A Self Growing Learning Algorithm for Determining the Appropriate Number of Hidden Units

Sheng-De Wang and Ching-Hao Hsu
Department of Electrical Engineering
National Taiwan University, Taipei 10764, Taiwan
E-mail: sdwang@twnmoe10.bitnet

Abstract

In this paper, we propose a new algorithm to determine the appropriate number of hidden units in a multilayer feedforward neural network. This algorithm is based upon "Heuristic Terminal Attractor Back Propagation" (HTABP), which can finish learning in finite time, reach the global minimum of error function and guarantee to converge faster than the back propagation (BP) algorithm. The criterions of adding a new hidden unit are the time-varying gain γ_c of HTABP and the normalized error function. Several simulation results show the algorithm is effective and reliable. With this algorithm, the estimation of number of hidden units by trial and error is not necessary any more.

1. Introduction

It has been proved that a network with as few as one hidden layer and appropriate hidden units is capable of arbitrarily accurate approximation to any real-valued continuous functions over a compact set [1] [2]. However, there is hardly a rule or method to determine the "appropriate number" of hidden units so far. It is clearly that a network with too many neurons will not be necessary and take high cost in circuit implementation while a network with too less neurons may not approach the desired function accurately. People usually determine the appropriate number of hidden units by trial and error. As a result, it may take a long time to get a network with proper size.

In recent years, several researchers have tried to solve the problem of determining the optimal number of hidden units. For example, Tenorio and Lee [3] introduced a self-organizing neural network network structure which can construct the network itself for system identification application. Fu [4] discussed the effect of the number of hidden units and input units by experimental results. Recently, Hirose, Yamashita and Hijiya [5] developed an algorithm based upon the BP algorithm and used the total error function as a criterion for varying the number of hidden units. The criterion proposed in [5] is that when the network becomes trapped in a local minimum, a new hidden unit is added for changing the shape of the weight space so that the network can escape from local minima. The local minimum condition corresponds to the case where the total error function is not small enough and decreases very slow.

In this paper, we introduce a new algorithm called "Self Growing Learning Algorithm" (SGLA) to determine the appropriate number of hidden units. This algorithm are developed from the HTABP algorithm [7], which can finish learning process in finite time, reach the global minimum of error function and guarantee to converge faster than the BP algorithm. Some ideas of the proposed algorithm SGLA are similar to Hirose's [5] but our algorithm has a complete set of rules for adding or deleting a hidden unit and shortens the learning time owing to the HTABP algorithm used. We argue that the use of the magnitude and the changing rate of the total error function as the only rule for deciding the adding of a hidden unit in [5] is not sufficient since the trapping in a local minimum may due the BP algorithm itself as well as the deficient in hidden units. We propose a time-varying factor together with the total error function for making the decision of adding or deleting a hidden unit. The time-varying factor used is important to the HTABP algorithm and can reflect the status of learning process.

2. Heuristic Terminal Attractor Back Propagation

2.1 Terminal Attractor

The concepts of terminal attractor are first introduced by Zak [6]. The terminal attractors are equilibrium points of a dynamic system where the Lipschitz condition is violated. The Lipschitz condition guarantees the existence of a unique solution for each of initial conditions. Hence, a solution evolving from an initial condition cannot intersect the corresponding equilibrium point, and therefore, the time of approaching the equilibrium points is always infinite. If the Lipschitz condition is violated, the fixed point becomes a singular solution which is intersected by all the attracted transients. Dynamic systems with terminal attractors can reach terminal attractors in finite time. For example, the equilibrium point 0 of the differential equation $\dot{x} = -x^k, 0<k<1$ is a terminal attractor, since the condition $dx/dx = -\infty$ at x=0 violates the Lipschitz condition $|dx/dt| < \infty$. So x can reach the point x=0 in finite time for any positive initial condition.

CH 3065-0/91/0000-01098 $1.00 © IEEE

2.2 HTABP

The HTABP algorithm is a new learning algorithm for multilayer neural networks based on the concept of terminal attractor and the back propagation algorithm. The key point is the introducing of time varying gains γ_c to the weight update law of BP algorithm. The concept of the gains γ_c is that it will depend on the total error function E and the derivative of E with respect to weights such that a terminal attractor for energy transients is formed at the equilibrium point of zero. We briefly explain the key concepts.

Define the error function E to be

$$E=\frac{1}{2}\sum_m\sum_l(\overline{V_l^m}-V_l^m)^2 \qquad (2.1)$$

where $\overline{V_l^m}$ and V_l^m are the target output and the actual output for the l-th neuron at output layer corresponding to the m-th pattern. Assume there are p processing elements used for implementing the neural network. The time evolution of weights is now given as

$$\frac{dT^c}{dt}=-\gamma_c\nabla_{T^c}E \qquad c=1,2,...,p \qquad (2.2)$$

where T^c means the weights vector governed by processing element c and γ_c is the adaptive gain of processing element c. γ_c can be formulated by

$$\gamma_c=\frac{E^k}{p\|\nabla_{T^c}E\|^2} \qquad c=1,2,...p,\ 0<k<1$$

Note that if $\gamma_c=1$, the weight update law degenerates to be the BP algorithm. Now substitute γ_c to (2.2), then the derivative of E with respect to time becomes

$$\frac{dE}{dt}=\left[\frac{\partial E}{\partial T^1}\ \cdots\ \frac{\partial E}{\partial T^p}\right]\cdot\begin{bmatrix}-\gamma_1\dfrac{\partial E}{\partial T^1}\\ \cdots\\ -\gamma_p\dfrac{\partial E}{\partial T^p}\end{bmatrix}$$

$$=-(\gamma_1\|\nabla_{T^1}E\|^2+\gamma_2\|\nabla_{T^2}E\|^2\ldots+\gamma_p\|\nabla_{T^p}E\|^2)$$
$$=-E^k. \qquad (2.3)$$

If there are sufficient number of hidden units such that the equilibrium point $E=0$ in (2.3) can be reached and $0<k<1$, the equilibrium point $E=0$ is a stable terminal attractor. With the properties of terminal attractor described previously, the relaxation time for a solution of the error function E from an initial condition $E=E_0>0$ to the terminal attractor zero is finite. Also we can find that $E\geq 0$, $\frac{dE}{dt}\leq 0$ and that $\frac{dE}{dt}=0$ only when $E=0$. By the Liapunov method, the system governed by (2.2) is a globally asymptotically stable system, so we can find the global minimum of error function.

The weight update law of HTABP is switched between BP's weight update law and (2.2) according to the value of γ_c, i.e., the adaptive gain γ_c is constrained to be greater than or equal to 1. This constraint guarantees HTABP to converge faster than the BP algorithm. For more discussion and simulation results about HTABP please refer to the paper [7] which also submitted to this proceedings. The HTABP algorithm is presented as the following.

HTABP Algorithm

*•Define NE to be normalized error function, i.e., NE=E/(pattern no. * output neuron no.)*
•Assume there are p processors.
•Initialize all variables: weights, input patterns and targets. Input the parameter Error-tolerance, Maxpass, h, a. All weights distribute uniformly in [-0.5 0.5], pass = 0.
•while (NE > Errortolerance and pass < Maxpass) do
{
Calculate forward to find the network outputs, V_i^m
Calculate backward to find δ_j^m *and* γ_c *, c= 1,2,3,...,p*
if $\gamma_c>1$ *update weights by*
$T_{ji}^c(n+1)=T_{ji}^c(n)+\eta\gamma_c\delta_j^mV_i^m(n)$
else update weights by
$T_{ji}^c(n+1)=T_{ji}^c(n)+\eta\delta_j^mV_i^m(n)+\alpha\Delta T_{ji}^c(n-1)$
pass=pass+ 1 }

3. Self Growing Learning Algorithm

3.1 Motivation

Now, we will present our algorithm "Self Growing Learning Algorithm" (SGLA). This algorithm can add new hidden units until the learning process is successful, then remove some redundant hidden units to get a more economical network.

The idea of SGLA is similar to simulated annealing. The principal concept of simulated annealing is that the probability of accepting a new state which makes system energy increase is $\exp(-\frac{\Delta E}{Temp})$, where ΔE is the energy changed by new state and *Temp* is the system temperature. The ground state of the system can be found by lowering temperature slowly. In addition to take the energy function as the criterion for making decision, we use the time-varying gains γ_c in the HTABP algorithm as another criterion, which may be very large when the trajectory enters a flat region of the error surface, since $\gamma_c \rightarrow \infty$ as $\|\nabla_T{}^c E\|^2 \rightarrow 0$. In this case, we may add a new node or continue the learning. Before we present our algorithm, we analysis the system first.

3.2 System Analysis

We have verified the HTABP algorithms can find the global minimum of the system, provided that there exists a set of weight making E=0. In the sequel, we will refer the set of weights that globally minimize the error function (i.e. satisfy E=0) to as the optimal set of weights. It is noted that if there has a sufficient number of hidden units the network can be trained such that value of error function can be sufficiently small. If the number of hidden units is not sufficient, the condition E=0 will not be reached no matter what kind of the learning algorithm is used. Now we are interested in what will happen in case the weight update law (2.2) is used and the optimal set of weights does not exist. If the optimal set of weights does not exist, the error function is an illegal Liapunov function and all the properties of terminal attractors are not guaranteed. The introducing of γ_c into weight update law (2.2) results in a singular point at $\nabla_T{}^c E = 0$. This means that when the error surface enters into a flat region the gains γ_c are so large such that the system (2.2) may either become unstable, where γ_c approaches infinity, or incur oscillations, where the value of the error function is large but bounded. The HTABP algorithm implements weight update law (2.2) by using first-order numerical integration and thus suffer some numerical problems, especially when γ_c approaches infinity. as well as some additional numerical problems, when γ_c approaches inifinty. We note that if the back propagation algorithm is used rather than the HTABP, the system is guaranteed to be stable since the gradient method always converge to a local minimum.

The conditions which may occur during a learning process are depicted as a tree as shown in Fig 1. From the analysis given above and Fig. 1, two criterions can be identified for the decision making of adding hidden nodes: the value of the time-varying gains γ_c and the minimum error function during the learning. Hirose [5] also used the minimum error function as a criterion of adding new hidden nodes. His criterion is that if the error function can't reduce one percent after 100 weight corrections add a new hidden unit immediately.

Some parameters in the proposed algorithm are listed and briefly described below.

Threshold: When γ_c is larger than **Threshold**, the algorithm may continue learning or add a new hidden node depending on a probability **Prob.** If we decide to continue the learning process, Threshold will be set to a larger value. Otherwise, **Threshold** will be reset to the an initial setting.

Temp: This parameter determines the probability of keeping learning continuously or adding a new node.

Prob: The probability of adding a hidden unit. The value of **Prob** is determined by **Temp** and NE. We hope the probability of adding a new node will decrease as the learning time increases.

Minerr: This parameter records the minimal value of error function during a learning process.

Count: This parameter records the number of iterations where the normalized error function is greater than **Minerr**. If the error function is less than the **Minerr**, **Count** will be reset and **Minerr** is set to the current value of the normalized error function. When **Count** is larger than the value we set, we add a new node immediately .

3.3 Self Growing Learning Algorithm(SGLA)

To simplify the problem, a network with one hidden layer is considered. The purpose of the SGLA is to vary the number of hidden units such that the network can learn to approximate a given function with the implementation cost as least as possible. The flow chart of SGLA is shown in Fig. 2

The SGLA mainly comprises three rules for adding or removing hidden unit, as described in the following.

a) The γ_c hidden unit adding rule:

A very large values of γ_c or a surge of the gains γ_c over the **Threshold** means the trajectory of

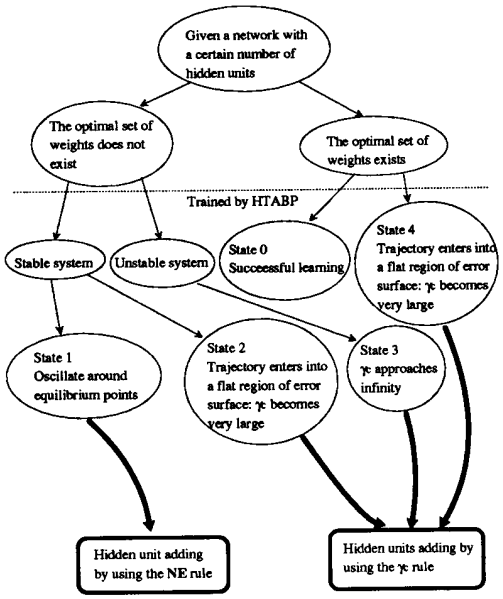

Fig. 1. The conditions ocur during a learning process using the HTABP algorithm.

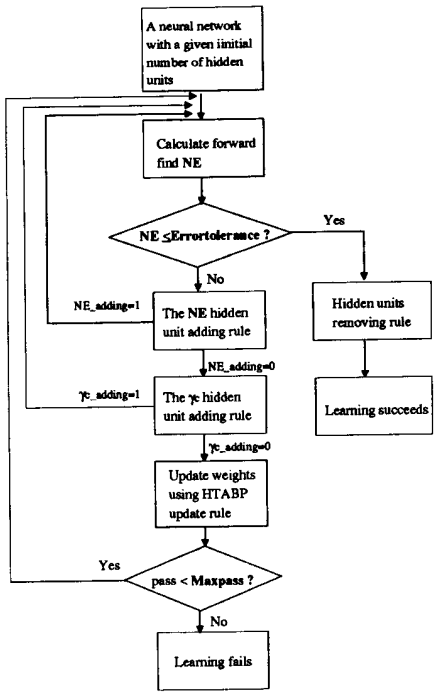

Fig. 2 The flow chart of the SGLA

Input: γ_c, **NE, Minerr**
Output: γ_c*_addingd= 1 add a new hidden unit.*
γ_c_addingd= 0 do not add a new hidden unit.
If (γ_c > **Threshold**)
*{***Prob**=*1-exp(-***NE/Temp***);*
Temp=*2**Minerr;*
If (**Prob** < *random()) {*
raise **Threshold** *by appropriate quantity;*
γ_c_adding=0;}
else γ_c_adding=1;}
else γ_c_adding=0;

Fig. 3 The γ_c hidden unit adding rule

Input: **NE, Minerr**
Output: **NE***_adding =1, add a hidden unit.*
NE*_adding =0, do not add a hidden unit.*

*if(***NE < Minerr***)*
*{***Minerr**=**NE;***
Count=*1;*
NE*_adding=0 }*
*else {***Count=Count+***1 ;*
*if (***Count > Maxcount***)* **NE***_adding=1*
else **NE***_adding=0 ;}*

Fig. 4 The **NE** hidden unit adding rule

error transients comes into a flat region of the error surface. As can be seen from Fig.1, a learning process using HTABP update law may go into one of the five states, among which state 2 and state 4 have very large values of γ_c while in state 3 γ_c would approach infinity. In the case of state 3, a hidden unit should be added for the possibility that the system may become stabilized. For state 2 and state 4, the value of γ_c may still can be tolerated and the learning can be either continued to find the optimal set of weights or restarted by adding a new hidden units to escape from current embarrassing situation. Since the two actions to be taken in state 2 and state 4 are not distinquisable by γ_c, One way to choose which actions can be resort to a random process. We define the parameter **Prob** to be $1-e^{-NE/Temp}$. **Prob** is the probability to add a new node to the network, which will decay as **NE** decreases but won't decay too fast because **Temp** becomes small during the learning process. The γ_c hidden unit adding rule can be expressed as the algorithm shown in Fig. 3.

b) The **NE** hidden unit adding rule:

If there is a deficit in hidden units, the network can not realize a desired function. In this case, as depicted in Fig. 1, the system governed by HTABP update law may set into a situation that the normalized error is bounded and satisfies **NE > Errortolerance** and oscillates around an equilibrium point or between equilibrium points. If the network stays in the above situation for a long time, we consider that **NE** will not reduce any more. Parameter **Count** is proposed to characterize this situation. As **Count** exceed a given value **Maxcount**, it is reasonable to conclude that the learning process can't reduce error function any more so we add a new node and restart learning. The NE hidden unit adding rule is shown as following

c) The hidden unit removing rule:

As soon as the algorithm finds the optimal set of weights, we save the set of weights in a temporary location and then remove the last added node and its corresponding weights. The learning process is continued and the **NE** hidden unit adding rule is applied to make decision if the deleted node should be recovered: if the learning process can still converge, i.e., **NE < Errortolerance**, we remove one node again; otherwise, we recover the previous set of weights from the temporary location and stop the algorithm. The hidden unit removing rule is shown in Fig. 5. This procedure is proposed for removing some redundant hidden nodes. We remove the last added hidden unit because we think it is an auxiliary unit to make learning easily.

The SGLA algorithm without removing rule may produce redundant hidden units. The probability of this condition may be reduced by setting a high **Threshold**, then the learning process will have high probability to success before adding new hidden units. On the other hand, a high **Threshold** need a longer time to be triggered so the searching process will take a long time.

Input: A trained network with redundant hidden units.
Output: A final network with redundant hidden units removed.
do{
  Save current network
  Remove the last added hidden unit
  Calculate forward, find **NE**
  do{
    Apply the **NE** *hidden unit adding rule*
    If (**NE**_*adding=1) exit*
    else
      Update weights by HTABP weight update law
      pass=pass+1
  } While (**NE** > **Errortolerance** *and pass* < **Maxpass**)
} While (**NE** < **Errortolerance**)

Fig. 5 The hidden unit removing rule

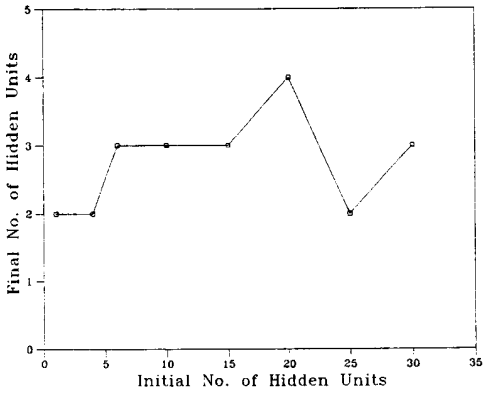

Fig. 6 The relation between initial number of hidden units and final number of hidden units found by SGLA

4. Simulation Results

In this section, three experiments are carried out through computer simulations to try to answer the following question.

- Is the final hidden unit number found by the SGLA algorithm to some extent independent to the initial guess of hidden unit number?
- Is the final hidden unit number found by the SGLA algorithm appropriate?

We assume only one processor governs the whole neural network, and the sigmoid function is tanh(x) for all simulation. Now we present the experiments and their results.

1) The first experiment is to investigate the effect of the initial guess of the number of hidden units on the final number of hidden units. XOR is used as a training example and different initial numbers of hidden units are given. Fig.6 shows the results. We can see from Fig. 6 that the final number of hidden units found by SGLA is in the interval [2:4] even the initial number of hidden units is distributed in a large region of [1:30]. This result demonstrates that the SGLA is reliable.

2) The second experiment checks whether the number of hidden units found by SGLA is proper or

not. Four examples are chosen to be modeled by neural networks. They are (i) coordinate transformation, (ii) a sinusoidal function: $h(x)=0.5\sin(\frac{2\pi x}{10})+0.5\sin^2(\frac{2\pi x}{20})$, (iii) 4 bits parity check, (iv) 3 to 8 decoder. In order to compare with the networks found by SGLA, several networks with fixed number of hidden units are trained by the BP algorithm each with 10 sets of initial weights. If the learning fails for all given initial weights, we may think the network has no chance to approach the targets. The results are shown in Table 1. We can see that the SGLA finds the optimal hidden nodes for the example (iii) and (iv) and produce some redundant node for example (i) and (ii). From the Table, we can find the network with too less hidden nodes must take more iterations to converge and the probability of success is lower than that with more hidden nodes. For example, in example (i), the network with 11 hidden nodes has faster convergent rate and higher probability to learn successfully.

Example	InitialTemp	Initial-Threshold	MaxCount	Error-tolerance	initial number of hidden node	final number of hidden node
(i)	0.03	9000	2000	0.0005	1	11
(ii)	0.03	9000	1500	0.0005	1	4
(iii)	0.03	9000	2000	0.0005	1	4
(iv)	0.03	9000	2000	0.0005	1	3

(a)

Example	No. of hidden nodes	Average iteration of successful cases	No. of failure
(i)	8	80960.	9
	9	79296.	8
	10	58187.7	7
	11	45750.7	6
(ii)	2	>100000	10
	3	54256.	0
	4	43297.7	0
	6	40900.1	2
(iii)	3	>30000	10
	4	9109.3	4
	5	4287.6	5
(iv)	2	>30000	10
	3	19241.1	1
	4	11685.7	0

(b)

Table 1 The simulation results of the SGLA. a) the initial parameters used by the SGLA and the final number of hidden units. b) the results of network with fixed number of hidden units trained by BP algorithm, where 10 sets of initial weights are tried.

3) Some examples found in literatures are taken as training examples in our SGLA in this experiment. They are 1) Baba's partial parity check[8]; 2) Baba's 6 bits parity check [8]; 3) Jacob's multiplexer example [9]. The results are listed in Table 2. Obviously, these training examples can be achieved by smaller network using SGLA. This comparison of the networks found by SGLA and the networks used by other authors is not fair in some sense. For example, the networks used by Baba have two hidden layers and the sigmoid function is $\frac{1}{1+e^{-x}}$. Even this we still believe that SGLA is an effective tool to design a network with proper size.

5. Conclusions

In this paper, we develop a new algorithm called SGLA. This algorithm can adjust the number of hidden units automatically. The criterions to increase hidden units are the time-varying gain γ_c of

HTABP and the normalized minimum error function. The simulation results show that this algorithm is effective in finding the proper number of hidden units. With the algorithm, we need not to estimate the number of hidden units by trial and error any more. We believe that the proposed SGLA algorithm are helpful in determining a proper size of network.

Example	No. of pattern	network used by author	network found by SGLA
Baba's partial parity check	10	7-7-7-1	7-1-1
Baba's 6 bits parity check	64	6-6-6-1	6-6-1
Jacob's multiplexer	64	6-6-1	6-4-1

Table 2 The comparison of networks found by SGLA and networks used by other authors.

REFERENCES

[1] K. I. Funahashi, "On the Approximate Realization of Continuous Mappings by Neural Networks," *Neural Networks*, vol. 2, pp. 183-192, 1989.

[2] K. Hornik, "Multilayer Feedforward Neural Networks are Universal Approximators," *Neural Networks*, vol. 2, pp. 359-366, 1989.

[3] M. F. Tenorio and W. T. Lee, "Self-Organizing Network for Optimum Supervised Learning," *IEEE Trans. on Neural Networks*, vol. 1, No. 1, March, 1990.

[4] L. M. Fu, "Analysis of the Dimensionality of Neural Networks for Pattern Recognition," *Pattern Recognition*, vol. 23, No. 10, pp. 1131-1140, 1990.

[5] Y. Hirose, K. Yamashits and S. Hijiya, "Back-Propagation Algorithm Which Varies the Number of Hidden Units," *Neural Networks* Vol. 4, pp. 61-66, 1991.

[6] M. Zak, "Terminal Attractors in Neural Networks," *Neural Networks*, vol. 2, pp. 259-274, 1989.

[7] S. D. Wang and C. H. Hsu "Terminal Attractor Learning Algorithms for Back Propagation Neural Networks," Submitted to International Joint Conf. on Neural Networks, Sigapore, 1991.

[8] N. Baba, "A New Approach for Finding the Global Minimum of Error Function of Neural Networks," *Neural Networks*, vol. 2, pp. 367-373, 1988.

[9] R. A. Jacob, "Increased Rates of Convergence Through Learning Rate Adaptation," *Neural Networks*, vol. 1, pp. 295-307, 1988.