FAST TRAINING OF LARGE TRAINING SETS IN BACK PROPAGATION NETWORKS BY SELECTIVE PRESENTATION

K.LAKSHMI NARAYANA ATUL NEGI
K.NARAYANA MURTHY

Artificial Intelligence Laboratory
University of Hyderabad
Hyderabad 500 134, India
fax-91-842-253145, telex-425-2050 UHYD IN
e-mail : knm-cs@uohyd.ernet.in

Abstract

Recently many training schemes have been proposed for speeding up the Back Propagation algorithms. In this paper a novel method is proposed. In our method selective and repetitive step-by-step enlargement of a preliminary subset of a large training set is made. This technique is applied on a training set of size 1002 for learning the inverse dynamics of a two-link planar manipulator.

keywords: *Artificial Neural Networks, Back Propagation Networks, Learning, Robotics, Inverse dynamics.*

1 Introduction

Multilayer Back Propagation Network (BPN) has become an extremely important and popular Artificial Neural Network (ANN) model because of its power and versatility in learning a large variety of non-linear functions. It is one of the most useful and well-studied ANN algorithm in the present arena of Neural Network research.

Training methods for multilayer perceptrons were first developed by Paul Werbos and independently by David Parker. The most popular BPN algorithm was rediscovered by Rumelhart and others [5]. The basic elements of a typical BPN are a Mc Culloch & Pitts linear filter with a sigmoid output function and Rosenblatt coupled error correction. It is an example of a multi-layered feed-forward network.

The learning procedure involves the presentation of a set of pairs of input and output patterns. The network first uses the input vector to produce its own output vector and then compares this with the desired output or target vector. If there is no difference, no learning takes place. Otherwise, the weights are changed to reduce the difference. The

algorithm is dealt in detail in [5]. In this paper we first review some methods for speeding up BPN learning. In section 3 we present our proposed method. Finally we conclude with an application of the proposed method to the inverse dynamics of a two-link robot.

2 Fast Learning BPNs

The BPN is a powerful mapping network which has been successfully applied to a wide range of problems. However, a major limitation of BPN is that its convergence is slow. This is a serious limitation of performance in real world applications. Various approaches have been proposed in the recent past to improve the convergence rate of the networks. One of the alternatives is to adopt ANN algorithms with similar classification performance as the BPN but with faster training capabilities. Examples of such networks are counter propagation networks. Another way of improving the speed is by using a modified activation function. Examples of such approaches are Fahlman [5], Jacobs [3] and Samad [6]. In the present approach a heuristic technique is used to enhance the convergence speed. The method is based on using a selective sequence of presentation of the training set.

Rumelhart et al [5] added to the basic BPN algorithm, a momentum term α, so as to improve the learning speed. The addition of the momentum term filters out the high frequency variations of the error surface in the weight space. In most of their simulations they recommend the value of α as 0.9 and state that the system can learn much faster with larger values of learning rate and momentum term.

Fahlman [1] proposed some techniques to enhance the learning rate in his "Quick Propagation" algorithm. In this the problem of 'flat spots', where the derivative of the sigmoid function approaches zero is considered first. In such cases the output of the sigmoid is replaced by either a constant value or a random number between 0.0 and 1.0. An improvement of 20% on his class of bench-marks in encoder/decoder problems, was reported. In other classes of problems, information about the rate of gradient descent in the previous epoch is used to update the weights. This method when applied iteratively, was found to be quite effective.

Delta-bar-Delta algorithm of Jacobs [3] is another technique for BPN speed-up. Jacobs suggested four heuristics: (i) Each weight has its own learning rate. (ii) These learning rates are varied based on the error surface information. (iii) When the error surface gradient has the same sign for many iterations, the corresponding learning rate is increased, since it indicates that a minimum lies ahead. (iv) When the error surface gradient flips signs for several consecutive time steps, the learning rate is decreased, since this indicates that a minimum is being jumped over. These parameters are specified by the user for modifying the learning rates in this scheme. Though these heuristics have shown considerable speed-up, there are certain drawbacks. Simulations by Jacobs himself showed that the technique is very sensitive to small variations in the values of its parameters. It is also difficult to preselect a desired set of parameters.

Samad's [6] variation aims at compensating for not only present weight changes but future expected changes as well. He used a method called "expected source values".

However, Godhwani,K.K., [2], found that Samad's variation did not appear to converge for low value of learning rates and took a large number of training epochs, as compared to Fahlman's and Jacob's methods.

3 Proposed Method

There are certain tasks in the real world where the amount of knowledge to be learnt by a network is huge. In such a situation, the state space to be learnt by the network is very large. Hence the exemplar set is also very large. The network needs a long time for training this state space. In the conventional method of training the network, all the patterns in the exemplar set are presented to the network during each training epoch.

Consider an exemplar set $\mathcal{U}$, containing $n = \mid \mathcal{U} \mid$ (the cardinal number of the set $\mathcal{U}$) input-output patterns. Let 't' be the time taken for the forward and backward propagation of a sample training pattern. If N training epochs are performed, the total time T_{con} taken for performing N epochs in the conventional method would be

$$T_{con} = n \times N \times t \tag{1}$$

In the present method of training, a subset of the exemplar set, which we call the "*training set*", is initially presented to the network and certain number of training epochs are performed. Keeping the structure of network the same, but using the updated weight matrix of the last epoch, some more patterns are added to the subset from the exemplar set and the training is continued for some more epochs. This process of enlarging the initial training set is continued repeatedly until the training set equals the original exemplar set.

Let $\mathcal{U}$ be the exemplar set. Let the first subset be A_1. The process of enlargement is performed for 'p' times. The training sets are $A_1, A_2, ..., A_{p+1}$, where

$$A_1 \subset A_2 \subset A_3 \subset ... \subset A_{p+1} \quad \& \quad A_{p+1} = \mathcal{U} \tag{2}$$

Let $n = \mid \mathcal{U} \mid$, $n_1 = \mid A_1 \mid$, $n_2 = \mid A_2 \mid$,..., $n_{p+1} = \mid A_{p+1} \mid$
Let N_1, N_2 ,...,N_{p+1} be the number of epochs performed using the training subsets A_1,A_2,...,A_{p+1} respectively.

The total time T for performing the training will now be

$$T = (n_1 N_1 + n_2 N_2 + ... + n_{p+1} N_{p+1}) \times t$$

$$\Rightarrow \quad T = t \times \sum_{i=1}^{p+1} (n_i \times N_i) \tag{3}$$

Total number of epochs by conventional training method = N
Total number of epochs by the present method = $\sum_{i=1}^{p+1} N_i$

The values of N_i are chosen such that ,

$$\sum_{i=1}^{p+1} N_i = N$$

and the performance achieved is compared with that of the conventional method. Since

$$n_1 < n_2 < ... < n_{p+1} \quad \text{and} \quad n_{p+1} = n$$

we have from (1) and (3),

$$T < T_{con}$$

Thus this technique will require lesser time than the conventional method of training.

We select A_i by a uniform sampling of the entire training space. The weight values learnt after training for A_i are a good initial set of training weights for the superset A_{i+1}. Because of the good initialization the output values are in a closer range to the desired output values, requiring lesser weight adjustments. Hence the training time is reduced.

4 Robot Simulator

The dynamic equation for an N-link manipulator is

$$H(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) + K(q)^T k = \tau \tag{4}$$

where,

- $H(q)$ is an $N \times N$ symmetric, non-singular Moment of Inertia matrix
- $C(q,\dot{q})$ is an $N \times N$ matrix specifying centrifugal and Coriolis effects
- $G(q)$ is an $N \times 1$ vector specifying the effects due to gravity
- $K(q)$ is a $6 \times N$ Jacobian matrix specifying the torques created at each joint due to external forces and moments exerted on the final link. $K(q)^T$ indicates its transpose.
- k is a 6×1 vector of external moments and forces exerted on the final link The first three components of this vector are the external forces on N and the last three, moments exerted on N.
- τ is a $N \times 1$ vector of torques at each joint.
- q is an $N \times 1$ vector of joint positions
- $\dot{q}$ is an $N \times 1$ vector of joint velocities
- $\ddot{q}$ is an $N \times 1$ vector of joint accelerations.

When $\ddot{q} = 0$,

$$C(q, \dot{q}) + G(q) + K(q)^T k = \tau$$

This torque is called the "*bias torque*" b. Hence

$$H(q)\ddot{q} = (\tau - b) \tag{5}$$

The bias vector b is computed by setting q, $\dot{q}$ & k to their current values and letting $\ddot{q} = 0$. The elements of the matrix $H(q)$ can be obtained from any of the methods in [7]. For any given input torque τ the simulator computes $\ddot{q}$ using (5).

In the present work a two-link planar manipulator is considered. It is assumed that all the mass exists as a point mass at the distal end of each link. The dynamic equations for this manipulator can be easily obtained, manually by using the standard Newton-Euler manipulator dynamics formulation. Then,

$$q = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix} \dot{q} = \begin{bmatrix} \dot{\theta_1} \\ \dot{\theta_2} \end{bmatrix} \ddot{q} = \begin{bmatrix} \ddot{\theta_1} \\ \ddot{\theta_2} \end{bmatrix} \tau = \begin{bmatrix} \tau_1 \\ \tau_2 \end{bmatrix}$$

Assuming that there are no external forces or moments, (4) reduces to

$$\tau = H(\Theta)\ddot{\Theta} + C(\Theta, \dot{\Theta}) + G(\Theta) \tag{6}$$

where

$$H(\Theta) = \begin{bmatrix} l_2^2 m_2 + 2 l_1 l_2 m_2 c_2 + l_1^2 (m_1 + m_2) & l_2^2 m_2 + l_1 l_2 m_2 c_2 \\ l_2^2 m_2 + l_1 l_2 m_2 c_2 & l_2^2 m_2 \end{bmatrix}$$

$$C(\Theta, \dot{\Theta}) = \begin{bmatrix} -m_2 l_1 l_2 s_2 \dot{\theta}_2^2 - 2 m_2 l_1 l_2 s_2 \dot{\theta}_1 \dot{\theta}_2 \\ m_2 l_1 l_2 s_2 \dot{\theta}_1^2 \end{bmatrix}$$

$$G(\Theta) = \begin{bmatrix} m_2 l_2 g c_{12} + (m_1 + m_2) l_1 g c_1 \\ m_2 l_2 g c_{12} \end{bmatrix}$$

where l_1 & l_2 are the link lengths, m_1 & m_2 are the link masses, θ_1 & θ_2 are the joint angles, $\dot{\theta}_1$ & $\dot{\theta}_2$ are the joint angular velocities, $\ddot{\theta}_1$ & $\ddot{\theta}_2$ are the joint angular accelerations, $c_1 = \cos(\theta_1)$, $c_2 = \cos(\theta_2)$, $c_{12} = \cos(\theta_1 + \theta_2)$, $s_1 = \sin(\theta_1)$ & $s_2 = \sin(\theta_2)$.

An input torque pair $\{\tau_1, \tau_2\}$ is given, the corresponding accelerations $\{\ddot{\theta}_1, \ddot{\theta}_2\}$ at the given initial positions $\{\theta_1, \theta_2\}$ and initial velocities $\{\dot{\theta}_1, \dot{\theta}_2\}$ are computed by the simulator, using (6).

5 Learning

A BPN is trained to learn the inverse dynamics of the two-link manipulator. Learning involves the following stages:

- Exemplar Set Generation
- Training

5.1 Exemplar set Generation

A set of input torque values, initial positions and initial velocities are given to the simulator and the corresponding acccelerations generated at the respective joints of the manipulator are computed. An exemplar set is generated according to the following configuration:

input vector $\quad \theta_1,\ \dot{\theta}_1,\ \ddot{\theta}_1,\ \theta_2,\ \dot{\theta}_2,\ \ddot{\theta}_2$

desired output vector : $\tau_{d1}(=\tau_1),\ \tau_{d2}(=\tau_2)$.

The ranges selected for various joint variables are as follows:

$$\theta_1 = 0 - 0.5\ rad. \qquad \theta_2 = 0 - 0.5\ rad.$$
$$\dot{\theta}_1 = 0 - 1\ rad./sec \qquad \dot{\theta}_2 = 0 - 1\ rad./sec$$

Based on a few tests using the inverse dynamic model, a set of input torques in the range

$$\tau_{d1} = 14 - 18\ Nm \qquad \tau_{d2} = 4 - 8\ Nm$$

are selected. The joint positions of each link are incremented by 0.125 rad, starting from 0 rad. The joint velocities of each link are incremented by 0.25 rad/sec, starting from 0 rad/sec. So a set of $5 \times 5 \times 5 \times 5 = 625$ combinations are obtained. The torques are incremented by 0.1 Nm starting from 14 Nm and 4 Nm of the first and second links respectively. Thus 41 different combinations are obtained for each link. Giving these as input torque commands the corresponding accelerations generated in the links are computed. The total number of accelerations computed would be

$$5 \times 5 \times 5 \times 5 \times 41 \times 41 = 1050625$$

Out of this large number, we selected 1002 patterns restricting the accelerations to the range -2 to $+2$ rad/sec^2. Thus an exemplar set of size 1002 is formed.

5.2 Training

The inverse-dynamics problem of a robotic manipulator is stated as: *Given the initial positions, velocities and accelerations of the respective links of the manipulator, to compute the torques that are to be given at the joints.* Fig. 1 shows the setup. Hence, the network learns the following rule:

> *To develop an acceleration of $\ddot{\theta}_1$ and $\ddot{\theta}_2$ in the links, with the given initial positions θ_1 and θ_2 and initial velocities $\dot{\theta}_1$ and $\dot{\theta}_2$, torques equivalent to the output of the network i.e., τ_{n1} and τ_{n2} are to be given to the robot.*

Three networks of different topologies 6-4-2, 6-8-2, 6-13-2 (figures correspond to input layer nodes, hidden layer nodes and output layer nodes respectively) trained by both the conventional and proposed methods for the above rule. After the training, 10 test patterns are randomly selected (some from the exemplar set and some outside the exemplar set), and the network is tested for generalization.

The error for a test pattern = network output − actual output

The rms error of the ten test patterns is chosen as the criterion for comparing the performance of the two methods.

The results obtained are as shown in table 1.

6 Conclusions

From table 1, we see that the proposed heuristic has shown considerable speed up compared to conventional method. For the considered problem a speed up of nearly 50 % is obtained. In terms of the rms error the proposed method is comparable or better than the conventional method.

As compared to other speed-up techniques, which rely on the exact values of critical parameters, the proposed method is not very sensitive to actual values of parameters like, p, N_i and n_i. This method shows a lot of promise. More theoretical and experimental studies are being conducted.

References

[1] Fahlman,S.E., "An empirical study of learning speed in back propagation neural networks", CMU. Technical report, CMU-CS-86-162., June, 1988.

[2] Godhwani,K.K., "Back propagation variations : Comparison for optical character recognition", University of Hyderabad, M.tech thesis., January, 1993.

[3] Jacobs,R.A., "Increased rates of convergence through learning rate adaption", Neural Networks, vol. 1., pp 295-298., 1988.

[4] Lakshmi Narayana,K., "Learning the dynamic behavior of a robotic manipulator using neural network techniques", University of Hyderabad, M.Tech thesis., July, 1992.

[5] Rumelhart,D.E., Hinton,G.E., & Williams,R.J., "Learning internal representations by error propagation", in Rumelhart,D.E. and Mc Clelland (Eds.)., Parallel Distributed Processing : Explorations in the micro-structures of cognition, I., pp318-362., Cambridge, MA : MIT press, 1988.

[6] Samad,T., "Back propagation with expected source values", Neural Networks, vol.4., pp615-618., 1991.

[7] Walker,M.W., Orin,D.E., "Efficient dynamic computer simulation of robotic mechanisms", Trans. of the ASME Journal of Dynamic Systems, Measurements and Control, vol. 101, pp187-192, September, 1979.

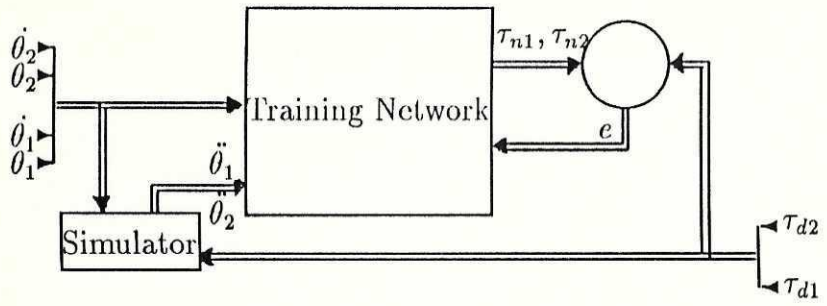

Fig. 1 The initial angular positions θ_1, θ_2, velocities $\dot{\theta}_1, \dot{\theta}_2$ and desired torque values τ_{d1}, τ_{d2} are given to the Simulator. It computes the accelerations $\ddot{\theta}_1, \ddot{\theta}_2$ These accelerations are given to the training network along with the initial positions and velocities. These torques computed by the network τ_{n1} and τ_{n2} are compared with the desired torques and the error e between these is backpropagated. The weights are modified according to the algorithm in [5]

Table 1

	Conventional Method			Proposed Method		
p	–			4		
Size of the Set	$n = 1002$			$n_1 = 50; n_2 = 100; n_3 = 400;$ $n_4 = 800; n_5 = 1002$		
No. of Epochs	150			$N_1 = N_2 = N_3 = N_4 = N_5 = 30;$ $\sum_{i=1}^{p+1} N_i = 150$		
Total time taken	$T_{con} = (1002 \times 150 \times t)$ $= 1,50,000t$			$T = (50 \times 30 + 100 \times 30 +$ $400 \times 30 + 800 \times 30$ $+1002 \times 30) \times t$ $= 70,560t (\approx T_{con}/2)$		
Network Topology	$6-4-2$	$6-8-2$	$6-13-2$	$6-4-2$	$6-8-2$	$6-13-2$
rms error	0.0554	0.1039	0.0872	0.0581	0.0659	0.0824

NOTES:

1. In both cases learning rate of 0.15 and a momentum value of 0.9 were used in the Back Propagation Algorithm as stated by Rumelhart et al.[5]

2. 'p' stands for the number of enlargements