TRAINING ALGORITHM OF UNIFORM PARZEN WINDOW NEURAL NETWORKS

Chao-Yin Hsiao and Shan-Hung Hsieh

ABSTRACT

In this article, we propose a uniform Parzen window neural network, and introduce the configuration of the neural network, methods for decision boundary determination, and forward training algorithms. For the neural network, we adopt the uniform distributed Parzen window density function to construct the nodes of the hidden layer, and the union function for the output nodes. We also design a pattern generator algorithm to create artificial pattern data, which can be used for simulation, performance evaluation, and neural network optimization.

Keywords: Uniform Parzen window, neural network, pattern generator.

I. INTRODUCTION

The Parzen window classifiers [3, 4, 6] are powerful tools for data clustering and pattern recognition. The two most popular density functions for a Parzen window classifier are the Gaussian distributed density function and the uniform distributed density function. Recently, Babich and Sibul [3] constructed a neural network based on the weighted Parzen window classifier of the Gaussian type [8, 10, 12, 20]. In this paper, we propose the uniform Parzen window neural network. Also, in sub-section 4-2 of this paper, we give a short comparison of these two types of classifiers.

Classifiers perform better if the decision boundaries are naturally related to the distribution of data in the input space [18]. We can use both the generalization ability and the rate of misclassification to evaluate a pattern classifier [2, 7, 9, 16, 21]. The generalization ability is the recognition rate with respect to the data that is not included in the training data. The rate of misclassification has two types, one fails to classify the members of class A into class A whereas the other classifies the members of class B into class A. For convenience, we call the former type one, and the later type two. In order to strike a balance between the generalization ability and the rate of misclassification, we have to well match the distribution of the sampled training data with that of its parent data, well match the decision boundaries with the distribution of the sampled training data, and allow some tolerance between the decision boundaries and the distribution of the sampled training data. For the former task, we need rich enough, well-distributed sampled training data, and for the last two tasks, some learning methods and some methods for modifying the decision boundaries can be used. Among the learning methods, the parzen window approach [3, 4, 6] together with either the forward training method or the back propagation method are powerful tools. Concerning modification of the decision boundaries, some functions, such as non-linear deterministic functions, fuzzy membership functions, and density distribution functions [1, 5, 13-15, 17, 19], have been attached to the decision boundaries. The attachment of those functions allows the decision boundaries to include some uncertainty information on classification. Any of those decision boundary modification methods can be applied to our neural network. However, the decision boundaries of the uniform Parzen window density function are highly geometrically irregular, so some of the aforementioned methods are not suitable. For this reason, we have modified the decision boundaries by enclosing each Parzen window inside a larger hyperbox.

We have also designed a pattern generator algorithm to create artificial pattern data, which can be used for simulation, performance evaluation, and neural network optimization.

We have organized this paper as follows: in section II, we briefly review the uniform Parzen window density
function. In section III, we present the uniform Parzen window neural network and the design of the training algorithm. In section IV, we give simulation results and make a brief comparison between the Gaussian classifiers and the uniform classifiers. In section V, we make some comments. Finally, in section VI, we draw conclusions.

II. THE PARZEN WINDOW WITH UNIFORM DISTRIBUTED DENSITY FUNCTIONS [3,4,6]

For estimating the data density of a \( d \)-dimensional pattern class \( C_i \) based on chosen samples, we place a region \( R \) of volume \( V \) centered at \( x \) on it. The estimate of the density function at \( x \), \( \hat{p}(x|C_i) \), is defined as

\[
\hat{p}(x) = \frac{k/n_i}{V},
\]

where \( k \) is the number of training patterns from class \( C_i \) falling in the region \( R \) and \( n_i \) is the total number of training patterns from \( C_i \).

Given a set of \( n \) \( d \)-dimensional training vectors \( X = \{x_1, x_2, \ldots, x_n\} \), the estimated probability density function of the Parzen Window [3,4,6] is given by

\[
f_n(x) = \frac{1}{n} \frac{1}{h^d} \sum_{i=1}^{n} \kappa \left[ \frac{x - x_i}{h} \right].
\]

Here, \( \kappa[\cdot] \) is the window function, and \( h \) is the width parameter. It will converge to the true density function if we properly select the width parameter \( h \) and \( \kappa[\cdot] \). The width parameter is required to be a function of \( n \) such that

\[
\lim_{n \to \infty} h^d(n) = 0
\]

and

\[
\lim_{n \to \infty} nh^d(n) = \infty.
\]

The window function is required to be a finite-valued non-negative density function, where

\[
\int_V k(y)dy = 1.
\]

Many window functions are possible. The rectangular and the Gaussian window functions are the most commonly used functions. In the rectangular case, the width parameter \( h \) in eq. (2) is replaced by the vector of \( h_i \), \( i = 1, 2, 3, \ldots, d \), the length of each window side, and the volume \( h^d \) in eqs. (2)-(4) should be replaced by \( \prod_{i=1}^{d} h_i \). In the Gaussian case, we replace the width parameter \( h \) in eq. (2) with the vector of the standard deviations of the Gaussian density function.

III. THE UNIFORM PARZEN WINDOW NEURAL NETWORK

The configurations of the uniform Parzen window neural network, the methods used for decision boundary determination, and the forward training algorithm are introduced in the following.

3.1 The neural network

The architecture of a two-layer uniform Parzen window neural network is shown in Fig. 1 (In the literature, this structure is called a three-layer structure.) The nodes of the hidden layer of the neural network are constructed by the uniform distributed Parzen window density functions. The output nodes are constructed by the union functions used to group together clusters of the same class.

(A) Determination of the node number

The number of input nodes \( N_i \) is equal to the dimension of the parameter space. The number of hidden nodes \( N_h \) is equal to the number of total clusters. The number of output nodes \( N_o \) can either be equal to the number of classes \( m \) for small \( m \), or be determined by \( m \leq 2^{N_h} \) for large \( m \). Each class can contain one cluster or several clusters.

(B) The nodes of the neural network

The function of the nodes of the uniform Parzen window density function is

\[
y_j = f(\bigcup_{i=1}^{N_j} \bigcup_{y=1}^{d} H\left( (x_a - x_{ay}) \leq h_{iy} \right)).
\]

where \( f \) is the nonlinear function used to maintain uncertainty of the decision boundaries, \( \bigcup \) is the union function, \( N_j \) is the number of uniform Parzen windows of the \( j \)-th cluster, and \( H \) is the hardlimit function, \( H(x) = 1 \) whenever \( x \leq 0 \); otherwise \( H(x) = 0 \).

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{Fig_1.png}
\caption{A two-layer uniform Parzen window neural network.}
\end{figure}
3.2 Determination of the decision boundaries of networks

Determining the decision boundaries among clusters is the same as determining the weights between the input nodes and the hidden nodes, while grouping clusters of the same class together is equal to determining the weights between the hidden nodes and the output nodes.

We often have to resolve the problem of overlap in order to determine the decision boundaries, and have to modify the decision boundaries in order to improve the generalization ability. Here, we introduce both the Baysian theorem and the convexity testing method used to resolve the overlapping problem, and use a larger hyperbox to enclose each original hyperbox of each unionized cluster region when the clusters are well separated.

(A) The relationship between the decision boundaries and the weights of the network

We can examine examples of linear perceptron networks and hyper-ellipsoidal networks to see the relationship between the decision boundaries and the weights of the networks.

For a linear perceptron network, the function of the node is

\[ y_j = f(\sum_{i=1}^{d} w_{ij} x_i + \theta_j) \]

(7)

where \( f \) is the nonlinear function used to maintain uncertainty of the decision boundaries. Then, we can see that the weights of the neurons \( w_{ij} \) are slopes of the hyper-plane of the decision function. For hyper-ellipsoidal node, the function of the node is

\[ y_j = f(\sum_{i=1}^{d} w_{ij} x_i + \theta_j) \]

(8)

from which we can see that the weights of the neurons \( w_{ij} \) are the relative length of the principle axes of the hyper-ellipsoidal decision function. This relationship can be extended to the Gaussian nodes because a hyper-ellipsoidal function is equal to the logarithm function composite with a Gaussian function.

Similarly, for a uniform Parzen window node, the function is

\[ y_j = f(\bigcup_{i=1}^{n} \bigcup_{1 \leq j \leq d} H_j (x_i - x_{ij})) \]

(9)

from which, and based on the same aforementioned arguments, we can take the window sides of the uniform Parzen window density function \( h_{ij} \) as the weights of the neurons. Here, \( \cup \) is the union function, \( N_j \) is the number of uniform Parzen windows of the \( j \)-th cluster, and \( H \) is the hardlimit function.

(B) Determination of weights

Neural networks can be trained using the forward training method. During the training stage, we can compare the density function \( f_d(x) \) of the uniform Parzen window in eq. (2) with a preset threshold \( \theta \). If \( f_d(x) \geq \theta \), we accept the uniform Parzen window; otherwise, we reject it. We use \( f_d(x) \) and \( \theta \) to determine the decision boundaries.

(C) Modification of decision boundaries

For the purpose of keeping the uncertainty information of classification, many decision boundary modification methods have been proposed. The most common methods assign some non-linear deterministic functions, some fuzzy membership functions, or some density distribution functions to each node of a neural network.

Any of those decision boundary modification methods can be applied to our neural networks. However, the decision boundaries of the uniform Parzen window density function are highly irregular geometrics. In order to well match the decision boundaries with the complex geometry learned from the sampled data distribution, and to improve the generalization ability, we form the decision boundaries of the uniform Parzen window density function are highly irregular geometrics.

Any of those decision boundary modification methods can be applied to our neural networks. However, the decision boundaries of the uniform Parzen window density function are highly irregular geometrics.

In cases of cluster region overlap, we can apply the Baysian theorem on the density distribution function of the cluster data to determine the decision boundaries.

For example, suppose we have two classes of pattern data,

\[ w_1 = \{ x_{11}, x_{12}, x_{13}, \ldots, x_{m_1} \} \]

(10)

\[ w_2 = \{ x_{21}, x_{22}, x_{23}, \ldots, x_{m_2} \} \]

According to the Bayes theorem,

\[ x \in w_j, \text{if } p(x \mid w_j)P(w_j) > p(x \mid w_k)P(w_k), \quad j, k = 1, 2 \]

(11)

The decision boundary can then be determined as:

\[ d_{ij} = p(x \mid \omega_j) - p(x \mid \omega_i) = 0 \]

(12)

(E) Resolving the overlapping problem using convexity testing [7]

The method of convexity testing is an alternative and efficient way to solve the overlap problem and determin
the decision boundaries. We can construct the training algorithm by using both the convexity testing method and the Bayesian theorem together. The details of the convexity testing method are introduced in the following.

(i) The Level of the probability density function

Let $M$ be the maximum value of the p.d.f. estimation on set $S$. $L$ levels are selected between 0 and $M$. The level at each point $x$, is an integer value defined as

$$\text{level}(x) = \text{INT}\left\{ L \times \frac{p.d.f.(x)}{M} \right\}$$

(with $1 \leq r \leq p$). (13)

(ii) Convexity testing

Refer to Fig. 2; assume that a non-empty pixel $H_1(x_r)$ is contained inside $H_3(x_r)$. The estimated mean values of $X_r$ in two domains $H_1(X_r)$ and $H_3(X_r)$ are given by $\rho[H_1(X_r)]$ and $\rho[H_3(X_r)]$, respectively. If

$$\rho[H_1(X_r)] > \rho[H_3(X_r)],$$

then the sampling point $X_r$ and the associated pixel $H_1(X_r)$ are defined as being “concave”; otherwise, they are convex.

The concept behind this convexity testing method can be explained using the example shown in Fig. 3. Figure 3(a) shows that we have three clusters in a 2-D parameter space. Figure 3(b) shows the testing result of a density level test at density level $t = 5$. This result is very close to the original distribution, but some pixels in different clusters are connected, and some pixels are isolated. On the other hand, Fig. 3(c) shows the testing result at density level $t + 1 = 6$; this result shows 3 well separated clusters as expected, but the distribution of each cluster is different from the original distribution to some degree. Neither of them are satisfactory. Now, if we adopt the convexity test, at the density level $t$, we keep the concave pixels and delete the convex pixels. The result is shown in Fig. 3(d). In this case, we not only can well identify the original three
clusters, but also can make the distribution similar to the original distribution.

(iii) The algorithm of convexity testing

1. Set the initial density level.
2. If all the clusters are separable and this is level one, then stop the test.
3. If any two of the clusters are un-separable and this is not the top level, increase the density level by one. Repeat this test until all clusters are separable or reach the top level.
4. If any two of the clusters are un-separable and this is at the top level, then separate the clusters based on the Baysian rule, and stop the test.
5. If all the clusters are separable and this is not level one, decrease the density level by one. Repeat this stage until some clusters are un-separable or level one is reached. If level one is reached, then go to step 2 and stop. Otherwise, if some clusters are un-separable, then go to step 6.
6. Perform convexity testing to remove the convex pixels. If all the clusters are separable, then stop the test. If some clusters are still un-separable, recover the removed convex pixels, increase the level by one, and stop the test.

3.3 The pattern generator

Similar to Monte Carlo simulation, we can use some artificial pattern data to compare the performance of several neural networks.

The pattern generator is an algorithm that is used to create the artificial pattern data with specified fundamental patterns and generalization levels. The specified fundamental patterns are used to mimic the shape of the pattern data of a specified cluster. The specified generalization level is used to represent the uncertainty level of the pattern data distribution of the specified cluster. Here, we use the variance of a normal distribution function to represent this distribution uncertainty.

The block diagram of the pattern generator is shown in Fig. 4, in which $F.P.$ is a specified fundamental pattern, and each element of the specified $F.P.$ is called a fundamental element. $G.L.$ is the generalization levels, and $P.E.$ is a created pattern element. (A point in the parameter space.)

For example, assume that $m$ is a fundamental point (vector) in a pattern space, and that $(G.L. = A; \text{then}, \text{each created pattern element} \ x = N(m, A)$, that is, $x$ is a random vector of normal distribution with mean $m$ and covariance matrix $A$. $A$ is a diagonal matrix with elements $a_i$, some positive real numbers.

The algorithm of pattern generator:

1. Specify a fundamental pattern in the parameter space and set the generalization level to $G.L. = A$.
2. For each point $m$ in the fundamental pattern, run a random signal generator to get some pattern elements, $x = N(m, A)$.
3. Repeat step 2 again and again for all the elements of the fundamental pattern.

3.4 The training algorithms

The neural network can be trained using either the forward method or back propagation method. Here, the forward one is introduced. It includes decision boundary determination, decision boundary modification, overlap resolution, union node attachment, and final optimization.

The supervised forward training algorithm:

1. Set the number of classes $m$, the number of clusters $n_i$, $i = 1, 2, ..., m$, the window width $h$, and the density threshold $\theta$.
2. Learn the density distribution function of each cluster by using the uniform Parzen window.
3. Perform the union operation on the enclosed cluster region of the same class, and determine the union nodes of the output layer.
4. Modify the decision boundaries by enlarging the window after the union nodes of the output layer.
5. Optimize the neural network.

Here, the optimization procedure can be conducted by using the data created by a pattern generator to finely tune the decision boundaries.

IV. SIMULATION AND COMPARISON

4.1 Simulations

We used the pattern generator to create the sampled data (for training the neural networks) and the testing data (for testing the trained neural network) for the following three examples.

The first example is used to show the classification capability in 2-dimensional space. The 2nd example shows the classification capability when the pattern data is distributed in spiral shapes. The 3rd example shows the classification capability in higher dimensional space, a 6-dimensional space example is conducted. For comparison, a two-layer perceptron neural network as well as our uniform Parzen window neural network was employed in all three examples. The simulation results are repre-
sented by + and *, respectively. Figure 5 shows the first example, and Fig. 5(a) shows the pattern distribution of four pattern classes in eight pattern clusters. Figure 5(b) shows the related pattern density distribution with softened boundaries. Figure 5(c) shows the misclassification rates for different generalization levels. Figure 6 shows the 2nd example, and Fig. 6(a) shows the pattern distribution of three spiral pattern classes. Figure 6(b) shows the related pattern density distribution with modified boundaries. Figure 6(c) shows the misclassification rates. Table 1 shows the misclassification rates of the 3rd example.

4.2 Comparison between Gaussian classifiers and uniform Parzen classifiers

Figure 7 shows four trained pattern clusters in a two-dimensional parameter space. Fig. 7(a) shows those classified using 4 Gaussian classifiers while Fig. 7(b) shows those classified using 4 uniform Parzen classifiers. For any tested pattern located in the space outside the 4 boundaries in Fig. 7(b), it will not be classified into any one of the 4 clusters using the uniform Parzen classifiers, but it will be forced into one of the 4 clusters by the Gaussian classifiers. The same situation will happen to linear perceptron classifiers such as that of sub-section 4.1. More classifiers can be used to solve this problem, but this will make the networks much more complex and will greatly increase the computation requirement.

Refer to Fig. 8; if the geometric shape of the data distribution near the decision boundaries is similar to that of Fig. 8(a), it will be proper to adopt the Gaussian classifiers. On the other hand, if the geometric shape of the data distribution near the decision boundaries is similar to that

Fig. 5. Pattern classification of the four/eight patterns. (a) The pattern distribution of four pattern; (b) The uniform Parzen window density classes in eight pattern clusters distribution of the four/eight patterns, with modified decision boundaries; (c) Misclassification rate comparison of the four/eight patterns.
of Fig. 8(b), then it is better to choose the uniform Parzen classifiers. However, in practice, in most cases, the shape of the data distribution near the decision boundaries can not be predicted, especially in high dimensional cases.

Usually, the boundaries of a uniform Parzen window can be some constant values, such as $x_{\text{max}}, x_{\text{min}}, y_{\text{max}}, y_{\text{min}}$, etc., so that the pattern classification procedure is used to make a comparison between the parameter coordinates of the tested patterns and these constants. While for Gaussian classifiers, the comparison is with the decision functions.

In general, the Gaussian classifiers can use fewer nodes and have much smoother decision boundaries. This might be a drawback of uniform classifiers.

V. DISCUSSION

When the classifiers are embedded inside the neural network, the Parzen window neural network is still a classifier that has the capabilities of a neural network, such as the capability of parallel processing. In this study, a two-layer uniform Parzen window neural network has been studied, and the same concept can be extended to multi-layers structures.

The uniform Parzen window neural network adopts hyperbox elementary cluster regions for the probability density functions. It requires more training time but keeps the information of the density distribution of the clusters. This information is useful for determining the decision boundaries, and it can also be very useful in un-supervised learning.

Finding a good match between the decision boundaries and the distribution of the parent data is essential for all classifiers. The uniform Parzen Window classifier leaves all the hyper-space outside the enclosed regions

![Fig. 6. Pattern classification of three spiral pattern classes. (a) The pattern distribution of three spiral pattern classes; (b) The uniform Parzen window density distribution of the three spiral patterns with modified decision boundaries; (c) Misclassification rate comparison of the three spiral patterns at different generalization levels.](image)
Table 1. Misclassification rate comparison of the six-dimensional patterns at different generalization levels.

<table>
<thead>
<tr>
<th>GL</th>
<th>mean vectors</th>
<th>No. samples of cluster</th>
<th>misclassification rate (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>class1</td>
<td>class2</td>
<td></td>
</tr>
<tr>
<td>1.0</td>
<td>( x_1 = [0\ 0\ 0\ 4\ 4\ 4] ) ( x_2 = [4\ 4\ 4\ 4\ 4\ 4] )</td>
<td>( x_1 = [0\ 4\ 4\ 4\ 4\ 4] ) ( x_2 = [0\ 0\ 0\ 0] )</td>
<td>20</td>
</tr>
<tr>
<td>1.5</td>
<td>( x_1 = [0\ 0\ 0\ 4\ 4\ 4] ) ( x_2 = [4\ 4\ 4\ 4\ 4\ 4] )</td>
<td>( x_1 = [0\ 4\ 4\ 4\ 4\ 4] ) ( x_2 = [0\ 0\ 0\ 0] )</td>
<td>20</td>
</tr>
<tr>
<td>2.0</td>
<td>( x_1 = [0\ 0\ 0\ 4\ 4\ 4] ) ( x_2 = [4\ 4\ 4\ 4\ 4\ 4] )</td>
<td>( x_1 = [0\ 4\ 4\ 4\ 4\ 4] ) ( x_2 = [0\ 0\ 0\ 0] )</td>
<td>20</td>
</tr>
<tr>
<td>2.5</td>
<td>( x_1 = [0\ 0\ 0\ 4\ 4\ 4] ) ( x_2 = [4\ 4\ 4\ 4\ 4\ 4] )</td>
<td>( x_1 = [0\ 4\ 4\ 4\ 4\ 4] ) ( x_2 = [0\ 0\ 0\ 0] )</td>
<td>20</td>
</tr>
<tr>
<td>3.0</td>
<td>( x_1 = [0\ 0\ 0\ 4\ 4\ 4] ) ( x_2 = [4\ 4\ 4\ 4\ 4\ 4] )</td>
<td>( x_1 = [0\ 4\ 4\ 4\ 4\ 4] ) ( x_2 = [0\ 0\ 0\ 0] )</td>
<td>20</td>
</tr>
</tbody>
</table>

Fig. 7. Possible misclassification of tested patterns. (a) The Gaussian classifiers; (b) The Uniform classifiers.
empty, and this can greatly reduce the rate of type two misclassification with little sacrifice in generalization ability. On the other hand, to improve the generalization ability and greatly reduce the type one misclassification error, we modify the decision boundaries by means of the following strategy: For well separated pattern classes, we adopt a larger window size to improve the generalization ability while for pattern classes which are not well separated, we can modify the decision boundaries so as to resolve the boundary determination problem.

As for the training algorithm, we adopt the forward training method. It can save much training time and has less difficulty with convergency especially in the case of cluster overlap, as compared to the backpropagation method. On the other hand, the backpropagation method can be employed on-line and has more flexibility.

VI. CONCLUSIONS

In this article, we have adopted the uniform distributed Parzen window density functions to construct a two-layer neural network and have constructed the forward training algorithm.

We have also proposed a pattern generator algorithm that can create pattern data with specified generalization levels. These data can be used for simulations, performance evaluation, and neural network optimization. They also can be used to compare the performance of different neural networks.

This approach can keep the advantages of both the Parzen window and the neural network. The Parzen window can learn the data distribution directly, which will make the decision boundaries properly match the cluster distribution for pattern recognition. On the other hand, the neural network has the capabilities of parallel data processing, fault tolerance, and fast on-line decision making. The proposed neural network can enclose the sampled data properly, with properly selected margins, has less difficulty with convergence in training, and can greatly reduce the rate of misclassification with little sacrifice in generalization ability. Simulations results obtained well support this conclusion.

REFERENCES

7. Jain, A.K. and R.C. Dubes, Algorithm for Clustering

Chao-Yin Hsiao was born in Taiwan in 1952. He received his B.S. degree in Mechanical Engineering from Feng-Chia University, in 1974, the M.S. degree from the Power Mechanical Engineering Department, National Tsing Hua University, Taiwan, R.O.C. in 1976, and the Ph.D. degree in the MANE Department, UCLA, California, U.S.A. in 1986. From 1986, He has served as an associate professor in the Department of Mechanical Engineering, Feng-Chia University. During 1987-1992 he was appointed to be the chairman of the departement, and during 1992-1995 he was appointed to be the director of the Research and Development Cooperative Committee by Feng-Chia University. His research interests include the filtering techniques, control law design, pattern recognition, and techniques of fuzzy and neural networks.

Shan-Hung Hsieh was born in Taiwan in 1974. He received the B.S. and the M.S. degrees from the Department of Mechanical Engineering, Feng-Chia University, Taichung, Taiwan, R.O.C. in 1996 and 1998 respectively. His research interests include the filtering techniques, control law design, pattern recognition, and techniques of fuzzy and neural networks.