Function Approximation Using LVQ and Fuzzy Sets

Shon Min-Kyu, Junichi Murata, Kotaro Hirasawa

Department of Electrical and Electronic Systems Engineering
Graduate School of Information Science and Electrical Engineering
Kyushu University, 6-10-1 Hakozaki, Higashi-ku, Fukuoka 812-8581, Japan

Abstract

Neural networks with local activation functions, for example RBFNs (Radial Basis Function Networks), have a merit of excellent generalization abilities. When this type of network is used in function approximation, it is very important to determine the proper division of the input space into local regions to each of which a local activation function is assigned. In RBFNs, this is equivalent to determination of the locations and the numbers of its RBFs, which is generally done based on the distribution of input data. But, in function approximation, the output information (the value of the function to be approximated) must be considered in determination of the local regions. A new method is proposed that uses LVQ network to approximate the functions based on the output information. It divides the input space into regions with a prototype vector at the center of each region. The ordinary LVQ, however, outputs discrete values only, and therefore can not approximate continuous functions. In this paper, fuzzy sets are employed in both of learning and output calculation. Finally, the proposed method uses the back-propagation algorithm for fine adjustment. An example is provided to show the effectiveness of the proposed method.

1 Introduction

Neural networks with local activation functions, for example RBFNs(Radial Basis Function Networks), have a merit of excellent generalization abilities. So they have been widely used in function approximation. When this type of network is used in function approximation, it is very important to determine the proper division of the input space into local regions to each of which a local activation function is assigned. In RBFNs, this is equivalent to determination of the locations and the numbers of its RBFs, which is generally done based on the distribution of input data by using, for example, clustering technique; more number of RBFs are placed where the input data are dense while the areas where the input data are sparsely distributed have fewer RBFs. Then the network weights are adjusted to minimize the approximation errors. So, the local regions are determined by the input information, and the output information is used only for the later adjustment. However, in function approximation, the above procedure does not work well. Consider a set of input data which are uniformly distributed. Then, the above procedure will give a uniformly divided input space. But, in the areas where the value of the function to be approximated changes violently, there should be a lot of small local regions (a lot of RBFs) to obtain good approximation, while in the areas where the change of the function values is very small, fewer regions (fewer RBFs) will suffice. So, the output information (the value of the function to be approximated) must be considered in determination of the local regions.

Here, a new method is proposed that uses LVQ (Learning Vector Quantization) network to approximate the functions. LVQ also has a characteristic of local activation functions. Each output node is given a prototype label which, used in function approximation, is its associated output value. For each output node, through LVQ learning, an appropriate local region is found where the output is approximated by the label value. Thus the input space division is done based on the output information. The ordinary LVQ, however, outputs discrete values only, and therefore can not approximate real-valued func-

0-7803-7087-2/01/$10.00

tions. In this paper, fuzzy sets are employed in both of learning and output calculation. In the learning phase, unlike the ordinary LVQ, the true output value may not completely match any of the prototype labels because the true output can take any real value while there are only a finite number of prototypes. This possible error between the true value and the winner's label introduces fuzziness in the judgment of whether the winner is correct or not. A fuzzy set is assigned to each node to cope with this fuzziness. To calculate the network output for the input points in between the prototypes, interpolation is necessary and is done using another fuzzy set. Finally, the proposed method uses the back-propagation algorithm for fine adjustment.

An example is provided to show the effectiveness of the proposed method.

2 LVQ Networks and their difficulties in function approximation

Kohonen's LVQ network is a supervised learning algorithm associated with the competitive network shown in Fig.1. The network consists of an input layer and an output layer. A weight vector $\boldsymbol{w}_i$ is associated to the i-th node in the output layer. In learning phase, the LVQ network selects a weight vector closest to a given input vector and then compares the output of LVQ network with the output of training data. If they match, the selected weight vector is updated so that it approaches the input vector. Otherwise, the selected weight vector is updated so that it moves away from the input vector. After the learning, the LVQ network chooses the nearest weight vector to a given input vector, and outputs its 'label' as the network output. Thus, a weight vector can be regarded as the center of a local region in the input space.

In function approximation problems, a certain fixed real value must be assigned to each weight vector as its 'label'. So, if there are a sufficient number of weight vectors with various 'label' values, they found their proper locations by their learning. This realizes the division of the input space based on the output values of the training data.

However, when the ordinary LVQ algorithm is used in function approximation, it has some problems. Let us consider an example function shown in Fig.2(a). The LVQ training algorithm drives a weight vector closer to a training input vector when the 'label' value of the weight vector matches the training output value. But, there are only a finite number of 'labels' while the training output data can take any real number. So, there are an infinite number of data that do not match any weight vector 'label', and thus the LVQ learning algorithm in its original form does not work well in function approximation problems. Another problem occurs when the output is calculated after learning. The LVQ network output can take a value among the finite set of 'label' values only. So, the approximation of the training data shown in Fig.2(a) by the LVQ network is at best a piecewise step function as shown in Fig.2(b).

3 LVQ network and Fuzzy Sets

3.1 Learning of LVQ network

In the learning phase, to overcome the above mentioned problem, the proposed LVQ network uses fuzzy sets. The fuzzy sets calculate how much a training output value and a 'label' value of a weight vector match. The weight vector is updated by the degree of matching. Therefore, the weight vector is trained even if the matching is not complete. A fuzzy 'label' is assigned to each weight vector. Fig.3 shows the typical membership functions for the fuzzy 'labels'. In the figure, on the interval of possible output value [MIN, MAX], five fuzzy 'labels' are defined,

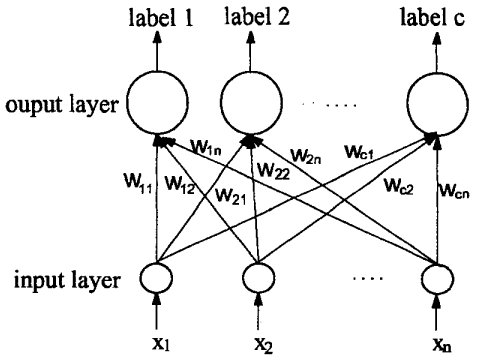

Fig. 1: Structure of LVQ network.

where MIN(=$n0$), $n1$, $n2$, $n3$ and MAX(=$n4$) are representative values of the fuzzy 'labels'. These membership functions are prepared from the output information on a training data set before performing training. For each 'label' n_i, r weight vectors are assigned. In other words, there are r weight vectors that have the same 'label' n_i. The weight values are initiallized appropriately. The weight vector updating is done based on the degree of matching of the 'label' and the training output value which is measured by the membership function. Fig.4 illustrates the idea. Suppose that the k-th training output value is $y(k)$ and that the weight vector with the 'label' $n2$ is nearest to the training input. Only this weight vector is updated. The degree of matching is calculated as $a(k)$ by the membership function. This degree $a(k)$ determines how close the weight vector can approach the current training input vector. The amount of updating a weight Δw_{ij} for this k-th training pair is expressed by equations (1), (2) and (3),

$$\Delta w_{ij}^k = \eta \cdot M(a(k)) \cdot (x_j(k) - w_{ij}), \quad (1)$$

$$M(a(k)) = E \cdot a(k), \quad (2)$$

$$E = (MAX - MIN)/u. \quad (3)$$

Here u is total number of labels n_i. When $a(k)$ is 0, $\Delta w_{ij}^k = 0$. This is repeated for all the training data k=1,2,3,$\cdots$,l. Then, updating of actual weights is performed by equation (4),

$$w_{ij(new)} = w_{ij(old)} + \frac{\sum_{k=1}^{l}(\Delta w_{ij}^k)}{m_i}, \quad (4)$$

Where m_i is the number of non-zero Δw_{ij}^k (k=1,2,$\cdots$,l). When the weight vector converges, the sum of Δw_{ij} over k=1,2,$\cdots$,l, becomes zero.

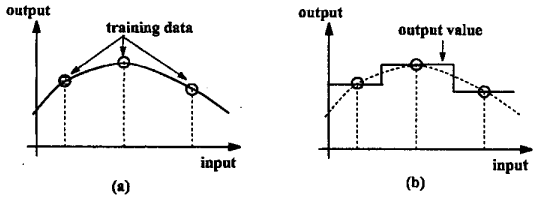

Fig. 2: Difficulties with LVQ algorithm in function approximation.

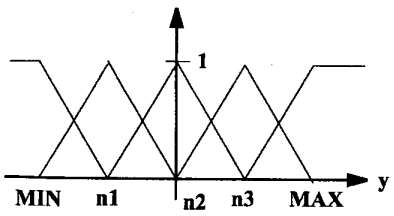

Fig. 3: Membership functions.

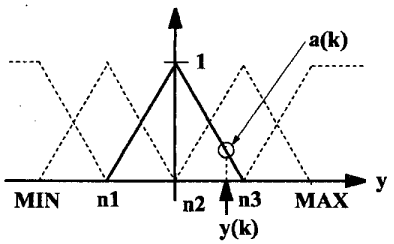

Fig. 4: Use of membership functions in learning.

This gives the limit value of the weight as

$$w_{ij(new)} = \frac{\sum_{k=1}^{l}(M(a(k)) \cdot x_j(k))}{\sum_{k=1}^{l}(M(a(k))}. \quad (5)$$

This imposes the limit on how close the weight vector can approach the input vector, and the limit depends on the degree of matching $a(k)$.

3.2 Output of LVQ Network

The ordinary LVQ network outputs discrete values only, and therefore can not approximate a continuous function(Fig.2(b)). In this paper, using fuzzy algorithm for output layer, the LVQ network can output continuous values and therefore approximate continuous functions.

For a given input vector, the LVQ network selects $p+1$ nearest weights to the input, and smooths their representative 'label' values using fuzzy sets defined on the input space. The equation of fuzzy algorithm is shown by equation (6),

$$y = \frac{\sum_{i=1}^{p+1}(\frac{1}{||\boldsymbol{w}_i - \boldsymbol{x}||} \cdot n_i)}{\sum_{i=1}^{p+1}\frac{1}{||\boldsymbol{w}_i - \boldsymbol{x}||}}, \quad (6)$$

where n_i is the representative value of 'label' of node i, and p is the dimension of input $\boldsymbol{x}$. The

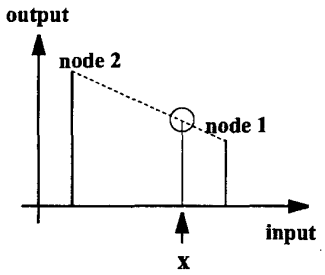

Fig. 5: Output value among nodes.

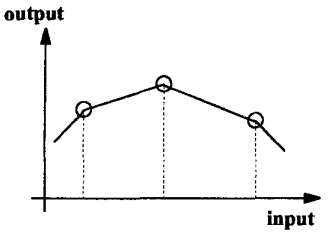

Fig. 6: Output value using fuzzy algorithm.

procedure is illustrated by Fig.5. The fuzzy algorithm outputs the curve shown in Fig.6 which is in contrast to Fig.2(b) obtained by the ordinary LVQ network.

4 Re-learning of weight vectors and Back-Propagation

In the learning, not all the weight vectors are updated. It is because some weight vectors locate far away from their matching(in a fuzzy way) training data in the input space. In order to obtain the best result, it is preferable to re-train those weight vectors that have not been trained.

Let S denote the number of untrained weight vectors in the first learning. Now, their labels are re-assigned. If many of the weight vectors corresponding to a label n_i were updated in the first learning, this implies that these weight vectors are necessary to approximate the given function and suggests that use of more weight vectors corresponding to this label n_i will be benificial. Reassignment of labels of untrained weight vectors is done based on this idea. LVQ network investigates the number of the trained weight vectors t_i for each label n_i after the first learning. Then each label n_i is sorted in ascending order of t_i. This order is denoted by j_i, i=1,2,$\cdots$,u. The number r_i of re-assiged weight vectors to label n_i, is determined by j_i as follows:

$$r_i = j_i \cdot \beta. \tag{7}$$

Since the total number of untrained weight vectors is S, the positive constant β must satisfy the following:

$$S = 1 \cdot \beta + 2 \cdot \beta +, \cdots, +u \cdot \beta. \tag{8}$$

This gives

$$\beta = \frac{2 \cdot S}{u \cdot (u+1)}. \tag{9}$$

To assign an integer value to each r_i, equation (7) is slightly changed as

$$r_i = \begin{cases} j_i \cdot [\beta] & ,\text{if } 1 \leq j_i \leq (u-1), \\ S - \sum_{j_i=1}^{u-1} (j_i \cdot [\beta]) & ,\text{if } j_i = u, \end{cases} \tag{10}$$

where $[\beta]$ is the largest integer smaller than β. Therefore, a large t_i gives a large r_i. Re-learning is carried out like the first learning for the unupdated weight vectors. So, the nearest weight vector to the training vector is chosen among those weight vectors only. This procedure can produce much better result by giving more weight vectors to the labels which were well used in the first learning.

Finally, the 'labels' of trained weight vectors are updated by back-propagation algorithm for fine adjustment after re-learning.

5 Example

To study the performance of the proposed method, it is applied to a function approximation problem where the target function is shown as Fig.7. Fig.7 shows the 256 training data. The representative values are generated by $(MAX + MIN)/10$ and are assigned to the output nodes of LVQ network. The LVQ network consists of 250 output nodes. Fig.8, 9 and

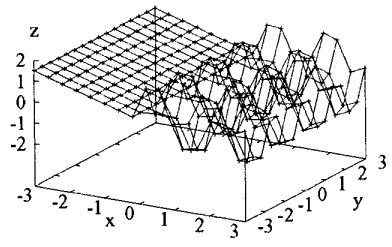

Fig. 7: Training data.

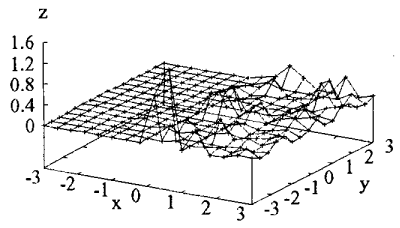

Fig. 9: Error after re-learning.

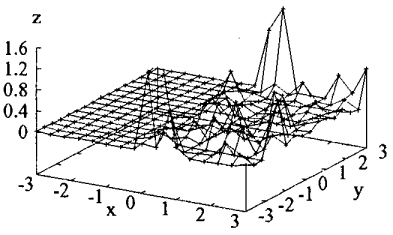

Fig. 8: Error after first learning.

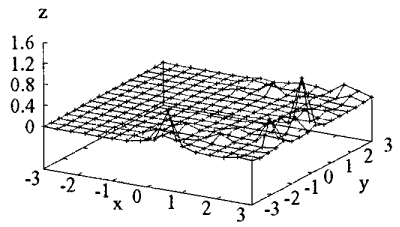

Fig. 10: Error after BP training.

10 show the error after each learning stage. The error decreases as the learning stage proceeds. This confirms the usefulness of re-learning and BP adjustment. Fig.11 shows the distribution of trained weights. We can easily find that more weights are located where the change of function value is larger in Fig.11. Result of function approximation using the proposed LVQ algorithm with re-learning and BP adjustment is shown in Fig.12, which approximates the target function well.

6 Discussions and Conclusions

This paper presents a method for approximation of functions using LVQ algorithm and fuzzy sets. More specifically, this paper investigates input space division by using the output information in addition to input information of training data. The effectiveness of the proposed method is verified by an example. In this method, one weight vector is updated for a given training data vector. But one data vector may contain useful information for updating more than one weight. If all of the weight vectors which correspond to a training data vector are trained, more efficient function approximation will be realized. Since the fuzzy membership functions for the output layer are straight lines, the approximated functions are not smooth. Using the fuzzy sets of other shape can approximate a lot of functions smoothly.

References

[1] Kohonen,T, "Self-Organizing Maps", *Springer Series in Information, Sciences*, 1995.

[2] Kohonen,T, "Learning Vector Quantization

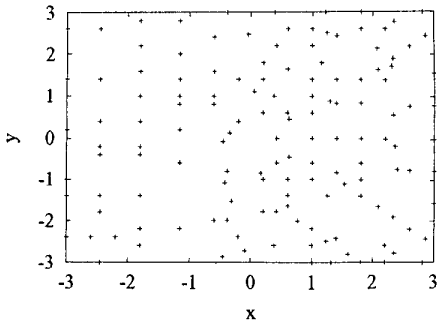

Fig. 11: Weight distribution after learning.

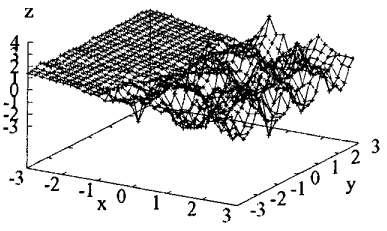

Fig. 12: Result of learning of data.

for Pattern Recongition", *Technical Report TKK-F-A602, Helsinki University of Technology*,Finland,1990.

[3] S.Chen, C.F.N.Cowan, and P.M.Grant, "Orthogonal Least Squares Learning Algorithm for Radial Basis Function Networks", *IEEE Trans. Neural Network, vol.l2*, 1991, 302-309.

[4] R. Katayama, Y.Kajitani, K.Kuwata, Y.Nishida, "Self generation radial basis function as neuro-fuzzy model and its application to nonlinear prediction of chaostic time series", *2nd Int. Conf on Fuzzy Systems (FUZZY-IEEE'93)*, 1993, 407-414.