Terminal Attractor Learning Algorithms for Back Propagation Neural Networks

Sheng-De Wang and Ching-Hao Hsu
Department of Electrical Engineering
National Taiwan University, Taipei 10764, Taiwan
e-mail: sdwang@twmoe10.bitnet

Abstract - In this paper, new learning algorithms called "Terminal Attractor Back Propagation" (TABP) and "Heuristic Terminal Attractor Back Propagation" (HTABP) for multilayer networks are proposed. The algorithms are based on the concepts of terminal attractors, which are fixed points in the dynamic system violating Lipschitz conditions. The key concept in the proposed algorithms are the introducing of time-varying gains in the weight update law. The proposed algorithms preserve the parallel and distributed features of neurocomputing, guarantees that the learning process can converge in finite time and find the set of weights minimizing the error function in global provided such a set of weights exists. Simulations are carried out to demonstrate the global optimization properties and the superiority of the proposed algorithms over the standard back propagation algorithm.

1. INTRODUCTION

Multilayer feedforward neural networks have been used to solve many practical problems, such as pattern recognition, signal processing and system modeling. Perhaps the key issue in applying multilayer networks is how to efficiently determine the connection weights. The Back Propagation (BP) learning algorithm [10] is widely used to determine the weights in the paradigm of supervised learning in multilayer networks. However, there are some drawbacks for BP algorithm, for example, the convergence of BP is generally slow and may fall into the local minimum of the error function.

In recent years, many researchers have developed improved versions of BP or alternative learning algorithm for alleviating its shortcoming of slow convergence. For instance, Jacob [1] developed the so-called delta-bar-delta algorithm based on the adaptation of learning rate; Baba [2] proposed a random optimization algorithm for finding the global minimum of the error function of neural networks; Nguyen and Widrow [3] improved the learning speed of a two layer network by choosing appropriate initial weights; Song and Hassoun [4] estimated the hidden targets to speed up the learning; Yang and Guest [5][6] used parabolic neurons and proposed appropriate rules for weight update law; Burrascano and Lucci [7][8] improved the back propagation algorithm by replacing the error function with the Chebyshev norm and modifying the sigmoid function; White [9] applied a multi-dimensional version of Newton's method to find these weights. Although these approaches can improve the convergence speed of BP algorithm, in some cases the convergence is still very slow and may stick to the local minimum of the error function. The stochastic approach proposed in Baba [2] can find the optimal weights at the expense of very long training processes.

We can categorize the learning algorithms according to how they update the connection weights to minimize an error function. Algorithms that update the weights using only local informations are classified as local techniques. All other algorithms are referred to as non-local techniques. For the purposes of conforming to parallel and distributed processing aspect of neurocomputing, we are only interested in local learning algorithms.

This paper presents algorithms based on the concept of terminal attractors to speed up the learning process. In fact, the proposed algorithms guarantee finishing learning in finite time and reaching the global minimum of the error function. Further, the proposed algorithms adhere to the locality constraints, that is, the learning law for weight adjusting based solely on local computations.

2. BASIC DEFINITIONS AND THE BACK ERROR PROPAGATION ALGORITHM

In a multilayer neural network, the i -th neuron in a layer computes a single activation V_i^m for the m -th input pattern, which may in turn serve as input to other neuron. Let u_i^m be the corresponding internal value of the i -th neuron. The standard mathematical relation of a neural network can be described as follows

$$u_j^m = \sum_i T_{ji} V_i^m \quad (2.1)$$

and
$$V_j^m = f(u_j^m) \quad (2.2)$$

where T_{ji} is the connection weight from the i -th unit in a layer to the j -th unit in next layer; $f()$ is the sigmoid

function. In the supervised learning process of the back propagation algorithm, training examples are presented to the network. The weights are to be found such that the differences between actual neuron outputs and desired outputs are minimized. To this end, an error function E can be defined as

$$E = \frac{1}{2} \sum_m \sum_l (\bar{V}_l^m - V_l^m)^2, \quad (2.3)$$

where $\bar{V}_l^m$ is the target output for the l -th neuron corresponding to the m -th pattern.

In order to minimize the error function with respect to weights, the learning algorithm BP updates the weights by a quantity proportional to the steepest descent of E , i.e.,

$$\frac{dT_{ji}}{dt} = - \frac{\partial E}{\partial T_{ji}} \quad (2.4)$$

Applying the chain rule to (2.4) and approximating the above differential equation with a first order difference equation, we get the following BP weight update law:

$$\Delta T_{ji}(n) = \eta \delta_j^m V_i^m, \quad (2.5)$$

where Δ denotes the difference operator, and $\delta_j^m = (\bar{V}_l^m - V_l^m) f_j'(u_j^m)$ for output layer and $\delta_j^m = f_j'(u_j^m) \sum_k \delta_k^m T_{kj}$ for hidden layers. In (2.5), η is a small positive value known as learning rate. A momentum term, denoting as $\alpha \Delta T_{ji}(n)$, is usually included in the update law to provide damping effects and hence reduce the possible oscillations and accelerate convergence. As a result, (2.6) is widely applied rather than (2.5).

$$\Delta T_{ji}(n+1) = \eta \delta_j^m V_i^m + \alpha \Delta T_{ji}(n) \quad (2.6)$$

where α is a constant. BP is a simple and useful algorithm. However, the BP learning process usually takes a long time even for a simple case. This weak point sometimes makes the applications impractical for large set of training examples. Furthermore, it may get into local minima of the error surface. In this paper, we try to solve the above two problems by introducing a time-varying gains into the BP weight update law.

3. TERMINAL ATTRACTORS

The terminal attractor is first introduced by Zak [11]. The idea of a terminal attractor is based upon a violation of the Lipschitz condition at a fixed point. Since the Lipschitz condition is violated, the fixed point becomes a singular solution which envelopes the family of regular solution. Nonlinear dynamic systems satisfying Lipschitz condition have a unique solution for each initial condition and the trajectory of the solution can approach equilibrium points asymptotically, i.e., the trajectory can never reach these points but can approach them as closely as possible. If the system is of ill condition, the time of convergence will be long, especially it may spend a lot of time at the end of transition. Unfortunately, neural network is often like an ill conditioned system [13]. The learning transient takes more time at the later stage than the transient at the beginning. Unlike asymptotical convergence, terminal attractors violate the Lipschitz condition at equilibrium points, and then the solution can reach these points in finite time. This is the fascinating point of terminal attractors. We illustrate the concept of terminal attractors with an example.

Example: Consider the differential equation

$$\dot{x} = -x^k, \quad k > 0.$$

Obviously $x=0$ is the equilibrium point and the Lipschitz condition is

$$\frac{dx}{dx} = -k x^{k-1} = \begin{cases} 0 & k > 1 \\ -1 & k = 1 \\ -\infty & 0 < k < 1 \end{cases} \quad \text{as } x \rightarrow 0$$

It can be seen that the Lipschitz condition is violated for $0 < k < 1$ at $x=0$ and the real part of the eigenvalue of matrix $[\frac{dx}{dx}]$ at $x=0$ is negative (in this case, $[\frac{dx}{dx}]$ is a scalar and its eigenvalue is $-\infty$), so the equilibrium point $x=0$ is stable in this case. Solving (2.8) directly, we get

$$x^{1-k} = (1-k)(c-t),$$

where c is a constant to be determined by initial conditions. Obviously, if $0 < k < 1$ the relaxation time of the solution x will reach the equilibrium point $x=0$ is finite from any positive initial condition, otherwise the transient will take infinite time. The trajectory will stay at $x=0$ because the derivative at $x=0$ is zero.

The terminal attractor had been applied to single layer neural networks [11] and multilayer neural networks [12] by using Lagrange multiplier technique. These approaches update the connection weights indirectly since the Lagrange multipliers should also be solved in the same time. In the next section, we will apply terminal attractors directly to the BP algorithm. It is simple and can reach equilibrium points in finite time theoretically; that is, the learning process of a back propagation network would be completed in finite time.

4. TABP AND HTABP ALGORITHM

By using the chain rule, the time derivative of the error function can be written as

$$\frac{dE}{dt} = (\nabla_T E)' \cdot \frac{dT}{dt} \quad (4.1)$$

where $\nabla_T E$ denotes the vector of $\frac{\partial E}{\partial T_{ji}}$, T is the vector of weight, and the prime ' denotes the matrix transpose. For standard BP method, the evolution of weights is described by the equation

$$\frac{dT}{dt} = -\nabla_T E. \quad (4.2)$$

In this case, the dynamics of the error function is obviously given by

$$\frac{dE}{dt} = -\|\nabla_T E\|^2 \leq 0$$

It can be seen that the error function will asymptotically converge to a local minimum, where the gradient satisfying $\nabla_T E = 0$.

To implement terminal attractors to BP algorithm, we introduce a time-varying gain γ to the weight update law. The key concept of γ is that it will depend on error function E and the derivative of E with respect to weights such that a terminal attractor is formed at the equilibrium points of zero. The time evolution of weights is now given by

$$\frac{dT}{dt} = -\gamma \nabla_T E, \quad (4.3)$$

and γ is chosen to be

$$\gamma = \frac{E^k}{\|\nabla_T E\|^2} \quad 0 < k < 1. \quad (4.4)$$

The dynamics of the error function becomes

$$\begin{aligned} \frac{dE}{dt} &= -(\nabla_T E)' \cdot \left(\frac{E^k}{\|\nabla_T E\|^2} \nabla_T E \right) \\ &= -E^k. \end{aligned} \quad (4.5)$$

By the characteristic of terminal attractors, the error function E will reach zero in finite time. Discretizing (4.3) with the first order approximation, we have

$$T(n+1) = T(n) + \eta \gamma \nabla_T E(n) \quad (4.6)$$

The momentum term used in BP algorithm is not necessary in the above equation. In (4.4), the gain γ is determined by E and $\|\nabla_T E\|$. Unfortunately, the computation of the value of $\|\nabla_T E\|^2$ need all weights in the network. Hence, the computation of γ depends on the global information. This is a drawback because the system is difficult to operate in parallel. Fortunately, we can easily extend (4.4) to a parallel version which can update weights using only local information.

To this end, assume there are p processing elements in a neural network, then, (4.3) can be modified as

$$\frac{dT^c}{dt} = -\gamma_c \nabla_{T^c} E \quad c = 1, 2, \dots, p \quad (4.7)$$

where T^c denotes the weight vector pertaining to processing element c and γ_c is the adaptive gain used in processing element c . The gains γ_c can be formulated by

$$\gamma_c = \frac{E^k}{p \|\nabla_{T^c} E\|^2} \quad c = 1, 2, \dots, p. \quad (4.8)$$

Substitute γ_c to (4.7), then (4.1) becomes

$$\begin{aligned} \frac{dE}{dt} &= \sum_{c=1}^p \left(\frac{\partial E}{\partial T^c} \right) \left(-\gamma_c \frac{\partial E}{\partial T^c} \right) \\ &= -E^k. \end{aligned} \quad (4.9)$$

Clearly, the gains γ_c are determined by local information of the network only and the most important is that the system still has the properties of terminal attractors.

Some properties of γ_c are discussed now. The variable γ_c plays an important role in our algorithm: a) they provide an adaptive learning rate to speed up the learning and makes learning converge in finite time; b) they makes a chance to escape from local minima. We explain these further as follows.

a) The difference between (4.2) and (4.7) is the gain γ_c . If $\gamma_c > 1$, learning is accelerated, otherwise learning becomes slower as compared to the standard BP algorithm. No matter what value of γ_c is, the transient in the solution of (4.7) will finish in finite time. In our experience, γ_c can speed up the training over BP by 3 orders in some cases.

b) The learning process of BP would fall into a local minimum in case of the gradient $\nabla_T E$ approaches zero while error function E is not in its global minimum. From (2.5), we can see that the gradient $\nabla_T E$ approaches zero if one of following three conditions satisfies:

- 1) the error between the neural actual output and the desired output approaches zero.
- 2) the derivative of the sigmoid function approaches zero, i.e., u_i^m is too large so that the output of neuron is saturated.
- 3) the value V_i^m at the input of the connection approaches zero.

It is obvious that when the gradient $\nabla_T E$ approaches zero, the components of error function are not necessary zero. On the other hand, in the proposed terminal attractor back propagation algorithm as described by (4.7) and (4.8), where the weight updates are governed by γ_c , defined by denominator $\|\nabla_{T^c} E\|^2$ and numerator E^k , as well as the gradient $\nabla_T E$. When the system falls into a local minimum, the denominator of γ_c approaches zero but the numerator does not, so γ_c becomes very large. At last the system will have the chance to jump from the local minimum and continue the learning process. If there exists a set of weights satisfying $E=0$, from (4.9) we have $E \geq 0$, $\frac{dE}{dt} \leq 0$. By Liapunov method, we guarantee to find the weights, which satisfy $E=0$. That is, we can find the optimal set of weights that globally minimize the error function.

Another heuristic algorithm(HTABP) which we proposed is based on (4.6), where the time-varying parameter γ_c is further constrained to be greater than or equal to 1. That is, in case of $\gamma_c \leq 1$, we set $\gamma_c = 1$ and switch update law to (2.6). This constraint is imposed for the case the norm of the gradient $\|\nabla_{T^c} E\|$ is very large. In this case, the value of γ_c may be quite small because γ_c is inverse proportional to the square of the norm of the gradient, $\|\nabla_{T^c} E\|^2$.

In sum, we claim that the proposed TABP algorithm can converge to global minimum within finite time and the HTABP learning algorithm can converge within finite time with faster speed to the global minimum of error functions if the optimal set of weights exists. It should be noted that the convergence speed of HTABP algorithm is always superior to the BP algorithm, since the BP algorithm is a special case of HTABP with $\gamma_c = 1$, while the HTABP algorithm with $\gamma_c \geq 1$. Our algorithm is shown in the following.

5. COMPUTER SIMULATIONS

In this section, several experiments are carried out through the computer simulation using 4 training examples to compare the performance of HTABP algorithm with that of TABP and BP method. These examples are chosen because they are used popularly in the literature as learning benchmarks. They include 1) XOR, 2) coordinate transformation, 3) a sinusoidal function, 4) issues concerning finding the global minimum of the error function. In each case, a network with 2 hidden layers is used and each weight is updated once after all input patterns have been

presented to the network for learning. The network is divided into p subsystem and implemented by p processing elements. Each subsystem includes the neurons and all weights connecting to these neurons. The parameters η and α for both HTABP and BP methods are 0.01 and 0.5, respectively. The exponent k in (4.4) and (4.8) is choosed to be $1/3$ and the sigmoid functions used is $\tanh(x)$, for all simulations. .

TABP ALGORITHM

- Define NE to be the normalized error function, i.e., $NE = E/(\text{pattern no.} * \text{output no.})$
- Assume there are p processing elements.
- Initialize all variables: weights, input patterns and targets. Input parameters ErrorTolerance, Maxpass, η . All initial weights are assumed to be distributed uniformly in $[-0.5, 0.5]$, and let pass = 0.
- while (NE > ErrorTolerance and pass < Maxpass) do
 - {
 - Calculate forward to find the network outputs, V_1^m
 - Calculate backward to find δ_j^m and γ_c , $c=1, 2, 3, \dots, p$.
 - $T_{ji}^c(n+1) = T_{ji}^c(n) + \eta \gamma_c \delta_j^m V_i^m$
 - pass = pass + 1.
 - }

HTABP ALGORITHM

- Define NE to be the normalized error function, i.e., $NE = E/(\text{pattern no.} * \text{output no.})$
- Assume there are m processing elements.
- Initialize all variables: weights, input patterns and targets. Input parameters ErrorTolerance, Maxpass, η and α . All initial weights are assumed to be distributed uniformly in $[-0.5, 0.5]$, and let pass = 0.
- while (NE > ErrorTolerance and pass < Maxpass) do
 - {
 - Calculate forward to find the network outputs, V_1^m
 - Calculate backward to find δ_j^m and γ_c , $c=1, 2, 3, \dots, p$.
 - if $\gamma_c > 1$ update weights by

$$T_{ji}^c(n+1) = T_{ji}^c(n) + \eta \gamma_c \delta_j^m V_i^m$$
 - else update weights by

$$T_{ji}^c(n+1) = T_{ji}^c(n) + \eta \delta_j^m V_i^m + \alpha \Delta T_{ji}^c(n-1)$$
 - 1)
 - pass = pass + 1.
 - }

1) XOR

The first example is the most famous problem XOR, a simple and high nonlinear example. The structure of network used is 2-2-2-1, i.e., a multilayer network with 2 input units, 1 output unit and 2 hidden layers each with 2 units. The parameters used are $p=1$, ErrorTolerance = 0.0001, MaxPass= 50000. The error transition of the simulation is shown in Fig 1, where we can see the excellent performance of TABP and HTABP.

2) Coordinate transformation

Transforming the Cartesian coordinate to the polar coordinate is important in engineering problems. Unlike the XOR example is a digital example, the coordinate transformation example has analog inputs and outputs. The network is a 2-10-10-2 network. Table 1 shows simulation results, where the parameters used are ErrorTolerance = 0.00001 and MaxPass = 50000. Obviously, the HTABP algorithm has the best performance. The next is TABP. The gains γ_c have successfully accelerated the converge rate as can be seen from Table 1.

3) A sinusoidal function

The third example is a continuous nonlinear mapping: $h(x) = 0.5 \sin(2\pi x/10) + 0.5 \sin^2(2\pi x/20)$. We uniformly choose 20 points in the interval $[0, 20]$ and normalize them, then present these points as input patterns into a 1-15-15-1 network. The parameters used are ErrorTolerance = 0.00001 and MaxPass = 50000. The simulation results are shown in Table 2. It is noted that the case 3 for HTABP takes much more learning cycles than other cases because it switches to BP algorithm all the time at the last stage of training. Even this case takes so many time to converge, but the converge rate is still faster than BP algorithm. One transition of normalized error function is shown in Fig 2. The training using TABP or HTABP algorithm converge at last after a number of oscillations in transient, which successfully help it escape from the local minimum of the error surface. These oscillations are resulted from the time-varying properties of γ_c . Sometimes the abrupt change of γ_c between cycles is like a narrow width pulse. This is a worrying problem in numerical method, however, it can prevent the learning process from sticking at local minima.

It is interesting to note that the computation of γ_c in both the TABP and the HTABP need only the local information in each processing element. However, if we use only one processing element (that is $p=1$) to

implement a network, the values of γ_c are now dependent on the global informations. The results in Table 1 and 2 show that for TABP algorithm the more the information received by γ_c the faster the convergence rate is.

4) Finding global minimum

The last example is about finding the global minimum of an error function. We choose XOR as the training example. 25 simulations are done for the BP, the TABP and the HTABP algorithms for 25 different sets of initial weights. From Table 3, we can find that the BP algorithm fails to reach the solution 4 times during the 25 experiments, but only once for the TABP algorithm and none for the HTABP algorithm. After analysis of simulation data, we find that the BP algorithm is failed due to the problem of sticking to a local minimum, while the TABP algorithm is failed due to the numerical problem of large variations of γ_c . We dump the connection weights and find all values are NaN (Not a Number), that is, they are out of value. It also can be seen from (4.8) that γ_c will become very large when system falls into a local minimum, so as the system has the chance to jump from the local minimum. If γ_c is so large, it may lead to overflow due to the representation limit of digital computer and as a results, the training will fail.

6. DISCUSSING AND CONCLUSION

This paper has introduced new approaches for supervised learning in multilayer networks. This method is based upon the concept of terminal attractors - the fixed points with violation of Lipschitz condition. The TABP method is based on the BP method and the introduction of time-varying gains γ_c . The value of γ_c can be calculated easily from original back propagation process. The calculation is simple and only local information is needed. The most important two things are that it can converge in finite time and that it can find the global minimum theoretically. Another algorithm called HTABP are combined with BP and TABP algorithm. The HTABP algorithm is guaranteed to be faster than the BP algorithm and preserve the properties of the TABP algorithm. The good performances are supported by simulations as well as mathematical analysis. On the other hand, γ_c may be very sensitive in transient in our experience, so a small learning rate η may be safe for learning. We suggest η should be less than 0.05 for a complex system.

REFERENCES

- [1] R. A. Jacob, "Increased Rates of Convergence Through Learning Rate Adaptation," *Neural Networks*, vol. 1, pp. 295-307, 1988.
- [2] N. Baba, "A New Approach for Finding the Global Minimum of Error Function of Neural Networks," *Neural Networks*, vol. 2, pp. 367-373, 1988.
- [3] D. Nguyen and B. Widrow, "Improving the Learning Speed of 2-Layer Neural Networks by Choosing Initial Values of the Adaptive Weights," *Proc. Int. Joint Conf. Neural Networks*, San Diego, California, 1990, pp. III-21 - III-26.
- [4] J. Song and M. H. Hassoun, "Learning with Hidden Targets," *Proc. Int. Joint Conf. Neural Networks*, San Diego, California, 1990, pp. III-93 - III-98.
- [5] H. Yang and C. C. Guest, "High Order Neural Networks with Reduced Numbers of Interconnection Weights," *Proc. Int. Joint Conf. Neural Networks*, San Diego, California, 1990, pp. III-281 - III-286.
- [6] H. Yang and C. C. Guest, "Linear Discriminants, Logic Functions, Back-propagation, and Improved Convergence," *Proc. Int. Joint Conf. Neural Networks*, San Diego, California, 1990, pp. III-287 - III-292.
- [7] P. Burrascano and P. Lucci, "A Learning Rule in the Chebyshev Norm for Multilayer Perceptrons," *Proc. Int. Joint Conf. Neural Networks*, San Diego, California, 1990, pp. III-81 - III-86.
- [8] P. Burrascano and P. Lucci, "Smoothing Backpropagation Cost Function by Delta Constraining," *Proc. Int. Joint Conf. Neural Networks*, San Diego, California, 1990, pp. III-75 - III-80.
- [9] R. H. White, "The learning rate in back-propagation systems: an application of Newton's method", *Proc. Int. Joint Conf. Neural Networks*, San Diego, California, 1990, pp. I-679 - I-684.
- [10] D. E. Rumelhart, G. E. Hinton and R. J. Williams, "Learning Internal Representations by Error Propagation," *Parallel Distribute Processing: Exploration in the Microstructure of Cognition*, vol. 1, Chapter 8, pp. 318-362, MIT Press, Cambridge, Massachusetts, 1986.
- [11] M. Zak, "Terminal Attractors in Neural Networks," *Neural Networks*, vol. 2, pp. 259-274, 1989.
- [12] M. Zak, "Neural Learning of Constrained Nonlinear Transformations," *Computer*, June, pp. 67-76, 1989.
- [13] A. J. Owens and D. L. Filkin, "Efficient Training of the Back Propagation Network by Solving a System of Stiff Ordinary Differential Equation," *Proc. Int. Joint Conf. Neural Networks*, Washington, DC, 1989, pp. III-381 - III-386.

Algorithm	No. of block	Iterations
BP		50000*
TABP	1	9125
	2	8931
	4	10122
	6	11852
	10	28622
HTABP	1	3940
	2	9566
	4	2514
	6	6994
	10	9531

* The normalized error function is 0.0000137 when learning stops.

Table 1. Simulation results for coordinate transformation.

Algorithm	No. of block	Iterations
BP		50000*
TABP	1	10992
	2	14247
	4	15785
	8	13776
	16	18080
HTABP	1	5897
	2	8119
	4	9732
	8	50000**
	16	7226

* The normalized error function is 0.000169 when learning stops.

** The normalized error function is 0.000011 when the learning stops.

Table 2. simulation results for mapping a sinusoidal function.

Algorithm	Simulations attempted	Simulations reached solution
HTABP	25	25
TABP	25	24
BP	25	21

Table 3 Simulation results of finding global minimum

Fig. 1 Error transitions of training XOR examples

Fig. 2 Error transitions of training a sinusoidal function