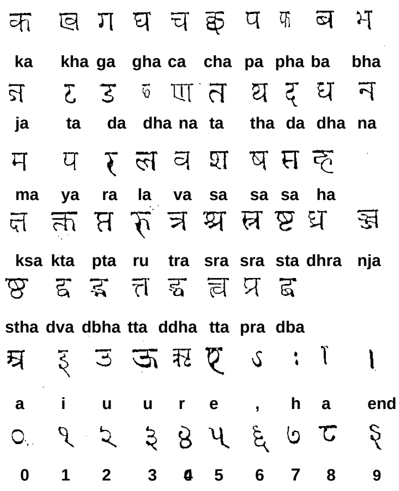
Since, the weight vectors are orthogonal to the separating hyper-plane, the initial weights are generated in the following way. First, the pair of critical points are determined from the training data as mentioned above. Next for all pair of critical points $(\boldsymbol{p}_i, \boldsymbol{q}_k)$ the weight vectors $\boldsymbol{m}_n$ are generated by the following equation:

$$\boldsymbol{m}_n = \frac{\boldsymbol{p}_i - \boldsymbol{q}_k}{||\boldsymbol{p}_i - \boldsymbol{q}_i||} \tag{20}$$

and the biases θ_n are generated by the following equation:

$$\theta_n = -\boldsymbol{m}_n^t \cdot \boldsymbol{P} = -\frac{(\boldsymbol{p}_i - \boldsymbol{q}_k)^t}{||\boldsymbol{p}_i - \boldsymbol{q}_k||} \cdot \frac{(\boldsymbol{p}_i + \boldsymbol{q}_k)}{2} \tag{21}$$

4. Databse



ka kha ga gha ca cha pa pha ba bha
ja ta da dha na ta tha da dha na
ma ya ra la va sa sa sa ha
ksa kta pta ru tra sra sra sta dhra nja
stha dva dbha tta ddha tta pra dba
a i u u r e , h a end
0 1 2 3 4 5 6 7 8 9

Fig. 2 Some of the basic Devanagari characters

The proposed method has been applied to the cancer database obtained from the University of California Machine Learning Repository (two-class classification problem) and Devanagari characters (multi-class classification problem). The cancer database is publicly available and it contains 699 instances each having 9 attributes. The Devanagari database consists of 44 different characters (23 consonants, 13 c-c combinations, 3 vowels, 2 special characters and 3 numerals). Sixty seven Devanagari characters taken from the text [12] are shown in Figure. 2. The characters are stored in the form of 16 × 16 floating point images. A details about the Devanagari databse can be found in [13].

5. Experiments

5.1 Experiments with artificial data

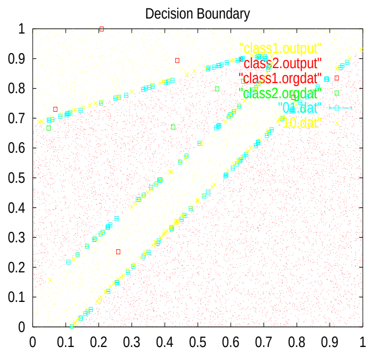


Fig. 3 Decision boundary given by the proposed method

In the present approach the number of hidden units is kept the same as the number of critical points calculated from the training data. Two dimensional data is used for training and testing. Five samples for each class (in this case 2 classes) were randomly generated for training. The network had two input units, two output units and the number of hidden unit was set to 3. Training was continued until the mean square error reach 0.001. For testing, 10000 samples were randomly generated and the class to which the testing sample falls is decided by considering the maximum activation of the output units. The network could learn the training data with 3 hidden units. The decision boundary is estimated from the output activation of the network in respect to the testing samples as follows.

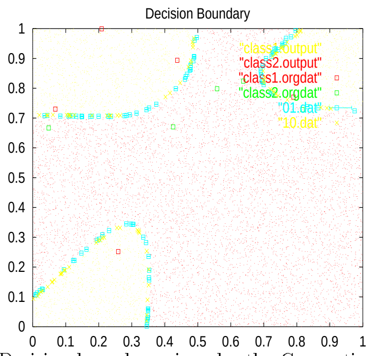


Fig. 4 Decision boundary given by the Conventional Backpropagation

For each testing sample correctly classified as class ω_1, find the nearest testing sample correctly classified as class ω_2. The same process is repeated for the testing samples classified as class ω_2. Now the line connecting the pairs mentioned above must pass through the decision boundary since the pair of samples correctly classified differently. The decision boundary given by the network is shown in Figure 3, where each of the calcuted pair of critical points are connected with a line.

In order to evaluate the effectiveness of the proposed method another set of experiments were performed by employing the conventional back-propagation algorithm and the decision boundary is shown in Figure 4. Next, the network was trained with five different initial weights (weights for hidden to output unit connection) , and the result is summarized in Table 1.

Init weight	Iteration Proposed method	Iteration Standard BP
Seed1	6345	21279
Seed2	6341	21347
Seed3	6251	21263
Seed4	6179	21050
Seed5	6224	22019
Mean	6268	21392

Table 1 Comparison of results

It can be seen in Table 1, that the iterations necessary for the proposed method is less than 1/3 of that of standard back-propagation. In case of standard back-propagation, there is no other way than to cut and try for determining the number of hidden units necessary for solving a problem. In case of the proposed method, the number of hidden unit is determined automatically.

5.2 Two-class classification problem

Experiments were performed by employing the cancer data base. It was divided into ten training and ten test sets and a ten fold cross validation was performed. In order to evaluate the effectiveness of the proposed method another set of experiment was performed by applying the standard back-propagation. The number of hidden unit was set to the number of critical points given by the proposed method. The average accuracy rate of the proposed method, standard back-propagation and the results of applying Bayes decision (assuming normal distribution for each category) rule is shown in Table 2. On the other hand, the following

Method	Accuracy (%)
Proposed	96.8
Standard BP	96.3
Bayes	95.27

Table 2 Average accuracy rate

linear programming methods for pattern recognition: Multi Surface Method [18] (MSM), Robust Linear Programming [19] (RLP), and Perturbed Robust Linear Programming [20] (RLP-P) were also applied on the same dataset and the results are summarized in Table 3.

It is clear from Table 3 that out of the three methods RLPP gives the best accuracy of 96.6 %. Now if compared to Table. 2, it is clear that the proposed method gives slightly better result than RLPP. On the other hand, the superiority of the proposed method over

Method	Accuracy (%)
MSM	92.6
RLP	96.4
RLP-P	96.6

Table 3 Average accuracy rate of linear programming methods

Bayes decision rule suggests that the boundary given by the proposed method is reliable.

5.3 Multi-class Classification problem

Here the proposed method is applied to Devanagari characters for evaluating its applicability to multi-class classification problem. However, a different approach has been taken for representing the output layer. The output layer coding method is thoroughly discussed in [12]. However, for the benefit of the readers a brief description of the method is given in the next subsection.

5.3.1 Automatic coding scheme

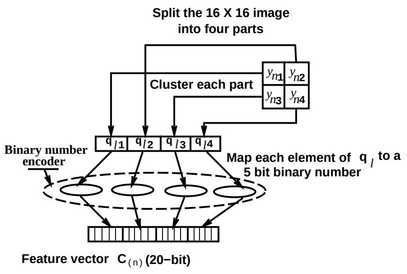

Split the 16 X 16 image into four parts
Cluster each part
y_{n1} y_{n2} y_{n3} y_{n4}
q_{l1} q_{l2} q_{l3} q_{l4}
Binary number encoder
Map each element of q_l to a 5 bit binary number
Feature vector $C_{(n)}$ (20-bit)

Fig. 5 Automatic coding procedure

One of the important aspect of information processing is the way the information is represented. The conventional approach of assigning a category to each cell is reasonable in the sense that, the distance among the codes is constant. However, this kind of approach can not be taken for representing a large number of categories. Although binary representation appears to be reasonable in the sense that it would require $\log_2 n$ bits for representing n categories; they are not well suited for network output representations because of the interdependency of cells which make the learning difficult. For example, if we represent the numbers 7 and 8 in binary as 0 1 1 1 and 1 0 0 0 then irrespective of the fact that the numbers are adjacent integers, 4 output units will have different value. Here, instead of a straight forward binary coding method, a different approach has been taken and the method is as follows. At first each training data $\boldsymbol{x}_n$ $(n = 1, \cdots, N)$ (16 × 16 pi xels) is split into four parts (8 × 8 pixels) in the following way.

$$\boldsymbol{x}_n = (\boldsymbol{y}_{n1}, \boldsymbol{y}_{n2}, \boldsymbol{y}_{n3}, \boldsymbol{y}_{n4})$$

As a result the following set of pattern is generated for each part $i = 1, \cdots, 4$.
$$Y_i = \{\boldsymbol{y}_{1i}, \boldsymbol{y}_{2i}, \cdots, \boldsymbol{y}_{Ni}\}$$

Next, each Y_i $(i = 1, 2, 3, 4)$ is clustered. Here the LBG method [14] was applied for clustering and the number of cluster was set to 16. After clustering, for each ω_l $(l = 1, \cdots, L)$ the cluster to which the part belongs is checked and the final cluster is determined by considering the maximum number of characters belonging to a specific cluster. So, in this way, for each category ω_l there will be a four dimensional vector $\boldsymbol{q}_l = (q_{l1}, q_{l2}, q_{l3}, q_{l4})$, where each element of the vector will correspond to the cluster to which the corre-

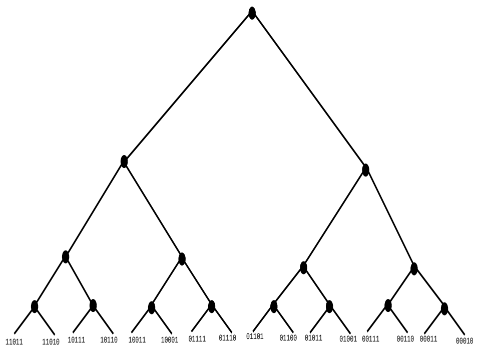
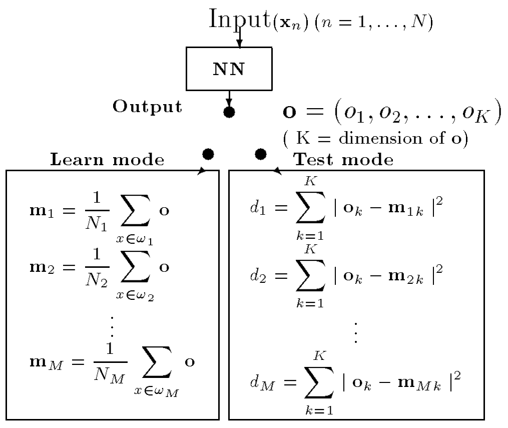


Fig. 6 Code assignment procedure

sponding part of the training sample belongs.Now, each element of vector $\boldsymbol{q}_l$ is mapped to a 5-bit binary pattern. The binary patterns are hand picked and they are mapped in a way such that for any two clusters whose parent is the same, the Euclidean distance between the binary patterns becomes one. This will somehow ensure that the similar part $(\boldsymbol{q}_l)$ of each category ω_l would get similar codes. On the other hand, neglecting the binary patterns containing only 0's or 1's, the minimum number of bits required to maintain this constraint is 5. The code assignment procedure is shown in Figure 6. Finally, $\boldsymbol{q}_l$ is transformed to a feature vector which is a 20-bit code. In this way for each training sample $\boldsymbol{x}_n$ $(n = 1, \cdots, N)$ there will be a feature vector $\boldsymbol{c}_{(n)}$ with 20 elements. (Here (n) represents the category number to which $\boldsymbol{x}_n$ belongs.) The entire process is summarized in Figure 5.

5.3.2 Recognition process

In the present approach, training process is divided into two phases. In the first phase, the network is trained with the training data in a usual way. In the second phase, the training data is fed to the network and the output vectors given by the network are employed for forming the prototypes for each category. The prototype formation process is as follows.



Input$(\mathbf{x}_n)$ $(n = 1, \ldots, N)$
NN
Output
$\mathbf{o} = (o_1, o_2, \ldots, o_K)$
(K = dimension of o)
Learn mode
Test mode
$\mathbf{m}_1 = \frac{1}{N_1}\sum_{x\in\omega_1}\mathbf{o}$
$\mathbf{m}_2 = \frac{1}{N_2}\sum_{x\in\omega_2}\mathbf{o}$
$\mathbf{m}_M = \frac{1}{N_M}\sum_{x\in\omega_M}\mathbf{o}$
$d_1 = \sum_{k=1}^{K} \mid \mathbf{o}_k - \mathbf{m}_{1k} \mid^2$
$d_2 = \sum_{k=1}^{K} \mid \mathbf{o}_k - \mathbf{m}_{2k} \mid^2$
$d_M = \sum_{k=1}^{K} \mid \mathbf{o}_k - \mathbf{m}_{Mk} \mid^2$

Fig. 7 Recognition process
$d_{M_*} = \min d_M \Longrightarrow M_*$: recognized category

$$\mathbf{m_1} = \frac{1}{N_1}\sum_{x\in\omega_1}\mathbf{o}$$
$$\mathbf{m_2} = \frac{1}{N_2}\sum_{x\in\omega_2}\mathbf{o}$$
$$\vdots$$
$$\mathbf{m_M} = \frac{1}{N_M}\sum_{x\in\omega_M}\mathbf{o}$$

Here N and $\mathbf{o}$ stand for the number of samples used for each category during training and the output vector respectively. These prototypes are later used for recognition.

During evaluation, Euclidean distance has been taken among the prototypes formed in the second phase of training and the output vector given by the network during test. The whole process, from formation of prototype vectors to evaluation process is summarized in Figure 7.

Here the target outputs are generated by employing the proposed automatic coding scheme. This solves the problem of output layer representation. The next issues are related to network architecture and initial weights.

Instead of performing trial and error, the number of hidden unit is determined by employing the knowledge of critical points. In this case the straight forward approach of setting the number of hidden unit to the number of calculated critical points pairs could not be applied due to the enormous number of critical points. Therefore, the pair of critical points are sorted based on the distance between each pair. Next, the number of hidden unit is determined by fixing a threshold value d_max over the distance. In this way if d_max set to x results in u hidden units, then first u critical points are employed for calculating the initial weights and biases. This way of selecting the critical points is reasonable in the sense that the patterns that stay close to each other in the input space would largely effect the learning process.

5.3.3 Results

The relation of d_max and the number of hidden units is given in Table. 4.

d_max	2.0	2.2	2.4	2.5	3.0
# Hidden units	35	46	57	61	76

Table 4 Relation of d_{max} and # hidden unit

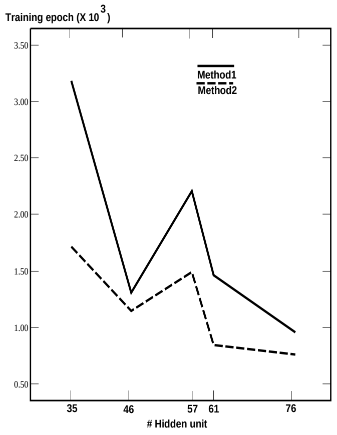


Fig. 8 Training epoch (Method1 Vs Method2)

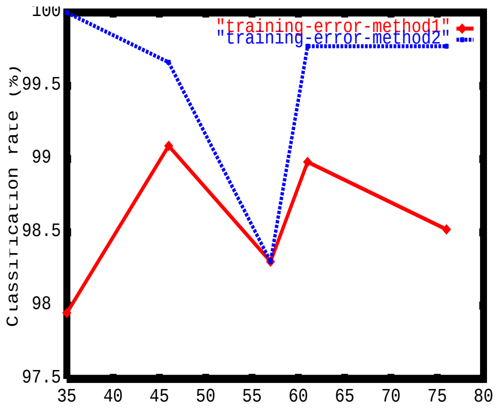


Fig. 9 Classification rate (Training data : Method1 Vs Method2)

The performance of the proposed automatic coding procedure could have been compared to the conventional output layer representation where each of the output unit represents a single category, however this is not considered in the present study. This is because if the real world problems where the number of category to be recognized is too large (for example Kanji) is considered then it is unrealistic to apply the conventional method that would require much more storage and/memory than the proposed method.

In order to evaluate the effect of the proposed method, one set of experiments were performed by employing only automatic coding procedure(Method1), and another set of experiments were performed by employing both automatic coding and automatic weight initialization procedure(Method2). However, in both of the cases, the learning parameters were kept the same. The experimental outcomes are shown in Figure. 8, 9, and 10.

It is clear from Figure. 8, that the number of training epoch required for the combination of the proposed methods is less than the case where only automatic coding procedure is employed.

In Figure 9, 10 it can be seen that the combined method results in better performance against both training and testing data. Now both of Method1 and Method2 gave same performance for training data with 57 hidden units (Figure. 9). However, compared to Method1, Method2 gave better performance with respect to testing data (Figure. 10). This assures that the combined method (Method2) produces better solution. Therefore, the results indicate that the initial weights are effective for faster training and better solution.

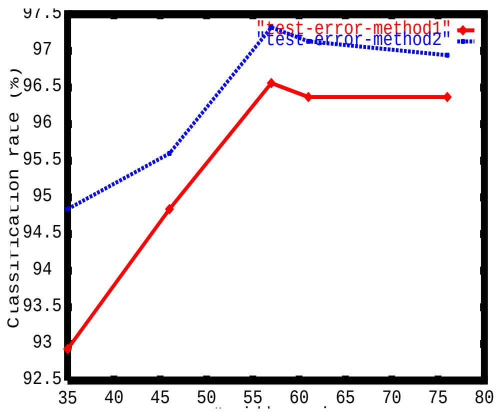


Fig. 10 Classification rate (Testing data : Method1 Vs Method2)

6. Conclusion

It has been successfully shown through experiments that the a priori related to decision boundary can be employed for determining the initial weights of a network. Compared to standard back-propagation the proposed method reduces training time. The method has been successfully applied to the publicly available cancer database and Devanagari characters. The method also determines the number of hidden units automatically.

However, in the present stage the pair of critical points are selected based on a threshold over the distance between the pair of critical points, which can be considered as a local approach. Hence, some other criterion for selecting the pair of critical points in case of complex class boundary is to be further investigated.

Acknowledgement

The author would like to thank the Education Center for Information Processing (ECIP) of Tohoku University for providing the Devanagari database. They are also grateful to Masayuki Kimura, Susumu Horiguchi and Kazunori Kotani of Japan Advanced Institute of Science and Technology for their valuable comments.

References

[1] Rumelhart, McClelland, and the PDP Research Group, "Parallel Distributed Processing," *The MIT Press*, 1989.

[2] Rosenblatt,F : Two Theorems of Statistical Separability in the Perceptron;*Proceedings of a Symposium on the Mechanization of Thought Process, Her Majesty's Stationary Office, London*,1959,421-456.

[3] Widrow, B., and Hoff, M.E: Adaptive Switching Circuits;Institute of Radio Engineers, Western Electronic Show and Convention, Convention Record,part4,1960,96-104.

[4] David H. Kil, Frances B. Shin, "Pattern recognition and Prediction with applications to Signal Characterization," *AIP PRESS*, 1996, pp. 134-138.

[5] Riedmiller, Martin and Braun : RPROP - A fast Adaptive learning algorithm; Technical report *Universitat Karlsruhe*, 1992.

[6] Y. Riedmiller, Martin and Braun : RPROP - A fast Adaptive learning algorithm; Technical report *Universitat Karlsruhe*, 1992.

[7] K. Hara and K. Nakayama: Training Data Selection Method for Neural Networks; *IEICE Trans. of Inf. Syst. Society of Japan, Fundamentals*, Vol.E81-A, No.3, pp. 374-381, March 1998.

[8] M. C. Mozer, P. Smolensky: Skelitonization : A technique for trimming the fat from a network via relevance assessment; in *Advances in neural information processing systems*, 1, pp. 107-115, 1989.

[9] J. Sietsma and Dow : Neural network pruning - why and how;Proceeding of the second international conference on neural networks, pp. 326-333, July 1988.

[10] Fahlman, E. Scott: An empirical study of learning speed in back propagation networks; Technical report *CMU-CS-88-162*, 1988.

[11] T. Ash: Dynamic node creation in back propagation networks; *Connection Science*, 1(4), pp. 365-375, 1989.

[12] H. Kern and Bunyiu Nanjio: *Saddharmapundarika*, ST.-PETERSBOURG, 1912.

[13] K. Keeni, H. Shimodaira, T. Nishino, Y. Tan " Recognition of Devanagari Characters Using Neural Networks," Transaction of Information and Systems Society of Japan, IEICE TRANS.INF. &SYST., VOL. E79-D, No. 5,May 1996.

[14] Y. Linde, A. Buzo, and R. M. Gray : An algorithm for vector quantizer design;*IEEE Trans. Comm., COM-28:84-95,*