Mathware and Soft Computing 1 (1994) 139-161

An Effective Way to Generate Neural Network Structures for Function Approximation

Andreas Bastian
Lab. for Int. Fuzzy Engineering Research
Siber Hegner Bldg. 4. Fl., 89-1 Yamashita-cho,
Naka-ku, Yokohama, 231 JAPAN

Submitted by L.T. Kóczy

Abstract

One still open question in the area of research of multi-layer feedforward neural networks is concerning the number of neurons in its hidden layer(s). Especially in real life applications, this problem is often solved by heuristic methods. In this work an effective way to dynamically determine the number of hidden units in a three-layer feedforward neural network for function approximation is proposed.

1 Introduction

Multi-layer feedforward neural networks are enjoying a rising popularity and their applications range from medical applications [1], handwriting recognition [2], the game of backgammon [3], currency exchange rate prediction [4], control and identification of nonlinear dynamically systems [5], to pig carcase grading in slaughterhouses [6].

Despite this obvious success, one frequent asked fundamental question is about the size of the hidden layer(s). Especially in industrial applications where compact networks which require a minimal amount

of storage memory and computing time are needed, the so-called "right size" of a hidden layer is a very important issue. If the network is too small, a good representation of the training data might not be possible. On the other hand, an oversized network, besides requiring long computational time, might loose its generalization ability.

It has been shown in [7] —by applying the Stone–Weierstrass Theorem— that a feedforward neural net with only one hidden layer using arbitrary squashing functions is capable to approximate any Borel measurable function from one finite dimension space to another. Thus, this kind of network can be regarded as an universal approximate. However, this theorem does not provides the number of hidden units necessary for approximating a function.

Although there exist a number of approaches to determine the number of hidden units, e.g. [8-10], most of them are too complicated for a fast application, or are only suitable for some specialized application. Furthermore, all approaches do not distinguish between the task of pattern learning and function approximation. Yet, it is well-known that those two tasks are very different: pattern learning requires the memorizing power of the network, while in function approximation the generalizing ability of a network is needed.

In this work, we will mainly deal with the problem of finding the optimal network structure for function approximation. In [11-12] the bounds on the number of hidden units in multi-layer feedforward networks needed to approximate a function with any desired accuracy were given. In this study, those theoretical results are derived to a simple and more practical approach to find the optimal number of hidden units. It will be shown, that given a modeling method that converges to the output, it is possible to find the minimum number of hidden units sufficient to represent the output by dynamically adding or deleting hidden units.

The initial size of a network is also of major importance when determining its optimal structure. Thus, the usage of feature extraction of the data is proposed to determine the initial number of hidden units. For this purpose, the well-known Fuzzy C-Means (FCM) clustering algorithm [21] is employed.

Although any learning method applicable for feedforward neural networks can be applied, in this study the backpropaga-

tion-learning algorithm [13] is used, since this algorithm is by now quite well understood as there exist a number of publications concerning its learning parameters [14-15], weights [16], improved learning algorithms [17], and optimum shape of the sigmoid function [18]. Furthermore, this learning algorithm is the probably most often applied one.

This paper is organized as follows: to make this study self-contained, in the next section some preliminary definitions are provided, followed by the introduction of the proposed method in Section 3. To demonstrate the effectiveness of this method, in Section 4 it is applied to several polynomial functions and one real life example, namely the modeling of a human operator of a chemical plant. Finally, conclusions and outlook are given.

2 Preliminary discussion

2.1 The backpropagation algorithm

Let's consider a multi-layer network of continuous-valued neurons. When a pattern is presented to the network, the general activation of each neuron in the network (except the input units whose activation is clamped by the input vector) is given by using a sigmoidal function such that:

$$o_{pj} = \frac{1}{1 + \exp\left(-\left[\sum_i w_{ij} o_{pi} + \Theta_j\right]\right)} \tag{1}$$

where o_{pj} is the activation of the neuron j due to the presentation of pattern p, and w_{ji} is the weight from neuron i to neuron j. Θ_j is the bias for the neuron j.

Without any loss of generality, in this work we will use the backpropagation-learning algorithm. This algorithm can be considered as a generalization of the least-square method used for approximating functions through polynomials. The network learns to map a set of inputs to a set of outputs by iteratively adjusting the weights w of the network by:

$$\Delta w_{ji}(t+1) = \eta \delta_{pj} o_{pi} + \alpha \Delta w_{ji}(t) \tag{2}$$

where t, η, δ_{pj}, α and w_{ji} are the presentation number, the learning

rate, the error signal for neuron j, the activation of neuron i as a result of the input pattern p, the momentum factor and the weight from neuron i to neuron j, respectively.

The error signal for an output neuron is calculated as follows:

$$\delta_{pj} = \left(target_{pj} - o_{pj}\right) o_{pj} \left(1 - o_{pj}\right) \tag{3}$$

where *target* is the desired target value. This error is then backpropagated to the previous layer and the error signal for a hidden layer neuron is calculated as follows:

$$\delta_{pj} = o_{pj} \left(1 - o_{pj}\right) \sum_{k} \delta_{pk} w_{kj}. \tag{4}$$

Although it is understood that the settings of the learning parameters in this work and shape of the sigmoidal function may not yield fast convergence, the parameters of the networks are kept constant to ensure a fair comparison between several network structures.

2.2 Bounds on the number of hidden units

In a multi-layer feedforward neural network every hidden unit constructs a hyperplane decision surface in the input space. In [12] it has been shown that any arbitrary training set with p training patterns can be mapped by a multi-layer feedforward neural network with one hidden layer and $p - 1$ hidden units. The following theorem was proofed:

Theorem 1 [12]: *Let S be a k-element of E^n where $k > 1$ is an integer, and let f be an arbitrary real-valued function defined on S, i.e., $f : S \rightarrow E$. An multi-layer perceptron with $k - 1$ hidden units is capable of realizing f.*

This theorem provides the number of hidden units needed for a correct representation of the k elements. Yet, it is obvious that the number of the hidden units can be reduced if similar training patterns are presented to the network. Therefore one may say that Theorem 1 gives us the sufficient number of neurons needed for the representation of the k elements.

Corollary 1 *The sufficient number of hidden units of a multi-layer perceptron with one hidden layer needed to represent a k element finite set S is $k-1$.*

The proof of this corollary follows directly from Theorem 1. Since Theorem 1 was already proven in [12], the reader is referred to that publication.

The minimum number of hidden units needed to represent a k element finite set S can not be deduced theoretically, since it depends very much on the data and the kind of application. However, a minimum number of hidden units exists for a multi-layer perceptron with one hidden layer, as it will be shown in the following.

Let $X = (X_1, \ldots, X_n)$ be a vector of n input variables with $x = (x_1, \ldots, x_n)$ being an instance of this vector, and let Y be the output variable with y being an instance of it. According to Theorem 1, a multi-layer perceptron with one hidden layer, consisting of $k-1$ hidden units, is able to represent k instances of the input vector. One may say that, given a bounded subset Y of the one dimensional Euclidean space and a bounded subset X of the n-dimensional Euclidean space, there exists a function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ with $Y = f(X)$.

If r units are removed from the hidden layer, then given a bounded subset Y of the one dimensional Euclidean space and a bounded subset X of the n- dimensional Euclidean space, there exists a function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ with $Y_r = f(X)$. The index r denotes the number of removed hidden units. Given a criterion to determine the representation error, the influence of removing r hidden units can be defined as:

Definition 1 *The loss of information due to a removal of r hidden units is given as*

$$LI = \int |Y - Y_r| \, dX, \tag{5}$$

where the index r indicates the number of removed hidden units.

Let's now consider a finite set of pattern to be represented. This is the problem of approximating a bounded function $f : A \subset \mathbb{R}^n \rightarrow \mathbb{R}$, from a bounded subset A of n-dimensional Euclidean space to a bounded subset $f(A)$ of one dimensional Euclidean space, by means of

examples (x^1, y^1), $(x^2, y^2), \ldots, (x^k, y^k)$ where $y^k = f(x^k)$. The loss of information for the discrete case is defined as:

Definition 2 *Given a finite set of input/output data, the loss of information due to the removal of r hidden units is defined by:*

$$LI = \sum_{i=0}^{k} |y^i - y_r^i| \; dX, \tag{6}$$

where the index r indicates the number of removed hidden units.

This loss of information for the finite case can be used to observe the influence of the removal of hidden units, thus providing a possibility to determine the optimum structure of a network to represent a given input-output relation.

Theorem 2 *Given an input-output data set with x^k vectors generated randomly from A in accordance with a fixed probability density function $\rho(x)$ (ρ is assumed to be 0 outside A), and given a modeling method that converges to the output Y and a method to judge the representation of the network, then it is possible to find the minimum number of hidden units sufficient to model the output.*

Proof:

Since LI indicates the loss of information, the k training patterns can be represented by using a three-layered feedforward neural network with $k-1$ hidden units. In the following, one hidden unit after the other is removed while watching the corresponding LI values.

- $LI > 0$ indicates that the lower bound of minimal number of hidden units is already exceeded.
- $LI = 0$ indicates that there still exist enough remaining hidden units to represent the training pattern.

End of proof.

For practical applications, Theorem 2 has to be relaxed by introducing a threshold value ξ such that while LI$\leq \xi$, hidden units can still be removed. Another relaxation is to allow the existence of local minima. However, using several networks with different initial weights will likely lead us to the best possible solution, although it can not be guaranteed that the global minimum is found. Nevertheless, from the applicational point of view there is no need to find the right network size for the local minimum, if the global minimum is very hard to find. Such cases are not unusual. In fact, until 1988 it was not even known for certain whether backpropagation error surfaces have local minima, when McInerney and co-workers [19] discovered a local minimum after a 12-hour run on a Cray-2 super computer.

It is obvious, that removing one hidden unit after the other from a network with an initial size of $k-1$ hidden units will require long computing times in most cases. It is therefore better to start with a smaller initial network. Yet, since there now exists the possibility that the selected network is too small, the influence of adding hidden neurons must also be observed.

Definition 3 *Given a finite set of input/output data, the gain of information due to adding a hidden units is defined by:*

$$GI = \sum_{i=0}^{k} \left| y^i - y_q^i \right| dX, \tag{7}$$

where the index q indicates the number of added hidden units.

Given this criteria to observe the influence of adding hidden neurons, Theorem 2 is still valid even if the initial amount of hidden units is less than $k-1$, since hidden units can be added while watching the gain of information.

- $GI > 0$ indicates that the lower bound of minimal number of hidden units is not yet exceeded, since additional hidden units improve the representation capability of the network.
- $GI = 0$ indicates that there already exists enough hidden units to represent the training pattern.

Like in the case of removing hidden units, a threshold value ξ can be introduced for practical applications.

2.3 Generalization

In the previous section a methodology to determine the number of hidden units was given. However, it was assumed that a correct learning scheme is provided. Since in this study we mainly deal with the problem of finding an optimal network structure for function approximation, the generalization ability of a network must be ensured, by which the ability of a network to generalize patterns outside the training set is meant. In other words: if an input vector lies between or close to training set examples, the net should produce outputs which are reasonable related to the outputs of those training sets. Moreover, during the learning phase the network should disregard noise and outliers.

Generally speaking, generalization depends on three parameters: the complexity of the training data, the number of training data, and the network size. As mentioned in the introduction, an oversized network will loose its generalization ability and memorizes the training data instead, including its noise and outliers. It has been pointed out in [20] that a network which simply memorizes the training patterns may fail when presented with similar but slightly different patterns.

One issue of memorization is the so-called overfitting of the data. This phenomenon usually occurs if the training set contains noise and outliers. As illustrated in Figure 1, a curve fitted too good to the data set might be very poor for interpolation and extrapolation.

Overfitting is basically related to the structure of the network and the training time. Since in this work only feedforward networks with one hidden layer are considered, the structure of the network is directly related to the number of neurons in the hidden layer. One may view the number of the hidden units as the available degrees of freedom a net has in order to represent a certain input-output relation. The second big influence on overfitting is the training time, that is, the number of times a pattern is presented to the network during the training. In this case, overfitting might be explained as the attempt of the network to fit all training patterns, even after a good approximation of the input-output relation is achieved. This is often addressed as overtraining.

One approach to avoid overtraining is to stop the training when a

Figure 1: Good fit Overfitting

good generalization ability is achieved. A simple method to estimate the generalization ability is to check the network using an unknown data set. Thus, the existing data are divided into a training and a validation set. The training set is used to adjust the weights of the net during the training, while the validation set is used to estimate the generalization ability of the trained net. The training is terminated when the error on the validation set begins to rise. Figure 2 shows this situation schematically. At first, the error of both training and validation set decreases with the training time. After a certain period, the error on the validation set stops to decrease and rises again, while the error on the training set continues to decrease. At this point the training should be stopped.

In this work, this method is used to avoid overtraining. It is understood, that this method might be not very practical if only a small data set is available. Yet, as it will be shown latter, good results can still be achieved if a network with a small number of hidden units is used, which is still capable to represent the desired input-output relation.

Figure 2: One method to prevent overtraining.

3 Determining the structure of the network

3.1 Basic idea

In the last section the framework for determining the optimal structure of a network for function approximation was provided. Since Theorem 1 provides the number of hidden units needed for a representation, we could actually determine the optimal structure by recursively removing one of hidden unit, train the network with the existing data, and compare its performance with the previous structure. This cycle could be repeated as long as no significant change in the representation ability of the net occurs. Even bound to result in a good structure, this method requires too much computing time.

On the other hand it was shown, that we can actually start with a very small network structure and increase the number of hidden units one by one while watching its performance. Again, also this method may require much computing time if the final structure contains many hidden units.

Therefore, a good initial net structure should be provided. Since the number of hidden units depends strongly on the complexity of the mapping task, it seems reasonable that a methodology to determine

the complexity of the input-output relation could be helpful. Recall that if an input vector lies between or close to training set examples, the network should produce outputs which are reasonable related to the outputs of that training sets. This means that data with similar features should also have similar outputs. To some extend one may say, that the number of features of a pattern is related with its complexity.

In the field of pattern recognition, cluster analysis is an very important tool for solving many problems. One widely used clustering algorithm for feature extraction is the Fuzzy C-Means (FCM) algorithm proposed by Bezdek [21]. In this work, this algorithm is applied to detect similarities between the data.

3.2 Clustering algorithm and validity criteria

Fuzzy clustering is best understood by comparing it with the more common hard clustering method. Let $X = \{X_1, X_2, \ldots, X_n\}$ be a set of n vectors in R^d, representing the data. For an integer $c \geq 2$, a hard clustering of X into c clusters will result in c disjoint subsets S_c of X. The clusters are defined using functions. For $i = 1, \ldots, c$ we define $u_i : X \to \{0, 1\}$ by $u_i(x) = 1$ if $X \in S_i$ and $u_i(x) = 0$ if $X \notin S_i$, for all $x \in X$. These functions are called membership functions since they are describing to which cluster each data point belongs.

However, it seems to be quite natural, that the degree of membership u_i is allowed to take values between 0 and 1, thus allowing a data point to belong to more than one cluster at the same time. This is actually the idea of fuzzy sets as introduced by Zadeh [22]. In the following, the fuzzy degree of membership of the kth data belonging to the ith cluster is denoted by μ_{ik}.

The FCM-algorithm is executed in the following steps [21]:

1) Set the number of clusters $c = 2$, and initialize cluster centers V arbitrarily. Choose a scalar $\epsilon > 0$, the weight $m[1.5,\ 3]$, and set $q = 1$.

2) Initialize memberships μ_{ik} of X_k such that

$$\sum_{i=1}^{c} \mu_{ik} = 1 \quad \text{for all } k \tag{8}$$

3) Compute the center v_i of the fuzzy cluster by

$$\nu_i = \sum_{k=1}^{n} (\mu_{ik})^m x_k \Big/ \sum_{k=1}^{n} (\mu_{ik})^m, \quad 1 \leq i \leq c \tag{9}$$

4) Update the fuzzy degree of membership

$$\mu_{ik} = 1 \Big/ \left[\sum_{j=1}^{c} \left(\frac{d_{ik}}{d_{jk}} \right)^{2/(m-1)} \right], \quad \text{for } d_{ik} > 0, \, \forall i, k, \tag{10}$$

where the Euclidean distance between the kth data to the center of the ith cluster is given by

$$d_{ik} = \|x_k - \nu_i\|. \tag{11}$$

5) If

$$\left\| V^{(q)} - V^{(q-1)} \right\| < \epsilon \tag{12}$$

then terminate the algorithm, otherwise set $q = q + 1$ and return to step 2.

Having determined the number of clusters, one still faces the question how well has the algorithm identified the structure present in the data. This is the so-called cluster validity problem. The choice of the cluster validity function is depending on the desired clustering. In this work the cluster validity function used in [23] is applied:

$$S(c) = \sum_{k=1}^{n} \sum_{i=1}^{c} (\mu_{ik})^m \left(\|x_k - \nu_i\|^2 - \|\nu_i - \overline{x}\|^2 \right). \tag{13}$$

where n is the number of data, c the number of clusters and $\overline{x}$ is the average of x. The number of clusters c is determined so that $S(x)$ reaches a local minimum. As one can see in (13) the first term of the right hand side of the equation is the variance of the data in a cluster, and the second term is that of the clusters themselves.

3.3 Finding the optimal number of hidden units

Having found the number of features of the data, this number is used to determine the number of hidden units of the initial network structure. The complete algorithm is as follows:

1) Determine the number of features f by clustering the output of all available data using the FCM-algorithm. Use the validation function given in Eq. (13) to stop the clustering procedure.

2) Divide the data into training and validation set.

3) Generate a three-layer network with $q = f$ neurons in its hidden layer.

4) Train the network using the training data. The training is terminated if the error on the validation set begins to rise or 50000 training cycles have been completed.

5) Calculate the mean square error $E(q)$ on the training set

$$E(q) = \sum_{i=1}^{k} \left(y_i - y_i^{net} \right)^2 / k, \tag{14}$$

where k is the number of data, y_i is the desired output, y_i^{net} is the output of the net, and q the number of neurons in the hidden layer as optimal solution.

6) Add one hiden unit, initialize the network and repeat the steps 4 and 5. If

$$\frac{E(q) - E(q+1)}{E(q)} \geq 0.01 \tag{15}$$

then set $q = q + 1$, and repeat step 6.
Else go to step 7.

7) If $q = f$ go to step 8. Else terminate the algorithm with q neurons in the hidden layer.

8) Remove one neuron from the hidden layer, initialize the network and repeat the steps 4 and 5. If

$$\frac{E(q+1) - E(q)}{E(q)} \leq 0.01 \tag{16}$$

then set $q = q - 1$, and repeat step 8.
Else terminate the algorithm with q neurons in the hidden layer as optimal solution.

4 Results

In this section, the practical applicability of the proposed algorithm is examined using several examples. In all examples feedforward neural networks with one hidden layer and the backpropagation learning algorithm are used. The learning factor η and the momentum factor α are set to 0.6 and 0.5, respectively. This constant setting enables a fair comparison of the tasks.

4.1 Polynomial functions of different orders

The algorithm is applied to approximate polynomial functions of different order of the form

$$z = x^n + y^n; \quad x, y \in [-2, 2]. \tag{17}$$

The training and validation sets are each containing 250 data vectors. The maximum number of iterations was set to 50000.

Example 1. Approximating $z = x^2 + y^2$.

In step 1 of the algorithm, all data of a polynomial function of the second order were used to determine the optimal number of clusters. As shown in Table 1, according to the cluster validity function given in Eq. (13), the optimal number of clusters is 11. For the readers convenience, the results of the cluster validity function for 2 until 14 clusters are also listed in Table 1.

clusters / number of hidden units	Cluster validity	MSE
2	–73.517	0.01728
3	–3772.108	0.000171
4	–5877.071	0.001728
5	–6192.468	0.000159
6	–6801.010	0.000142
7	–6991.644	0.000162
8	–7128.981	0.000179
9	–7783.212	0.000147
10	–7883.291	0.000169
11	*–8412.938*	0.000139
12	–8378.291	*0.000104*
13	–8292.963	0.000120
14	–8333.124	0.000130

Table 1. Results for the polynomial function $z = x^2 + y^2$.

In the following step 2, the data were divided into training and validation data sets. In step 3 a network with one hidden layer containing 11 hidden units was initialized and trained. After the training phase was terminated, according to Eq. (14) a mean squared error (MSE)=0.0000139 was calculated. The result can be found in the right column of Table 1.

In the steps 4 and 5, a net with one additional hidden unit was initialized and trained. Since the resulting MSE is 0.0000104 and therefore much smaller than the previous computed value (0.000139), step 4 and 5 of the algorithm were repeated. This time, the resulting MSE after the training was much bigger than the previous one, thus the algorithm was terminated with 12 hidden units as optimal solution.

For the readers convenience, the MSEs for all networks with 2 until 14 hidden units are also displayed in Table 1. As one can see, the network size found by the proposed algorithm is indeed the best solution.

Example 2. Approximating $z = x^3 + y^3$.

Since the procedure of finding the optimum structure of the network is always the same, a detailed description is left out.

Notice that in this example the number of clusters is identical with the final solution of the algorithm. However, if the clustering algorithm is continued, another minimum can be found for 13 clusters. The corresponding MSE to that network size (13 hidden units) was indeed smaller than the one with 11 hidden units, as shown in table 2. This results indicates, that the clustering algorithm should be continued even if a minimum is reached. The cause of this lies in the fact, that the validation function only finds local minimum. However, the first minimum found also leads to a good representation.

clusters / number of hidden units	Cluster validity	MSE
2	–1203.991	0.002185
3	–1529.123	0.001542
4	–2115.547	0.001121
5	–2537.083	0.000038
6	–2674.592	0.000027
7	–2717.522	0.000027
8	–3621.382	0.000025
9	–4600.322	0.000025
10	–4723.944	0.000025
11	*–4752.003*	*0.000019*
12	–4721.030	0.000025
13	*–4782.269*	*0.000015*
14	–4752.302	0.000025

Table 2. Results for the polynomial function $z = x^3 + y^3$.

Example 3. Approximating $z = x^4 + y^4$.

The results for approximating the fourth order polynomial function also indicate that the search for the optimum number of clusters should be continued for several steps. As shown in Table 3 the cluster validity funciton reached a minimum for 8 clusters, and in this case, the network structure identification algorithm terminated with 9 hidden units as the optimal solution.

However, the clustering procedure was continued and the validity function reached another minimum at 14 clusters, here the algorithm

terminated with 14 hidden units as the optimal solution. Note that in this case a smaller MSE was achieved, as shown in Table 3.

clusters / number of hidden units	Cluster validity	MSE
2	–47.430	0.014685
3	–603.453	0.002632
4	–1031.382	0.001712
5	–1610.574	0.002310
6	–1740.545	0.000120
7	–1960.001	0.000111
8	*–1974.007*	0.000168
9	–1960.532	*0.000083*
10	–1942.900	0.000095
11	–1921.112	0.000085
12	–1904.732	0.000080
13	–1905.232	0.000075
14	*–2012.583*	*0.000060*
15	–1992.241	0.000068

Table 3. Results for the polynomial function $z = x^4 + y^4$.

Example 4. Approximating $z = x^5 + y^5$.

Finally, the algorithm was applied on data from a fifth order polynomial function. In this case there exists only one minimum for the validation function, as shown in Table 4. The optimal number of cluster was 7, which also turned out to be the optimal number of hidden units.

clusters / number of hidden units	Cluster validity	MSE
2	–719.201	0.001983
3	–1720.214	0.000872
4	–2414.241	0.000026
5	–3212.231	0.000020
6	–3444.954	0.000021
7	*–3721.328*	*0.000019*
8	–3671.293	0.000020
9	–3674.293	0.000020
10	–3420.203	0.000020
11	–3523.023	0.000020
12	–3600.343	0.000020
13	–3670.324	0.000020
14	–3669.342	0.000020

Table 4. Results for the polynomial function $z = x^5 + y^5$.

4.2 Human operation of a chemical plant

In the last example the proposed algorithm is tested using data of a human operator's control of a chemical plant producing a polymer by the polymerization of some monomers [23]. The operator determines the set point for the monomer flow rate and passes this information to a PID controller, which calculates the actual monomer flow rate input for the plant. There are 70 data sets available with each set consisting of five inputs (monomer concentration, change of monomer concentration, monomer flow rate, temperature 1 and temperature 2 inside the plant) and one single output, namely the set point for monomer flow rate.

Here, the initial network contained 6 hidden units. The algorithm terminated with 8 hidden units (see Table 5).

Figure 3: Result of the modeling.

number of hidden units	MSE
6	0.000035
7	0.000032
8	*0.000030*
9	0.000032

Table 5. Result for the chemical plant.

The normalized modeling result of the network with 5 inputs, 8 hidden units and one output is compared with the actual handling data of the human operator in Figure 3. An overall good fit between the real data and the model can be observed. Note that the biggest error occur when abrupt changes of the operator's control actions happen,

e.g. at the very early beginning of the process.

5 Conclusions

In this study a simple algorithm to dynamically determine the number of hidden units of a three-layer neural network was proposed. This algorithm starts by clustering the output data. The number of clusters is the initial number of hidden units of the network. In the following, the hidden layer of the network grows or shrinks in size, depending on the result of the generate-and-evaluate procedure.

The results in section 4 indicate, that the application of the FCM-algorithm to determine an initial network structure is very useful. In all cases the clustering result was close to the final result of the algorithm, and therefore saved much computational time. Yet, one should always keep in mind that the FCM-algorithm may get stuck in a local minimum, as it has been shown in some of the presented examples. The clustering procedure should therefore be continued for at least another three clustering steps after the validation function found a first minimum.

Although the proposed algorithm is mainly thought to be used for function approximation, it can also be used to some extend for pattern learning. In this case, all input and output data should be clustered.

The algorithm was developed for potential control and modeling design applications. Current and future work is therefore directed in that way.

Acknowledgments

The author would like to thank Prof. T. Terano, the director of LIFE, Prof. L.T. Kóczy, who endows the LIFE chair of fuzzy theory at the Tokyo Institute of Technology, and especially the reviewers for their kind help and valuable suggestions.

References

[1] Saito, K., Nakano, R., Medical diagnostic expert System based on PDP model, *Proc. Int. Joint Conf. on Neural Networks*, pp. 255-262, San Diego, 1988.

[2] Le Cun, Y., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubard, W., Jackel, L.D., Backpropagation applied to Handwritten ZIP code Recognition, *Neural Computation*, Vol. 1, pp. 541-551, 1989.

[3] Tesauro, G., Neurogammon Wins Computer Olympiad, *Neural Computation*, Vol. 1, pp. 321-323, 1990.

[4] Refenes, A.N., Azema-Barac, M., Chen, L., Karoussos, S.A., Currency Exchange Rate Prediction and Neural Network Design Strategies, *Neural Comput. & Applic.*, Vol. 1 (1), pp. 46-58, 1993.

[5] Narendra, K.S., Parthasarathy, K., Identification and Control of Dynamical Systems Using Neural Networks, *IEEE Trans. on NN*, Vol. 1, N. 1, March, 1990.

[6] Thodberg, H.H, Neural Information Processing System for Pig Carcase Grading in Danish Slaughterhouses, *Neural Comput. & Applic.*, Vol. 1 (4), pp. 248-255, 1993.

[7] Hornik, K., Stinchcomb, M., White, H., Multilayer feedforward networks are universal approximators, *Neural Networks* 2, pp. 359-366, 1989.

[8] Sietsma, J., Dow, R.J.F., Neural net pruning - why and how, *Proc. IEEE Int. Joint Conf. Neural Networks*, pp. 1325-1333, San Diego, 1988.

[9] Karnin, E.D., A Simple Procedure for Pruning Back-Propagation Trained Neural Networks, *IEEE Trans. on Neural Networks*, Vol. 1(2), June 1990.

[10] Noda, I., Learning Method by Overload, *Proc. IJCNN'93*, pp. 1357-1360. Nagoya, 1993.

[11] Sartori, A.S., Antsaklis, P.J., A simple method to derive bounds on the size and train multilayer neural networks, *IEEE Trans. on Neural Nets*, letter, Vol. 2 (4), pp. 467-471, 1991.

[12] Huang, S.C., Huang, Y.F., Bounds on Number of Hidden Units in Multilayer Perceptrons, *IEEE Trans. Neural Networks*, Vol. 2 (1), pp. 47-55, 1991.

[13] Rumelhart, D.E., Hinton, G.E., Williams, R.J., Learning repesentation by back-propagating errors, *Nature*, 232, pp. 533-536, 1986.

[14] Ochiai, K., Usui, S., Improved Kick Out Learning Algorithm with Delta-Bar-Delta-Bar Rule, *Proc. IEEE Int. Conf. on NN 93*, vol. 1, pp. 269-274, San Francisco, 1993.

[15] Vogl, T.P., Mangis, J.K., Rigler, A.K., Zink, W.T., Alkon, D.L., Accelerating the convergence of the Back-Propagation Method, *Biol. Cybern.* 59, pp. 257-263, 1988.

[16] Pirez, Y.M., Sarkar, D., Back-Propagation with controlled Oscillation of Weights, *Proc. IEEE Int. Conf. on NN 93*, vol. 1, p. 21-26, San Francisco, 1993.

[17] van Ooyen, A., Nienhuis, B., Improving the Convergence of the Back-Propagation Algorithm, *Neural Networks*, Vol. 5, p. 465-471, 1992.

[18] Yamada, T., Yabuta, T., Neural Network Controller Using Autotuning Method for Nonlinear Functions, *IEEE Trans. on NN*, Vol. 3, N. 4, July 1992.

[19] McInerney, J.M, Haines, K.G., Biafore, S., Hecht-Nielsen, R., Can backpropagation error surfaces have non-global minima?, *IJCNN'89*, II, 627, 1989.

[20] Sietsma, J., Dow, R.F.J., Creating Artificial Neural Networks That Generalize, *Neural Networks*, vol. 4, pp. 67-79, 1991.

[21] Bezdek, J.C., Pattern Recognition with Fuzzy Objective Function Algorithm, New York: Plenum Press, 1981.

[22] Zadeh, L.A., Fuzzy Sets, *Inform. Contr.*, Vol. 8, pp. 338-353, 1965.

[23] Sugeno, M., Yasukawa, T., A fuzzy-Logic-Based Approach to Qualitative Modeling, *IEEE Trans. on Fuzzy Systems*, 1 (1), 1993.