Computers chem. Engng, Vol. 14, No. 3, pp. 337–341, 1990
Printed in Great Britain.

0098-1354/90 $3.00 + 0.00

IMPROVEMENT OF THE BACKPROPAGATION ALGORITHM FOR TRAINING NEURAL NETWORKS

J. LEONARD and M. A. KRAMER

Laboratory for Intelligent Systems in Process Engineering, Department of Chemical Engineering, Massachusetts Institute of Technology, Cambridge, MA 02139, U.S.A.

(*Received* 5 *May* 1989; *final revision received* 1 *November* 1989; *received for publication* 30 *November* 1989)

Abstract—The application of artificial neural networks (ANNs) to chemical engineering problems, notably malfunction diagnosis, has recently been discussed (Hoskins and Himmelblau, *Comput. chem. Engng* **12**, 881–890, 1988). ANNs "learn", from examples, a certain set of input–output mappings by optimizing weights on the branches that link the nodes of the ANN. Once the structure of the input–output space is learned, novel input patterns can be classified.

The backpropagation (BP) algorithm using the generalized delta rule (GDR) for gradient calculation (Werbos, Ph.D. Thesis, Harvard University, 1974), has been popularized as a method of training ANNs. This method has the advantage of being readily adaptable to highly parallel hardware architectures. However, most current studies of ANNs are conducted primarily on serial rather than parallel processing machines. On serial machines, backpropagation is very inefficient and converges poorly. Some simple improvements, however, can render the algorithm much more robust and efficient.

BACKPROPAGATION WITH THE GENERALIZED DELTA RULE

The BP using the GDR is explained in detail by Rumelhart *et al.* (1986), and will only be briefly summarized here. The network to be trained consists of L layers of nodes, as shown in Fig. 1. The kth layer contains N_k nodes, and for $k \neq L$, one "bias" node whose activation is always 1. Adjacent layers are exhaustively interconnected by weighted branches. The weight w_{ijk} refers to the branch from node i in layer k to node j in layer $k+1$. The first layer contains the network inputs $\mathbf{x}$ and the last layer the network outputs $\mathbf{y}$.

In the forward propagation mode, $\mathbf{x}$ is given, and each node in the second and subsequent layers calculates its activation $\mathbf{z}$ as an exponential function of the weighted sums of its inputs:

$$z_{jk} = \frac{1}{1 + e^{-u_{jk}}}, \tag{1}$$

where

$$u_{jk} = \sum_{i=1}^{N_{k-1}+1} z_{i,k-1} * w_{i,j,k-1}. \tag{2}$$

The network outputs $\mathbf{y}$ are the activations of the last column, $\mathbf{z}_L$.

In the learning mode, a set of training examples consisting of P input/output vector pairs $(\mathbf{x}_p, \mathbf{d}_p)$ is given. The objective is to select weights that minimize the sum of squared errors between the net predictions $\mathbf{y}_p$ and the desired outputs specified by the training examples $\mathbf{d}_p$ over all training examples:

$$\min_{\mathbf{w}} J = \sum_{p=1}^{P} E_p, \tag{3}$$

where E_p is the sum-of-squares error associated with a single training example:

$$E_p = ||\mathbf{y}_p - \mathbf{d}_p||^2. \tag{4}$$

To begin learning, the network is initialized with small random weights on each branch. A training example is selected randomly, and the input vector $\mathbf{x}_p$ is propagated through the network to get the predicted output $\mathbf{y}_p$. A gradient in the space of network weights is then calculated using the GDR. The GDR gives the steepest descent direction $\mathbf{m}_p$ associated with the training example p:

$$m_{ijk} = \delta_{j,k+1} * z_{ik}, \tag{5}$$

where m_{ijk} is the component of the gradient associated with w_{ijk}. For the output layer L:

$$\delta_{j,L} = (d_j - y_j) * y_j * (1 - y_j), \tag{6}$$

where $1 \leqslant j \leqslant N_L$, and for other layers,

$$\delta_{ik} = z_{ik} * (1 - z_{ik}) * \sum_{j=1}^{N_{k+1}} w_{ijk} * \delta_{j,k+1} \tag{7}$$

where $1 < k < L - 1$ and $1 \leqslant i \leqslant N_k$.

Using the gradient $\mathbf{m}_p$, the weight changes on step q, $\Delta_q \mathbf{w}$, are calculated according to the following formula:

$$\Delta_q \mathbf{w} = \eta \mathbf{m}_p + \alpha \Delta_{q-1} \mathbf{w}. \tag{8}$$

In this expression appear two constants, η called the learning rate which is equivalent to a step size, and α which acts a momentum term to keep the direction of descent from changing too rapidly from step to step. After the weights are updated, a new training example is randomly selected, and the procedure

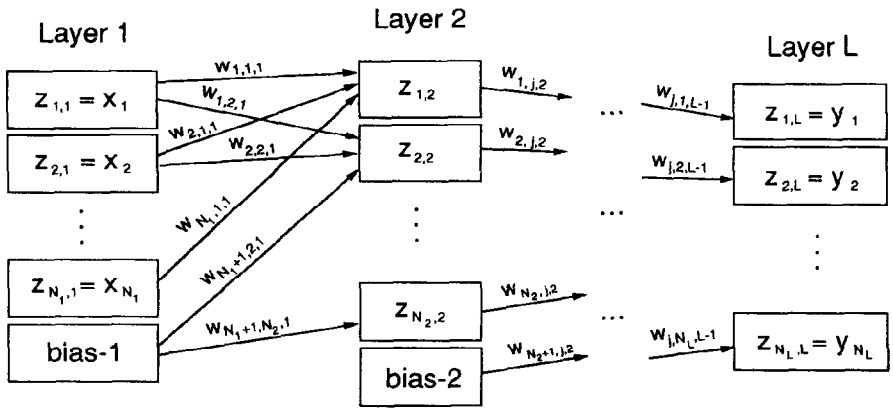

Fig. 1. Node and weight indexing (not all branches shown).

repeated until satisfactory reduction of the objective function is achieved.

PAST WORK ON IMPROVING BACKPROPAGATION

Several past authors have attempted to improve the speed of BP, based on the fact that BP is a type of gradient-descent, nonlinear optimization algorithm. A common strategy is "batching" (Rumelhart *et al.*, 1986). The gradients $\mathbf{m}_p$ represent the direction of steepest descent with respect to the partial objective E_p. It is not the direction of steepest descent with respect to the overall objective J, which is given by the sum of the gradients for the individual training examples:

$$\mathbf{M}_P = \sum_{p=1}^{P} \mathbf{m}_p. \tag{9}$$

It should be apparent that P sequential steps in the directions indicated by $\mathbf{m}_p$ is not the same as a movement in the direction $\mathbf{M}_P$, since the gradient is a local property of the solution surface. Therefore, it is most advantageous to calculate the overall direction of descent $\mathbf{M}_P$ before any change of weights is undertaken.

Watrous (1987) investigated the Davidon–Fletcher–Powell (DFP) and the Broyden–Fletcher–Goldfarb–Shanno (BFGS) methods, and compared these methods to BP and steepest descent with line search for determination of step size (SD–LS). DFP, BFGS and SD–LS all took significantly fewer iterations than BP, but the Hessian updates in DFP and BFGS required considerably more work per iteration than SD–LS, the latter having the lowest overall operations count. Due to the $O(N^2)$ scaling, Watrous concluded that the second-order methods may not be advantageous for large problems. SD–LS also performed well, relative to BP, in a study by Hush and Salas (1988).

Other investigations of second-order methods include Parker (1987), who derived an inverse Hessian update formula suitable for parallel implementation. Becker and Le Cun (1988) employed a diagonal approximation of the Hessian, trivial to invert, but less accurate and requiring more iterations, resulting in little net speed up. Kollias and Anastassiou (1989) propose the use of a "near-diagonal" form of the Hessian which also allows for weights to updated in parallel. This method converged more stably than BP, but did not provide a significant speedup on a serial machine.

Stornetta and Huberman (1987) have pointed out the effect of the choice of node transfer function on the speed of BP. Shepanski (1988) has suggested using linear least squares to train each layer of the network separately, but this method does not guarantee an overall optimum. Fahlman (1988) suggested several empirical adjustments to BP such as modifying the error function and the derivative values, combined with what amounts to a diagonal approximation of the Hessian which can be performed in parallel.

PROPOSED IMPROVEMENT TO THE GDR

Batching and optimization of step size through line search are the two strategies in the realm of first-order methods that provide most consistent improvements to BP. Although SD–LS uses a line search, it is well known that steepest descent is a relatively poor optimization method. Therefore, we propose to combine batching of examples and line search with gradient descent in conjugate directions. As we show momentarily, the conjugate gradient (CG) method can be viewed as batch BP with dynamic adjustment of η and α. In BP and batch BP, η and α are generally fixed or changed at infrequent intervals, usually by user intervention. Fixed η is undesirable since a step of arbitrary length in a local direction does not necessarily result in reduction of the objective function (for this reason, increases in the objective often occur in BP). Use of fixed momentum can cause cycling about a minimum. In SD–LS, η is adjusted on each step, but $\alpha = 0$ implies steepest descent, which can cause slow traversal of long, narrow valleys.

In place of equation (8), CG adopts the search direction (Fletcher and Reeves, 1964):

$$\mathbf{M}'_P = \mathbf{M}_P + \alpha \mathbf{M}'_{P-1}, \tag{10}$$

where the momentum is given by:

$$\alpha = ||\mathbf{M}_P||^2 / ||\mathbf{M}_{P-1}||^2. \tag{11}$$

On the first step and after every $n+1$ subsequent steps, the steepest descent direction is used ($\alpha = 0$). This is done because successive directions become nearly parallel after many iterations, but resetting the algorithm more frequently will violate the quadratic convergence (Beveridge and Schechter, 1970).

Applying this search direction, the optimization step on iteration q is given by:

$$\mathbf{w}_{q+1} = \mathbf{w}_q + \eta \mathbf{M}'_P. \tag{12}$$

The step size is determined on each iteration by performing a line search which minimizes the objective function in the direction of search. The line search must be exact, in theory, for the directions to be truly conjugate. In practice, exact minimization of the line search (to the limits of machine precision) may not result in optimal efficiency, especially for the early iterations. However, when a coarse line search is used, it is possible for the CG formulae to provide a direction that is actually ascending instead of descending. In this case the algorithm is restarted using $\alpha = 0$.

Example 1—exclusive "OR" network

XOR is a simple test case involving a network of two inputs and (as we have set it up) two outputs. The training data is given in Table 1. It is known that one hidden layer is required to fit the input/output data in this problem, so a network dimension of two nodes in each of three layers was selected. Including the bias nodes, the network has 12 weights to be determined.

The primary factor controlling the speed of convergence of BP is the size of the parameter η. The primary factor for the CG method is the tolerance used in the line search. Ten trials were done at each of several values for these parameters. In each trial, different random initial weights were used, both methods using the same initial weights. The learning was halted when the absolute error on all training examples was less than 0.1, or when the value of the objective changed by a factor less than 10^{-7} on successive iterations. The momentum factor was fixed at 0.9 for the BP method.

Since BP learns after each presentation of a training example, and batched methods only after P presentations, iterations for BP were counted in terms of the number of presentations of the entire training set ("epochs").

Table 1. Data for exclusive "OR" test problem

Case	Input 1	Input 2	Output 1	Output 2
1	0	0	0	1
2	0	1	1	0
3	1	0	1	0
4	1	1	0	1

Table 2. Result of exclusive "OR" test problem

Traditional backpropagation				
Step size, η	Iterations	Time (s)	Fraction converged	Fraction globally converged
0.75	517	5.72	0.8	0.8
0.65	281	3.16	0.7	0.7
0.5	219	2.41	0.5	0.5
0.4	252	2.78	0.6	0.6
0.25	345	3.78	0.8	0.8
0.125	569	6.45	1.0	1.0
0.0625	959	10.63	0.9	0.9
Conjugate gradients				
Line search tolerance	Iterations	Time (s)	Fraction converged	Fraction globally converged
—	24.0	1.20	1	0.6
0.5	23.2	1.20	1	0.5
0.1	26.0	1.77	1	0.6
0.01	21.2	1.74	1	0.4
0.001	20.7	2.11	1	0.5
1.0E − 4	24.6	3.15	1	0.6
1.0E − 5	21.4	3.33	1	0.5
1.0E − 6	19.4	3.54	1	0.5

— Rough bracketing of minimum without convergence test.

Table 2 gives the average number of iterations, the time to convergence, and the fraction of cases converged for BP and CG. Cases where BP failed to converge in less than 2000 presentations of the training set were *excluded from the statistics on iterations and time*. This occurred in 24% of the cases, on average. In each of the nonconvergent cases, the algorithm was cycling and no further progress was being made when the run was terminated. CG converged in all cases, attaining the global optimum in 52% of the cases. Otherwise, the algorithm converged to local optima of 5/3, 4/3 and 1.

CG is twice as fast as BP when both algorithms are set to converge as fast as possible, and at these settings, both provide comparable probabilities of finding the global minimum. It is important to note that when CG does not find the global minimum, it *still converges efficiently*, albeit to a local minimum. It seems to be preferable to make a few runs with CG using different initial conditions to find the global optimum, rather than get involved with the possibility of infinite cycling in BP. Although BP appears generally to have a greater propensity to find the global optimum *when it converges*, the tendency of the method not to converge can lead to BP requiring much more time to find the global optimum in practice.

The number of iterations for CG does not change much as the line search tolerance is changed, but the time increases logarithmically as the method does more function evaluations per iteration. Although the search directions are not truly conjugate when a coarse line search is used on this problem, the algorithm converges most quickly this way, and the convergence is still stable as long as one uses the modified restart criteria described above.

Table 3. Results for Hoskins and Himmelblau problem[a]

Hidden nodes	BP ($\eta = 0.65$)		BP ($\eta = 0.25$)		CG (tol = 0.5)	
	Iterations	Time	Iterations	Time	Iterations	Time
0	33.5 (1050)	2.79	102.8 (3200)	8.50	8.7	2.29
1	—				170.4	38.2
2	—		—		116.2	30.9
3	40.0 (57)	3.86	87.2 (111)	8.61	23.4	7.54
6	31.5 (34)	5.07	59.9 (69)	9.46	13.7	6.40
10	42.8 (38)	10.20	47.1 (58)	11.01	16.0[b]	11.45[b]
14	44.0[b] (49)	13.03[b]	41.9 (53)	13.61	19.8	17.18

[a]Values in parentheses indicates previously published data for this case.
[b]Median was used for these cases due to a single outlier biasing the mean.
— Did not converge in any run.

Example 2—example of Hoskins and Himmelblau

Hoskins and Himmelblau (1988) give an example of fault diagnosis using ANNs trained by the BP. The example represents three continuous stirred tank reactors in series with a second-order reaction. There are six inputs and six outputs. They present the number of learning iterations vs the number of hidden nodes (arranged in a single hidden layer), for learning rates between 0.25 and 0.9, and $\alpha = 0.9$. For this problem, the target values for the output nodes are set at 0.1 and 0.9, instead of the usual 0 and 1. The same termination criteria as Example 1 were used.

The average performance over 10 runs for BP and CG was determined, shown in Table 3. With no hidden layer, H–H found that BP took 500–3500 passes through the training set, whereas the improved algorithm converged in less than nine iterations, on average.† BP failed repeatedly to converge with one or two nodes present in the hidden layer, whereas CG converged in every case, locating the global optimum in the latter case in nine out of 10 trials. With one hidden node, even though there was not sufficient structure in the network to fit all the training examples, CG located local optima where the resulting network correctly classified six out of 12 examples, on average. For three to 14 hidden nodes, there was little to differentiate the two methods. However, we observed that when the targets were reset to 0 and 1, BP had much more difficulty converging.

CONCLUSION

The proposed algorithm for training neural networks differs from the conventional BP algorithm in the following respects:

1. Rather than taking steps based on directions indicated by single training examples, a net gradient representing the steepest descent direction over the complete set of training cases is utilized. The required gradient is simply the sum of the gradients generated by the GDR for the individual training examples.
2. The momentum factor α is determined on a step-to-step basis rather than being held constant. Using α as given by equation (11) results in search along conjugate directions. Methods based on conjugate gradients are generally more effective than methods based on steepest descent when solution surfaces are poorly scaled.
3. The step size; rather than be held fixed at an arbitrary value, is adjusted on each step using a line search. This procedure assures a monotonic reduction in the objective function, and prevents cycling and overshoot which are frequently observed using BP.

Experiments with the improved algorithm show much better convergence stability and speed than BP. It is recommended that the current algorithm be adopted when neural networks are trained using conventional serial computing hardware.

†Although most of the original results were reproduced well, with no hidden layer our BP algorithm converged in considerably fewer iterations than required by H–H.

REFERENCES

Becker S. and Y. Le Cun, Improving the convergence of back-propagation learning with second order methods. *Proc. Connectionist Models Summer School* (D. Touretzky, G. Hinton and T. Sejnowski, Eds), pp. 29–37. Morgan-Kaufman, San Mateo (1988).

Beveridge G. S. G. and R. S. Schechter, *Optimization: Theory and Practice*. McGraw-Hill, New York (1970).

Fahlman S. E., Faster-learning variations on back-propagation: an empirical study. *Proc. Connectionist Models Summer School* (D. Touretzky, G. Hinton and T. Sejnowski, Eds), pp. 38–51. Morgan-Kaufman, San Mateo (1988).

Fletcher R. and C. M. Reeves, Function minimization by conjugate gradients. *Comput. J.* **7**, 149–154 (1964).

Hoskins J. C. and D. M. Himmelblau, Artificial neural network models of knowledge representation in chemical engineering. *Comput. chem. Engng* **12**, 881–890 (1988).

Hush D. R. and J. M. Salas, Improving the learning rate of back-propagation with the gradient reuse algorithm. *IEEE Second Int. Conf. Neural Networks*, San Diego, pp. I-444–447 (1988).

Kollias S. and D. Anastassiou, An adaptive least squares algorithm for the efficient training of artificial neural networks. *IEEE Trans. Circuits Systems* **36**, 1092–1101 (1989).

Parker D. B., Optimal algorithms for adaptive networks: second order back propagation, second order direct propagation, and second order Hebbian learning. *IEEE First Int. Conf. Neural Networks*, San Diego, pp. II-593–600 (1987).

Rumelhart D. E., G. E. Hinton and R. J. Williams, Learning internal representations by error propagation. *Parallel Distributed Processing*, Chap. 8. (D. E. Rumelhart and J. L. McClelland, Eds). MIT, Cambridge, Mass. (1986).

Shepanski J. F., Fast learning in artificial neural systems: multilayer perceptron training using optimal estimation. *IEEE Second Int. Conf. Neural Networks*, San Diego, pp. I-465–472 (1988).

Stornetta W. S. and B. A. Huberman, An improved three-layer backpropagation algorithm. *IEEE First Int. Conf. Neural Networks*, San Diego, pp. II-637–643 (1987).

Watrous R. L., Learning algorithms for connectionist networks: applied gradient methods of nonlinear optimization. *IEEE First Int. Conf. Neural Networks*, San Diego, pp. II-619–627 (1987).

Werbos P. J., Beyond regression: new tools for prediction and analysis in the behavioral sciences. Ph.D. Thesis, Harvard University Committee in Applied Mathematics (1974).