# IJCNN-90-WASH DC

# International Joint Conference on Neural Networks

January 15-19, 1990
Omni Shoreham Hotel
Washington, DC

## Volume I
Theory Track
Neural and Cognitive Sciences Track



co-sponsored by the
International Neural Network Society
and the
Institute of Electrical and Electronics Engineers, Inc.

LEA LAWRENCE ERLBAUM ASSOCIATES, PUBLISHERS
Hillsdale, New Jersey Hove and London

# Extrapolatory Methods for Speeding Up the BP Algorithm

Hasanat M. Dewan
Department of Computer Science
Rutgers University
New Brunswick, NJ 08903
dewan@paul.rutgers.edu

Eduardo D. Sontag
SYCON-Center for Systems and Control
Rutgers University
New Brunswick, NJ 08903
sontag@fermat.rutgers.edu

August 8, 1989

**Abstract**

We describe a speedup technique that uses extrapolatory methods to predict the weights in a Neural Network using Back Propagation (BP) learning. The method is based on empirical observations of the way the weights change as a function of time. We use numerical function fitting techniques to determine the parameters of an extrapolation function and then use this function to project weights into the future. Significant computational savings result by using the extrapolated weights to *jump* over many iterations of the standard algorithm, achieving comparable performance with fewer iterations.

## 1 Introduction

In this note we describe some extrapolation techniques that appear to speed up convergence in backpropagation (BP). Numerical analysis techniques are often used in order to make BP more efficient than straightforward gradient descent, and recently it has been proposed that stiff ODE solvers be used instead of discrete approximations [1]. Our remark here is that *in addition* to these techniques one may be able to exploit the particular form of the differential equation being solved (or its discretization). More precisely, if one uses a sigmoidal response $\frac{1}{1+e^{-x}}$ for neurons, then a rough and nonrigorous analysis suggests that weights tend to grow logarithmically after many iterations, while they tend to behave as $1/t$ for intermediate values of the number of iterations, $t$. The logarithmic asymptotic behavior is suggested by an approximation of the differential equations [2], while the form $1/t$ is apparent from empirical observations of the way the weights change as a function of time. We use these observations as a basis of a speedup technique that uses extrapolatory methods to predict the weights in a network at a future time, given the weights up to the present. By extrapolating the weights, it is possible to economize on the iterations required by BP before an acceptable set of weights result. We use the general form

$$w(t) = a + b/t + c\log t$$

and variants where either $b$ or $c$ are forced to be zero. The parameters are fit via least squares techniques, and this function is then used to predict future weights. We then feed the projected weights back into the BP simulator and continue iterating. The phases of extrapolation and iteration are alternated until a satisfactory set of weights are obtained.

For simplicity, we base our experiments on a standard BP simulator, but the same technique could be used with any variants such as those using stiff ODE solvers. Although this work is empirical in nature, the simulation results are very encouraging, frequently affording considerable savings in computation time.

## 2 Weights as a Temporal Function

If the growth of the weights follow a logarithmic trend, given by the equation $w(t) = a + b \log t$ where $a$ and $b$ are constants and $t$ represents time or the number of iterations, then for large $t$ the expression $t(w(t+1) - w(t))$ would have to approach a constant since

$$t(w(t+1) - w(t)) = t\frac{w(t+1) - w(t)}{(t+1) - t} \approx tw'(t) = t\frac{b}{t} = b$$

On the other hand, if the hyperbolic function $w(t) = a + \frac{b}{t}$ approximates the weights, then the expression $t^2(w(t+1) - w(t))$ should approach some constant for large t, since

$$t^2(w(t+1) - w(t)) = t^2\frac{w(t+1) - w(t)}{(t+1) - t} \approx t^2 w'(t) = t^2\frac{-b}{t^2} = -b$$

To verify these possibilities, we set up a 2-2-1 (2 input, 2 hidden, 1 output unit) network to learn the XOR problem. The BP algorithm was allowed to run for some time after the network classified the four inputs for XOR correctly. Any output unit is considered to have classified correctly if the desired output is 1 and the activation is greater than 0.5, or if the desired output is 0 and the activation is less than 0.5. Some typical graphs for the products mentioned above are shown in fig. 1 as a function of $t$.

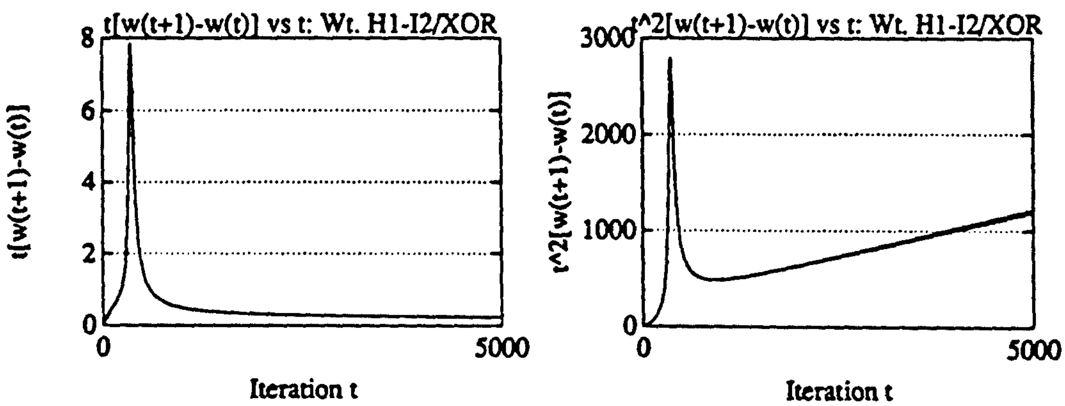


Figure 1: Growth of Weights may be a Log or Combined Hyperbolic-Log Function

It appears from the graphs in fig. 1 that the product *t(w(t+1)-w(t))* approaches some constant value as t becomes large, hence the growth of the weights may indeed be logarithmic. However, the product $t^2(w(t+1) - w(t))$ is asymptotically a straight line. Thus, for some constants B and C, $t^2 w'(t) \approx B + Ct$. Dividing by $t^2$ and integrating both sides, we get $w(t) = A + B/t + C \log t$. Thus the weights seem to follow a combined hyperbolic-logarithmic evolution. Near zero, this is mostly hyperbolic, while for large t it is logarithmic.

In fig. 2 we show typical weight curves from the XOR example, superposed with the hyperbolic-logarithmic functions that approximate them. The actual data is shown in solid lines, while the functions are shown in dashed lines. It is easy to see that the functions approximate the actual weights quite closely.

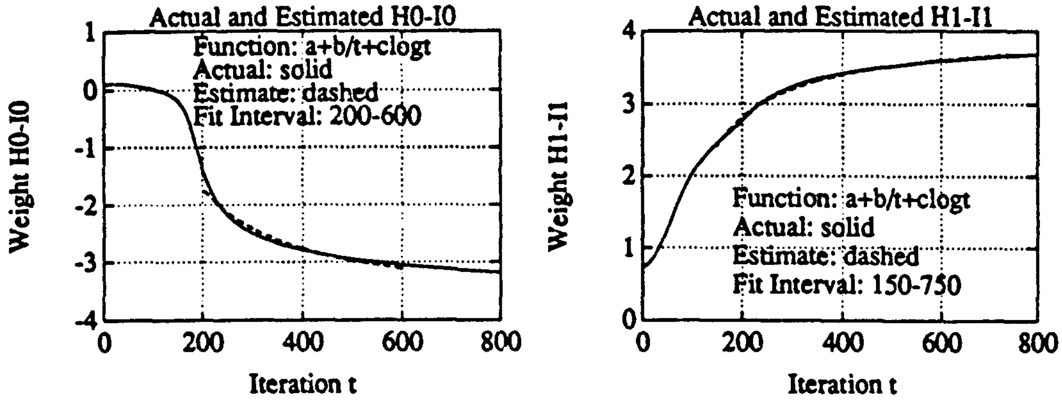


**Figure 2: Some Actual and Estimated Weights from Hidden Layer to Input Layer**

# 3 Experiments with Various Networks

## 3.1 Extrapolation Procedure

Briefly, the extrapolation procedure consists of first obtaining a value $t_c$ of $t$ for which a given network learns a certain problem. This is the 100% learning point, indicated by the fact that all output units match their desired values according to the following criterion: An output unit is considered to have classified correctly if the desired output is 1 and the activation is greater than 0.5, or if the desired output is 0 and the activation is less than 0.5. For our experiments, we obtained $t_c$ by averaging over several runs of training the network in question. However, this *guessing* operation can be somewhat automated by noting that as a rough approximation, $t_c$ can be considered to be directly proportional to the sum of number of input and output units, while it is inversely proportional to the number of hidden units, and then developing some heuristics based on these observations. It should be mentioned that such heuristics can only provide approximate values of $t_c$, and will not perform well for every problem.

After obtaining $t_c$, we set the extrapolation starting point to $t_s = 0.5t_c$. We then fit the hyperbolic function $w(t) = a + b/t$ to typically 20 iterations of actual weight data starting at $t_s$. Once the constants are determined, we use the hyperbolic function to extrapolate the weights to $t_e = 2.0t_c$. The weights thus obtained are then fed back into the BP simulator, and it is allowed to run until it maps 100% correctly. We keep track of the total number of actual simulator iterations. This is denoted by $t_a$. It is frequently the case that $t_a < t_c$, indicating computational savings in training the network. The ratio $(t_c - t_a)/t_c$ is a measure of the improvement obtained.

At this point, the network has learned the training data. However, the normalized error per output unit may still be quite high. To reduce this error, we perform the following steps repeatedly: the combined hyperbolic-logarithmic function $w(t) = a + b/t + c\log t$ is fit to approximately 20 points of weight data and the the weights are extrapolated for an aditional interval in the range $2.0t_c$ to $3.0t_c$. The weights are then fed back and the simulator restarted for $0.25t_c$ iterations, and the process is alternated until the error per output unit (a measure of convergence) reaches the desired value.

## 3.2 Test Cases

Our first test case is a 2-2-1 network, learning the Excusive OR function. The next test case is a 3-3-3 network which maps its binary inputs to their two's complement. The last case is a 3-2-8 network that learns the 3-to-8 decoding function for binary inputs.

# 4 Summary of Results

The results obtained by following the extrapolation procedure outlined above as applied to the three test cases is shown in the two tables below. The first table summarizes for three networks, the percent improvement in terms of actual iterations of BP that was obtained in mapping inputs to outputs 100% correctly by using extrapolation. For the same three networks, the second table shows the normalized error per output unit at the iteration when all outputs were correct (i.e at $t_c$), the normalized error at $4.0t_c$ obtained by extrapolating from $t_c$ for an iterval of $3.0t_c$, and a percent reduction in the normalized error per output unit.

| *Network* | $t_c$ | $t_s$ | $t_e$ | % Improvement |
|---|---|---|---|---|
| 2-2-1 XOR | 300 | 150 | 162 | 46 |
| 3-3-3 Two's Compl | 813 | 407 | 647 | 20 |
| 3-2-8 3 to 8 Decoder | 1468 | 734 | 1221 | 17 |

| *Network* | *Normalized Error per Output Unit at $t_c$* | *Normalized Error per Output Unit at $4.0t_c$* | *% Reduction in Norm. Error* |
|---|---|---|---|
| 2-2-1 XOR | 0.1499 | 0.0321 | 78 |
| 3-3-3 Two's Compl | 0.0293 | 0.0127 | 56 |
| 3-2-8 3 to 8 Decoder | 0.0319 | 0.0121 | 61 |

# 5 Conclusion

We have shown that extrapolatory techniques may substantially increase the speed of learning *and* the speed of convergence in networks using the BP algorithm. This provides motivation for constructing parametrised BP simulators with integrated ability for extrapolating weights using specified functions and heuristics. It is observed from our experiments that the particular extrapolation function can affect the acceleration of learning to a considerable degree. Discovering the extrapolation functions that work best requires further work.

# References


[1] A. J. Owens and D.L. Pilkin, *Efficient Training of the Back Propagation Network by Solving a System of Stiff Oridinary Differential Equations; in Proceedings IJCNN, 1989, pp. II-381:II-386.*

[2] Eduardo D. Sontag, *Some Remarks on the Backpropagation Algorithm for Neural Net Learning, SYCON Report 88-02, Rutgers Center for Systems and Control, Dept. of Mathematics, Rutgers University, July 1988.*

[3] W. M. Kolb, *Curve Fitting for Programmable Calculators, Syntec Inc., Bowie, MD 20716.*

[4] J. L. McClelland and D. E. Rumelhart, *Explorations in Parallel Distributed Processing: A Handbook of Models, Programs, and Exercises, MIT Press, 1988.*