Strict Generalization in Multilayered Perceptron Networks

Debrup Chakraborty[1] and Nikhil R. Pal[2]

[1] Computer Science Department, CINVESTAV-IPN
Av. IPN No. 2508, Col. San Pedro Zacatenco
Mexico, D.F. 07360, MEXICO
[2] Electronics and Communication Sciences Unit
Indian Statistical Institute, 203 B.T. Road
Calcutta 700108, INDIA

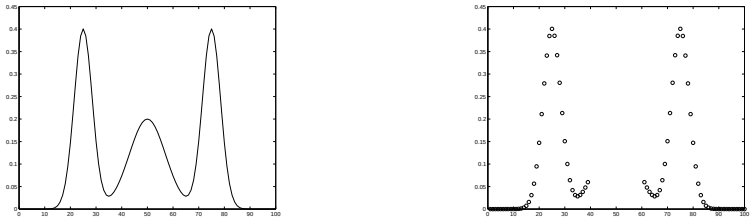
Abstract. Typically the response of a multilayered perceptron (MLP) network on points which are far away from the boundary of its training data is not very reliable. When test data points are far away from the boundary of its training data, the network should not make any decision on these points. We propose a training scheme for MLPs which tries to achieve this. Our methodology trains a composite network consisting of two subnetworks : a mapping network and a vigilance network. The mapping network learns the usual input-output relation present in the data and the vigilance network learns a decision boundary and decides on which points the mapping network should respond. Though here we propose the methodology for multilayered perceptrons, the philosophy is quite general and can be used with other learning machines also.

1 Introduction

Multilayered perceptrons (MLP) are widely used to realize nonlinear mappings between input-output training data. It is known that MLPs can generalize on unknown data with reasonable accuracy. In [3] we demonstrated that the generalization capability of MLPs is generally over estimated and they can generalize well only on test points which are in the vicinity of the training data. The output of an MLP for points which lie far away from the boundary of its training sample is never reliable. This fact though known is seldom considered while training or using neural networks. An user who gets a trained neural network may (usually will) not have the training data with him (her), thus it is not possible for the user to know about the domain in which the network can perform meaningful generalizations. Some experiments reported in [3] clearly demonstrate that for classification problems, a trained MLP can produce very high response for a test point which is far away from the boundary of the training data. And in most cases such responses are *useless*. Ideally, a trained network must not respond to test points which lie far away from its training sample. We call this kind of generalization as "strict generalization". In [3] we proposed a scheme which does so only for classification problems. Also the method in [3] depends on a technique

to generate additional training points to detect the boundary of the training data. This method of generating new points becomes computationally expensive for reasonably high dimensional data. In this paper we address the same problem but with a different methodology which do not have the limitations of the method in [3]. This method is well suited for function approximation problems also and it does not require generation of additional points as in [3].

Our method involves building a composite network consisting of two subnetworks, each for a different task: (a) to learn the input-output mapping present in the training set, and (b) to learn the boundary of the training set. The composite network not only performs the main task of function approximation/classification, but also it learns a decision boundary as in classification problems. We call the first network which learns the input-output mapping as the *mapping network* and the other network which learns the decision boundary as the *vigilance network*. We propose a novel method to train the vigilance network which does not require generation of additional points as in [3], but it involves decomposing the training sample into small subsets, and making the vigilance net learn the boundary of such sets. The vigilance network is then combined with a mapping network to realize strict generalization for both function approximation (FA) and classification tasks.



(a) (b)

Fig. 1. (a)Plot of 3-Peaks (b)The points in 3-Peaks used for training

2 The Motivation

Let us consider the function:

$$y = 0.2e^{-\left(\frac{x-50}{10}\right)^2} + 0.4e^{-\left(\frac{x-25}{5}\right)^2} + 0.4e^{-\left(\frac{x-75}{5}\right)^2}. \quad (1)$$

We call this function as 3-Peaks. Fig. 1(a) depicts the function 3-Peaks. We sample a few points from the function in eq. (1) to train an MLP. Intentionally we sample points in such a manner that there remains a gap in the input space. Figure 1(b) shows the sampled points, we call this set of points as PT_1. The MLP trained with these sampled points are tested on a data set which contains 1000 equispaced points generated in [0,100]. Figure 2 shows the generalization done

on the test data by four MLPs trained with different initializations. From Fig. 1(b) it is clear that the interval [40, 60] is not represented by any training data, so the MLP is not expected to perform well over this interval. Figure 2 shows some queer generalizations. Specially the generalization shown in Fig. 2(b).

When training data are collected from a live process then there may remain areas in the input space which are not well represented by the training data or are not *at all* represented by the training data. For test points which lie in those areas, ideally, an MLP should not respond at all. But an MLP, as shown in Fig. 2, will always produce some output. We device a mechanism here which can take care of this problem.

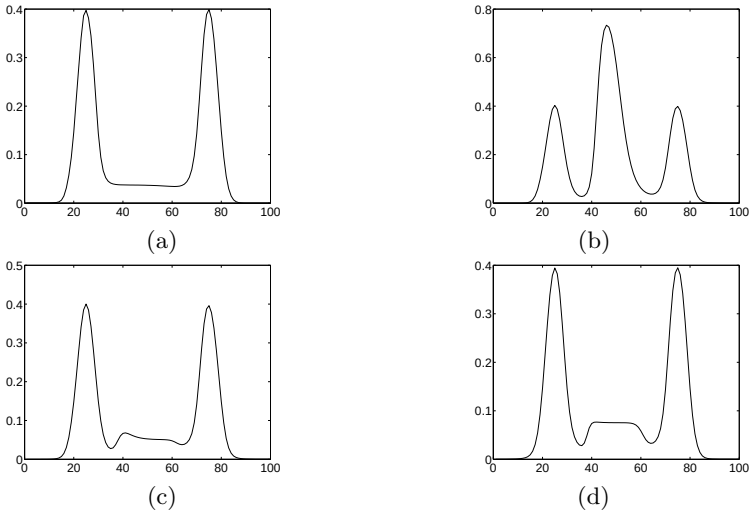


Fig. 2. Generalization an ordinary MLP trained with PT_1 for 4 different initializations

3 Training Scheme

Let $T = \{(\boldsymbol{x}_1, \boldsymbol{y}_1), (\boldsymbol{x}_2, \boldsymbol{y}_2), ..., (\boldsymbol{x}_N, \boldsymbol{y}_N)\}$ be our training set with N training samples where $\boldsymbol{x}_i \in \Re^s$ be an input vector and $\boldsymbol{y}_i \in \Re^t$ be the corresponding output vector. Let $X = \{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_N\}$ be the set of input vectors in the training set T and $Y = \{\boldsymbol{y}_1, \boldsymbol{y}_2, ..., \boldsymbol{y}_N\}$ be the set of output vectors in T. The task here is to learn the unknown input-output mapping that exists between $\boldsymbol{x}$ and $\boldsymbol{y}$. An ordinary MLP trained with conventional backpropagation or any other method can accomplish the task with a reasonable accuracy for almost all kinds of data. But we have an additional objective. We want to train an MLP in such a manner that it does not respond to test points which are away from the "boundary" of X. This can be realized if we can make the MLP learn the boundary of X along with the input-output mapping between $\boldsymbol{x}$ and $\boldsymbol{y}$. Thus, we want our network to

learn a decision boundary as in case of classification problems. To realize this we use two networks. The first one is an usual MLP which learns the input-output mapping, we call this as the *mapping* network. The second network is called the *vigilance net* which decides whether the MLP should respond to a point or not. The final output for a test point is obtained by suitably combining the outputs of both networks.

3.1 Training the Vigilance Network with Receptive Fields Around Data Points

We call the vigilance network as the receptive field vigilance network (RVN), because it uses Gaussian receptive fields around clusters of data points.

We can assume that the input vectors of the training set X can be divided into a number of hyperspherical clusters $X_i, i = 1, 2, ..., n$, such that $\cup_{i=1}^{n} X_i = X$ and $X_i \cap X_j = \phi, \ \forall i, j; i \neq j$. Such a decomposition into hyperspherical clusters can be done using any conventional clustering algorithm like the k-means [4], or the Fuzzy c-means [2]. The vigilance net is trained in such a manner that it can detect whether a test point falls in any of these clusters or not.

This RVN is a three layered network. It has s nodes in the input layer (if $X \subset \Re^s$), k nodes in the hidden layer and one node in the output layer. Each node in the hidden layer has two parameters $\boldsymbol{\mu}_i \in \Re^s$ and $\sigma_i \in \Re$ associated with it. For a input vector $\boldsymbol{x}$, the i^{th} hidden node computes

$$z_i = exp\left(-\frac{||\boldsymbol{x} - \boldsymbol{\mu}_i||^2}{\sigma_i^2}\right), \quad \forall i = 1, 2, ..., k. \tag{2}$$

The single output node in the third layer aggregates the outputs of the k hidden nodes to give a single response. Let b be the output of the third layer node :

$$b = \max_{i=1,2,...,k} \{z_i\}. \tag{3}$$

Each node in the hidden layer represents a cluster in the data set X. The parameters $\boldsymbol{\mu}_i$ and σ_i are decided using the FCM algorithm. If we decide k as the number of hidden nodes then, we find out k clusters from X and denote $\boldsymbol{\mu}_i, \quad i = 1, 2, ..., k$ as the i^{th} cluster center. FCM produces a set of centroids $\boldsymbol{V} = \{\boldsymbol{v}_1, \boldsymbol{v}_2, ..., \boldsymbol{v}_k\}$, and a partition matrix $U = [u_{ij}]_{k \times N}$, where u_{ij} denotes the degree to which $\boldsymbol{x}_j$ belongs to the i^{th} cluster and $\boldsymbol{v}_i$ is the centroid of the i^{th} cluster. Here we take $\boldsymbol{\mu}_i = \boldsymbol{v}_i$. The fuzzy partition matrix obtained from FCM can be hardened using the maximum membership rule [2]. In other words, we can consider that a point $\boldsymbol{x}_i \in X$ belongs to cluster $c, 1 \leq c \leq k$, if $u_{ci} = \max_j \{u_{ji}\}$.

So, the clustering output can be used to partition X into k disjoint sets $X_1, X_2, ..., X_k$. The σ_i is chosen as :

$$\sigma_i = \max_{\boldsymbol{x}_j \in X_i} \{||\boldsymbol{x}_j - \boldsymbol{\mu}_i||\}, \forall i = 1, 2, ..., k. \tag{4}$$

For a test point $\boldsymbol{x} \in \Re^s$ each hidden node in the vigilance network gives an output related to the distance of $\boldsymbol{x}$ from the cluster center that the node

represents. Thus, if a test point lies in or around the boundary of the cluster that a hidden node represents, then the output of that hidden node will be high. Therefore, for a test point $\boldsymbol{x} \in \Re^s$, if b takes a high value then we conclude that $\boldsymbol{x}$ lies within or around some cluster of X; otherwise, it lies far from all k clusters of X. So, b can be used as an indicator of whether $\boldsymbol{x}$ lies in or around the boundary of X.

The structure of the RVN is similar to a Radial Basis Function (RBF) network but its function is quite different from that of an RBF.

3.2 The Composite Network

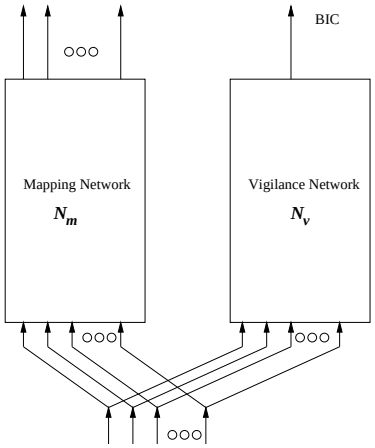


Fig. 3. The composite network $\mathcal{N} = (\mathcal{N}_m, \mathcal{N}_v)$

Another network is trained along with the vigilance network. This second network is an ordinary MLP, which is trained with the points in X along with its associated output, i.e., with T. This network is called the *mapping* network (maps input to output). The vigilance network and the mapping network are combined together to a composite network which makes the final decision. Denoting the trained vigilance network as $\mathcal{N}_v$ and the mapping network as $\mathcal{N}_m$, the final trained network $\mathcal{N}$ is represented by the tuple $\mathcal{N} = (\mathcal{N}_m, \mathcal{N}_v)$. If the output dimension of the data is t, then the composite network will have $t + 1$ output nodes. The first t output nodes correspond to the output of the mapping network ($\mathcal{N}_m$) and the $(t + 1)^{th}$ node corresponds to the output of the vigilance network ($\mathcal{N}_v$). We call the output of $\mathcal{N}_v$ as the *boundary indicator component* (BIC)(please refer to Fig. 3). A test point is fed to the composite network, and if the BIC gives a value greater than a threshold th, then the output of the test point corresponds to the output of the remaining t nodes. If the BIC bears a value lower than th, the network infers that the point is away from the boundary of the training

set and hence the net may not produce a correct output (decision) for it. The threshold th is generally user defined. In our simulations we use $th = e^{-1}$. The reason for such a choice is that in case of our vigilance network, σ_i is the largest distance of a training point that belongs to the cluster associated with the i^{th} receptive field. So, it is reasonable to assume that the receptive field of a node is extended up to a distance equal to its σ or a little beyond that. Based on this idea we choose th equal to the response of a node at a distance σ, which is equal to e^{-1}.

4 Simulation Results

We use two function approximation and two classification data sets for demonstrating the effectiveness of our network. The function approximation data sets are 3-Peaks and Boston-Housing. For the data set 3-Peaks we can show the generalization properties pictorially and conclude that our network does a good job. But, for the real life data set, Boston-Housing, such a pictorial representation is not possible as this data set is in high dimension. For this data set we define some measures which help us to evaluate the performance of our network. Let $T = \{(\boldsymbol{x}_i, \boldsymbol{y}_i) : i = 1, 2, ..., N\}$ be the training set and $X = \{\boldsymbol{x}_i : i = 1, 2, ..., N\}$ be the input vectors of the training set T. Let $X_{Te} = \{\boldsymbol{x}'_i : i = 1, 2, .., M\}$ be the input vectors of the test set. A trained composite network $\mathcal{N} = (\mathcal{N}_m, \mathcal{N}_v)$, will either respond to a test point $\boldsymbol{x}'_i$ or will not respond to it. Thus, the set X_{Te} can be partitioned into two disjoint sets X^A_{Te} and X^R_{Te}. X^A_{Te} contains the points for which the composite network produces a response and X^R_{Te} includes the points for which the composite network does not produce any response. Now, for each test point $\mathbf{x}'_i$ we define a function Δ as:

$$\Delta(\boldsymbol{x}'_i) = \min_{\boldsymbol{x}_j \in X} ||\boldsymbol{x}'_i - \boldsymbol{x}_j||. \tag{5}$$

Hence, $\Delta(\boldsymbol{x}'_i)$ represents the distance of $\boldsymbol{x}'_i$ from its nearest neighbor in X. Let $\mu_{\Delta A}$ and $\mu_{\Delta R}$ respectively denote the mean Δ for points which are accepted by the vigilance network (i.e., points in X^A_{Te}) and the points which are rejected by the vigilance network (i.e., points in X^R_{Te}) respectively. Thus,

$$\mu_{\Delta A} = \frac{1}{|X^A_{Te}|} \sum_{\boldsymbol{x}'_i \in X^A_{Te}} \Delta(\boldsymbol{x}'_i), \tag{6}$$

and

$$\mu_{\Delta R} = \frac{1}{|X^R_{Te}|} \sum_{\boldsymbol{x}'_i \in X^R_{Te}} \Delta(\boldsymbol{x}'_i). \tag{7}$$

For a test set X_{Te} if $\mu_{\Delta A} < \mu_{\Delta R}$ then it is reasonable to assume that the network serves the intended purpose. Because $\mu_{\Delta A} < \mu_{\Delta R}$ implies that the points for which the composite network responds are more close to the training data than those for which the network does not respond.

3-Peaks: The 3-Peaks data set has been discussed in Section 2. We sample 80 points uniformly from the interval [0,100]-[40,60] and call them PT_1. We test the generalization capabilities of trained networks on a test set of 1000 equispaced points generated in the interval [0,100].

As PT_1 does not contain any point in the interval [40,60] (refer Fig. 1(b)), an ordinary MLP is not expected to produce meaningful response for test points which lie in the interval [40,60]. In Fig. 2 we have already shown that this is indeed the case.

A composite network $NP_1 = (NP_{m1}, NP_{v1})$ trained with PT_1 produces better generalizations. Figure 4 shows the generalizations of 4 different composite networks. Figure 4 reveals that the composite networks do not respond to test points which fall in the area not represented in the training set. Note, the response is plotted only when BIC ≥ 0.368.

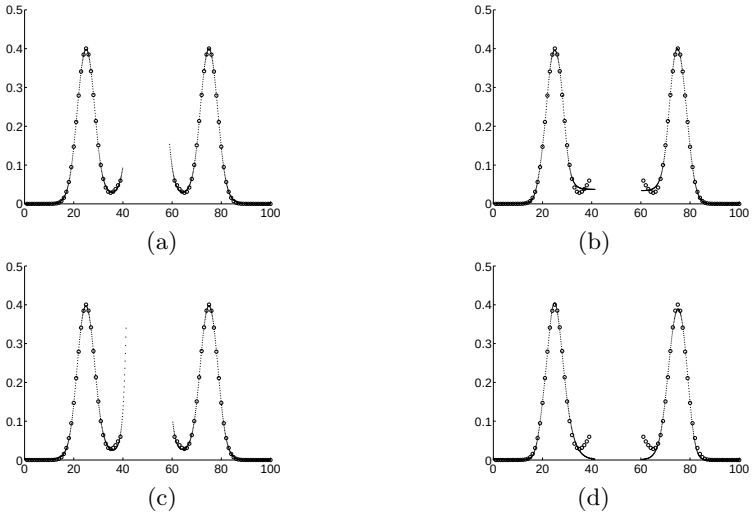


Fig. 4. Generalizations produced by $NP_1 = (NP_{m1}, NP_{v1})$, (using RVN) when trained with PT_1 for various initializations(the large dots denotes the training points).

Boston-Housing : Boston-Housing data set [1] contains 506 samples in 13 dimension and it contains only one output. We use a normalized version of this data set. We divide each input feature and the output by the respective maximum value so that they lie between 0 and 1. For Boston-Housing data, we create a random training-test partition so that the the training and test set contains equal number of data points. We train 10 different composite networks with different initializations. For each run we use a mapping network with 10 hidden nodes and a RVN with 10 receptive fields. Table 1 shows the results on the test sets for this data set. In Table 1, column 2 shows the number of points

for which the composite network makes a decision, while column 3 gives the number of cases for which the network refuses to produce an output. Comparing column 4 with column 5 we see that for all cases $\mu_{\Delta A}$ is significantly lower than $\mu_{\Delta R}$, indicating that the points for which the network makes predictions are in the vicinity of the training points. Columns 6 and 7 show the mean test error (the absolute deviation of the network response from the true output) for accepted and the rejected points. Comparing columns 6 and 7, we find that in all cases the network rejects those points which produces more error. Note that, the composite network does not respond to the rejected points (the points in X^R_{Te}), but in column 7 of Table 1 we report the deviations of the outputs of the mapping network for the rejected points ignoring the values of the BIC produced by the vigilance network. It is not expected that a trained network will produce good results for test points which are away from the training set, and comparing columns 6 and 7 we see that this is true for all the runs with Boston-Housing data.

Table 1. Run statistics for Boston-Housing on 50% Training-Test partition

Run No.	$\|X^A_{Te}\|$	$\|X^R_{Te}\|$	$\mu_{\Delta A}$	$\mu_{\Delta R}$	Mean Test error for accepted points	Mean test error for rejected points
1	245	8	0.155	0.324	0.090	0.149
2	246	7	0.152	0.369	0.092	0.212
3	249	4	0.158	0.395	0.068	0.386
4	245	8	0.153	0.439	0.078	0.187
5	251	2	0.152	0.280	0.068	0.304
6	246	7	0.157	0.330	0.115	0.140
7	243	10	0.160	0.317	0.098	0.397
8	246	7	0.158	0.454	0.445	0.602
9	247	6	0.161	0.425	0.438	0.585
10	248	5	0.158	0.246	0.153	0.219

To validate that in average $\mu_{\Delta A} < \mu_{\Delta R}$, we perform another experiment. In this experiment, we use all 506 points as training examples and test the networks with 1000 additional points generated in the 10% inflated hyperbox containing the training data. Here too we train 10 different networks and test with different test sets each containing 1000 points. Table 2 shows the results for the 10 networks. From columns 2 and 3 of Table 2 we see that the number of points rejected is much more than the number of points accepted by the composite network. This is due to the fact that the input vectors are 13 dimensional, and we have only 506 training points. So, the training points occupy only a small part of the total hyperbox bounded by the data. And most of the artificially generated points fall outside the boundary of the training sample. The scenario was different in case of Table 1 as there it is expected that the test points follow the same probability distribution as that of the training points, hence in Table

Table 2. Run statistics for Boston-Housing on artificially generated test data

Run No.	$\|X^A_{Te}\|$	$\|X^R_{Te}\|$	$\mu_{\Delta A}$	$\mu_{\Delta R}$
1	86	914	0.810	1.242
2	155	845	0.862	1.243
3	103	897	0.804	1.242
4	135	865	0.864	1.230
5	108	892	0.818	1.231
6	88	912	0.799	1.233
7	56	944	0.812	1.218
8	143	857	0.873	1.235
9	118	882	0.796	1.241
10	139	861	0.851	1.233

1 only a few points got rejected. Comparing columns 4 and 5 of Table 2 we see that for all cases $\mu_{\Delta A} < \mu_{\Delta R}$, which shows that the networks respond only to points which are in the vicinity of the training points. As in this case the test data are artificially generated, we cannot measure the deviation of the network output from the true output.

Table 3. Run statistics for Wine

Run No.	$\|X^A_{Te}\|$	$\|X^R_{Te}\|$	$\mu_{\Delta A}$	$\mu_{\Delta R}$
1	86	3	0.390	0.560
2	87	2	0.388	0.733
3	87	2	0.387	0.814
4	87	2	0.398	0.618
5	84	5	0.377	0.606
6	87	2	0.406	0.767
7	88	1	0.391	0.912
8	88	1	0.402	0.781
9	82	7	0.394	0.723
10	88	1	0.399	0.912

4.1 Results on Classification

We report results on two classification data sets : Breast Cancer [1] and Wine [1]. Tables 3 and 4 summarize the run statistics of 10 networks trained and tested for Wine and Breast-Cancer data respectively. For both these data sets we used equal number of points in the training and test sets. Also we used 10 hidden nodes in the mapping network and 10 receptive fields in the RVN. Tables 3 and 4 clearly show that for all networks $\mu_{\Delta R}$ is significantly greater than $\mu_{\Delta A}$.

Table 4. Run statistics for Breast-Cancer

Run No.	$\|X_{Te}^{A}\|$	$\|X_{Te}^{R}\|$	$\mu_{\Delta A}$	$\mu_{\Delta R}$
1	341	1	2.160	8.307
2	341	1	1.999	5.916
3	341	1	2.054	9.165
4	340	2	2.016	7.083
5	337	5	2.081	8.517
6	334	8	2.016	7.058
7	338	4	1.923	7.854
8	341	1	2.131	5.916
9	341	1	2.083	9.165
10	338	4	2.119	6.730

5 Conclusion

We proposed a training scheme for MLPs which can equip an MLP with the property of strict generalization. Our method uses a composite network that judiciously integrates two subnetworks, a mapping network and a vigilance network. The simulation results demonstrate that our training scheme serves the purpose quite satisfactorily both for function approximation and classification tasks. The basic philosophy of vigilance network is quite general in nature and can be used with other machine learning tools like radial basis function networks.

References

1. C.L. Blake and C.J. Merz, UCI Repository of machine learning databases [http://www.ics.uci.edu/ mlearn/MLRepository.html]. Irvine, CA: University of California, Department of Information and Computer Science, 1998.
2. J. C. Bezdek, J. Keller, R. Krishnapuram and N. R. Pal, *Fuzzy Models and Algorithms for Pattern Recognition and Image Processing* Kluwer, Massachusetts, 1999.
3. D. Chakraborty, N.R. Pal, "A novel training scheme for multilayered perceptrons to realize proper generalization and incremental learning", *IEEE Trans. Neural Networks*, vol 14, no 1, pp. 1-14, 2003.
4. R.O. Duda, P.E. Hart, D.G. Stork, *Pattern Classification*, John Wiley, New York, 2000.
5. M. T. Hagan and M. B. Menhaj, "Training Feedforward Networks with the Marquardt Algorithm," *IEEE Trans. on Neural Networks*, vol. 5, no. 6, pp. 989993, 1994.
6. M. Sugeno, T. Yasukawa, "A fuzzy-logic based approach to qualitative modeling," *IEEE Transactions on Fuzzy Systems*, vol. 1, pp. 7-31, 1993.