Simulated Annealing and Weight Decay in Adaptive Learning: The SARPROP Algorithm

Nicholas K. Treadgold and Tamas D. Gedeon

***Abstract*—A problem with gradient descent algorithms is that they can converge to poorly performing local minima. Global optimization algorithms address this problem, but at the cost of greatly increased training times. This work examines combining gradient descent with the global optimization technique of simulated annealing (SA). Simulated annealing in the form of noise and weight decay is added to resiliant backpropagation (RPROP), a powerful gradient descent algorithm for training feedforward neural networks. The resulting algorithm, SARPROP, is shown through various simulations not only to be able to escape local minima, but is also able to maintain, and often improve the training times of the RPROP algorithm. In addition, SARPROP may be used with a restart training phase which allows a more thorough search of the error surface and provides an automatic annealing schedule.**

***Index Terms*—Adaptive, backpropagation algorithm, gradient descent, neural network, RPROP, simulated annealing, weight decay.**

I. Introduction

THERE ARE two traditional methods for training feedforward neural networks: gradient descent and global optimization. The most commonly used method is gradient descent, which includes algorithms such as backpropagation [1], conjugate gradient methods [2], and the Levenberg–Marquardt algorithm [3]. Conjugate gradient methods and algorithms using the Hessian, such as the Levenberg–Marquardt algorithm, generally converge to minima more rapidly than methods based only on steepest descent, such as backpropagation [2], [3]. Backpropagation, however, has the advantage of being less computationally expensive for a given size network; a factor which becomes much more important for larger networks. There are a number of algorithms which improve on backpropagation's convergence properties while maintaining its computational simplicity [4]–[8].

One problem inherent with gradient descent methods is their convergence to local minima. While some local minima can provide solutions which are acceptable, they often result in poor performance. This problem can be overcome through the use of global optimization. In the field of neural networks some global optimization algorithms which have been employed include simulated annealing (SA) [9]–[11], evolutionary methods [12], [13], random methods [14], [15], and deterministic searches [16]. Global optimization, however, has the problem of being computationally expensive, particularly for large networks.

Both gradient descent and global optimization methods have inherent problems: gradient descent can converge to poorly performing local minima, and global optimization is computationally expensive. In order to overcome these problems, hybrid methods employing gradient descent and some form of global optimization have been examined [17]–[19]. This combination of techniques often results in improved convergence times compared to the standard global optimization, and in some cases even maintains the guarantee of convergence to a global minimum [17], [19].

In this paper, we look at combining a quick and computationally cheap gradient descent algorithm, resilient backpropagation (RPROP) [8], [20], with the global search technique of SA. The aim of this combination of techniques is to maintain quick convergence using a computationally cheap algorithm, while reducing the likelihood of convergence to poor local minima. This paper is organized as follows. First, the RPROP algorithm is described and the reasons for its choice as the gradient descent method are discussed. Next, the SA enhancements made to RPROP to obtain the SARPROP algorithm are given. The benefits of using SARPROP with a restart training phase are also discussed through the introduction of the ReSARPROP algorithm. The results of comparative simulations between RPROP, SARPROP, and ReSARPROP are then presented and discussed, and conclusions drawn.

II. Resilient Backpropagation

There have been a number of refinements made to the backpropagation (BP) algorithm. One the most successful in terms of convergence is RPROP [8], [20]. Not only is RPROP one of the faster converging BP variants, its also has the important advantage of having only a single user set parameter. In addition, this single parameter is relatively invariant to its initial value since it is adapted quickly by RPROP to suit the problem [8]. This is a great advantage over many other BP variants which have a number of parameters whose setting often greatly influences the performance of the algorithm on a given problem [8]. A further advantage of RPROP is that it maintains the computational simplicity of BP.

There are two major differences between BP and RPROP. First, RPROP modifies the size of the weight step taken adaptively, and second, the mechanism for adaptation in RPROP does not take into account the magnitude of the gradient $(\delta E/\delta w_{ij})$ as seen by a particular weight, but only

Manuscript received December 17, 1996; revised March 15, 1998.

The authors are with the Department of Information Engineering, School of Computer Science and Engineering, The University of New South Wales, Sydney N.S.W. 2052, Australia.

Publisher Item Identifier S 1045-9227(98)04454-3.

$\forall i,j$: $\Delta_{ij}(t) = \Delta_0$
$\forall i,j$: $\delta E/\delta w_{ij}(t-1) = 0$
Repeat
  Compute Gradient $\delta E/\delta w(t)$
  For all weights and biases
  if $(\delta E/\delta w_{ij}(t-1) * \delta E/\delta w_{ij}(t) > 0)$ **then**
    $\Delta_{ij}(t) =$ **minimum** $(\Delta_{ij}(t-1) * \eta^+, \Delta_{max})$
    $\Delta w_{ij}(t) =$ - **sign** $(\delta E/\delta w_{ij}(t)) * \Delta_{ij}(t)$
    $w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t)$
    $\delta E/\delta w_{ij}(t-1) = \delta E/\delta w_{ij}(t)$
  else if $(\delta E/\delta w_{ij}(t-1) * \delta E/\delta w_{ij}(t) < 0)$ **then**
    $\Delta_{ij}(t) = \Delta_{ij}(t-1) * \eta^-$
    $\Delta_{ij}(t) =$ **maximum** $(\Delta_{ij}(t), \Delta_{min})$
    $\delta E/\delta w_{ij}(t-1) = 0$
  else if $(\delta E/\delta w_{ij}(t-1) * \delta E/\delta w_{ij}(t) = 0)$ **then**
    $\Delta w_{ij}(t) =$ - **sign** $(\delta E/\delta w_{ij}(t)) * \Delta_{ij}(t)$
    $w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t)$
    $\delta E/\delta w_{ij}(t-1) = \delta E/\delta w_{ij}(t)$
Until (converged)

Fig. 1. The RPROP algorithm.

the sign of the gradient. This allows the step size to be adapted without having the size of the gradient interfere with the adaptation process [8]. In a number of previous BP variants, the learning parameter, η, has been varied adaptively [4], [5]. Both the learning parameter and the magnitude of the gradient, however, effect the actual step size taken in these algorithms. The size of the gradient is unforeseeable, and hence it has the potential to disrupt the adaptation of the learning parameter. RPROP overcomes this disruption, and the effect of this improvement has been demonstrated empirically in comparisons with a number of BP variants [8].

The RPROP algorithm works by modifying each weight by an amount $\Delta_{ij}(t)$, termed the update value, in such a way as to decease the overall error. All update values are initialized to the value Δ_0. The update value for a weight is modified in the following manner: if the gradient $(\delta E/\delta w_{ij}(t))$ direction has remained the same in successive epochs, then the update value is multiplied by a value η^+ (which is greater than one). Similarly, if the gradient direction has changed, the update value is multiplied by the value η^- (which is less than one). This results in the update value for each weight adaptively growing or shrinking as a result of the sign of the gradient seen by that weight. There are two limits placed on the update value: a maximum $\Delta_{\max}$, and a minimum $\Delta_{\min}$. The initial update value Δ_0 is the single parameter which requires setting by the user, with all other algorithm parameters treated as constants. The RPROP algorithm is shown in Fig. 1.

An algorithm which is similar in nature to RPROP is Salomon and Leo van Hemmen's modification of BP through the use of dynamic self adaptation [7]. In this algorithm the step size is adapted in a similar manner to RPROP, except that at each stage both an increase and a decrease to the step size is examined for each weight, with the best performing modification selected. The gradient is also normalized in this algorithm to stop the interference of varying gradient magnitudes on the step size taken.

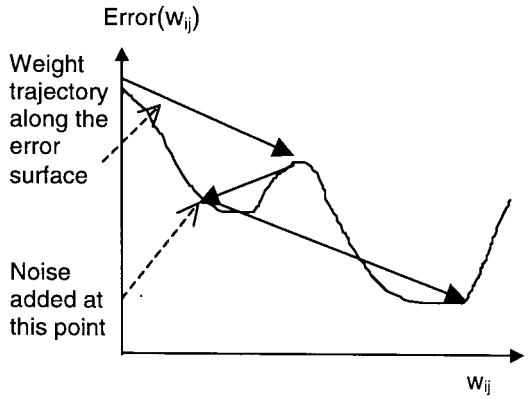

Fig. 2. An example of a weight trajectory along an error surface, and the result of SARPROP's noise addition.

III. SARPROP

While RPROP can be extremely fast in converging to a solution, it suffers from the same problem faced by all gradient-descent-based methods: it can often converge to poor local minima. SARPROP attempts to address this problem by using the method of SA [21], [22]. SA, in general, involves the addition of noise to the parameters undergoing optimization. The amount of noise added is associated with a SA term, which decreases the effect of the noise as training progresses. The addition of noise allows the network to move in a direction which is not necessarily the direction of steepest descent. The benefit this provides is that it can help the network escape from local minima. Burton and Mpitsos [18] have shown that SA can help increase the speed of convergence of the BP algorithm, and conclude that noise "simply permits or facilitates greater access to such pathways that are not easily reached in the networks not containing noise."

In SARPROP, noise is added to the standard RPROP weight update value when both the error gradient changes sign in successive epochs, and the magnitude of the update value is less than a value proportional to the SA term. The amount of noise added is proportional to the SA term. The reason for adding noise to the update value only when both the error gradient changes sign, and the update value is below a given setting, is to minimize the disturbance to the normal adaptation of the update value. Following this scheme means that the update value is only modified by noise when it has a relatively small value (indicating a number of previous gradient crossings). This can allow the weight to jump out of local minima (Fig. 2), while minimizing the disturbance to the adaptation process.

SARPROP uses SA not only on the noise added to the weight updates, but also on the amount of weight decay the network uses. Weight decay is implemented in SARPROP by adding a penalty term to the error function, which results in a modification of the error gradient. The weight decay term in SARPROP is designed such that small valued weights decay more rapidly than larger weights, and is similar to the penalty term in [23]. This decay term allows important functionality already learned by the network (and represented by the large weights) to be retained more easily. This form of weight decay

```
∀i,j: Δij(t) = Δ0
∀i,j: δE/δwij(t-1) = 0
Repeat
    Compute SARPROP Gradient δE/δw(t)
    For all weights and biases
    if (δE/δwij(t-1) * δE/δwij(t) > 0) then
        Δij(t) = minimum (Δij(t-1) * η+, Δmax)
        Δwij(t) = - sign (δE/δwij(t)) * Δij(t)
        wij(t+1) = wij(t) + Δwij(t)
        δE/δwij(t-1) = δE/δwij(t)
    else if (δE/δwij(t-1) * δE/δwij(t) < 0) then
        if (Δij(t-1) < 0.4* SA2) then
            Δij(t) = Δij(t-1) * η- + 0.8*r*SA2
        else
            Δij(t) = Δij(t-1) * η-
        Δij(t) = maximum (Δij(t), Δmin)
        δE/δwij(t-1) = 0
    else if (δE/δwij(t-1) * δE/δwij(t) =0) then
        Δwij(t) = - sign (δE/δwij(t)) * Δij(t)
        wij(t+1) = wij(t) + Δwij(t)
        δE/δwij(t-1) = δE/δwij(t)
Until (converged)
```

Fig. 3. The SARPROP algorithm.

was found to produce better convergence properties than the one used in earlier version of SARPOP [24] which employed a weight decay rate proportional to the weight size. The SA term applied to the weight decay in SARPROP results in the influence of the weight decay decreasing as training proceeds. The SARPROP error gradient is shown below

$$\delta E/\delta w_{ij}^{\text{SARPROP}} = \delta E/\delta w_{ij} - 0.01 * w_{ij}/\left(1 + w_{ij}^2\right) * \text{SA}$$

$$\text{where SA} = 2^{-T*\text{epoch}} \quad \text{and } T = \text{temperature}.$$

The effect of this form of weight decay is to modify the error surface so that initially smaller weights are favored. As training progresses the weight decay magnitude is reduced, which modifies the error surface to allow for the easier growth of large weights. This technique has two purposes. First, the modification of the error surface performs a task similar to that of the noise in the weight updates: it may allow the network to explore regions of the error surface which were previously unavailable. Second, the use of weight decay is a form of regularization which has been found to improve generalization [25]. Weight decay restricts the type of functionality which the network can produce by favoring networks producing smoother functions. Smooth output functions are more likely in general to represent the underlying functions of real-world data.

SARPROP's use of noise and weight decay may allow it to compensate for a problem which can occur in RPROP. A disadvantage of the exponential growth of the update value in RPROP is that it can overshoot a good local minimum and perhaps converge to a poor one. In such cases SARPROP may be able to escape this poor solution through its use of noise, and thus arrive at the better performing local minima. In addition, the use of weight decay will modify the error surface of a given problem in such a way as to reduce the growth of large update values.

After these enhancements have been incorporated, the resulting SARPROP algorithm is shown in Fig. 3, in which r is a random number between zero and one. The only parameter requiring setting prior to training is the temperature T, a part of the SA term which effects the speed by which the noise and weight decay are reduced. The optimal setting of this parameter is problem dependent, although good values were found in the range 0.01 to 0.05. In general it was found the more complex the problem, the lower the temperature value required (and hence the slower the annealing process).

It should be noted that there is still no guarantee that SARPROP will converge to a good local minimum, only that the likelihood of such is increased. It may well be the case that the effect of noise and weight decay will push SARPROP away from a good solution. As in pure SA, however, the chance of converging to a good local minimum is increased via these additions to SARPROP. In order to increase this chance further it is possible to use SARPROP in a restart mode. This is done by restarting training whenever SARPROP converges. The current weight values are used as the new initial weights, so that any prior training knowledge is maintained. Since training is restarted, the SA terms are reset, which may allow the network to jump out from its current local minimum and perhaps converge to a better solution. This algorithm will be termed ReSARPROP.

A further advantage of ReSARPROP is that it solves the problem of selecting a good value for the temperature parameter. SARPROP requires this parameter to be set prior to training, and the optimal value depends on the problem. Using ReSARPROP, the temperature can be initially set to give fast annealing. If a good solution has not been reached this temperature can be reset to allow for slower annealing when the network is restarted. The removal of the temperature parameter in ReSARPROP results in ReSARPROP being (user) parameter free.

The annealing schedule used for ReSARPROP was based on the following temperature values: 0.050, 0.048, 0.044, 0.036, 0.020, 0.010, 0.010, ... (based on an exponential sequence with a minimum of 0.010). A given training run was considered to have reached a local minimum after both a minimum training period, and the rms error improves by less than 0.001 over a 50-epoch period. A minimum training time is specified to allow the effect of the noise and weight decay to diminish sufficiently for convergence to occur. The minimum training time in epochs is given by the formula: 6/Temperature. This results in the noise and weight decay magnitudes being reduced by a factor of 2^{-6}, a value which is small enough to allow convergence to occur.

IV. Comparative Simulations

To test the effectiveness of SARPROP and ReSARPROP, their performance was compared against that of RPROP on a number of standard benchmark problems. SARPROP uses the standard RPROP constant settings [8] $\eta^+ = 1.2, \eta^- =$

TABLE I
ENCODER RESULTS

	TRAINING (EPOCHS)		
	Average	Median	Non-Cvg.
ENC10:			
RPROP	28.44	27	0
SARPROP	29.78	30	0
ReSARPROP	29.78	30	0
ENC12:			
RPROP	197.3	185.5	0
SARPROP	158.52	154.5	0
ReSARPROP	160.2	154.5	0
ENC16:			
RPROP	317.26	303.5	0
SARPROP	273.48	258	0
ReSARPROP	279.1	258	0
ENC32:			
RPROP	184.48	184	0
SARPROP	192.88	191	0
ReSARPROP	194.92	191	0

$0.5, \Delta_{\max} = 50, \Delta_{\min} = 1 \times 10^{-6}$. In addition, a constant value of 0.0001 was added to the derivative of the sigmoid in order to overcome the "flat spot" problem [6] for all algorithms. All weights were initialized to random values in the range -0.7 to 0.7.

The single user set parameter in SARPROP, the temperature, was chosen to maximize its performance. RPROP also uses a single parameter, the initial update value Δ_0. As noted in [8], this parameter was found to be generally problem independent, and its optimization resulted in little improvement in RPROP. A value which performed well and was selected for RPROP was 0.1. SARPROP also uses this setting for Δ_0. This value is treated as a constant in SARPROP.

To compare RPROP, SARPROP, and ReSARPROP, each algorithm was used to train a network on a given problem. For each problem, all training attempts were repeated 50 times using different initial weights. A maximum number of training epochs was selected, and training was halted if this number was reached. In this case the training was defined to be nonconverging. In order to compare the algorithms it was decided to take three measurements: the average and median of the number of epochs required for training, and the number of nonconverged training runs. For classification problems the 40-20-40 threshold and margin criterion was used [6]. In using the 40-20-40 criterion, a class has been learned correctly if the neuron's output is in the correct upper or lower 40% of its output range. For regression problems, a RMS error value was specified to indicate convergence. The performance on the test sets was also measured where applicable, and the median value reported.

The first series of comparisons were performed using the encoder data sets. These consist of identical input output binary training patterns. Each training pattern has only one bit set to one, with the rest set to zero. The number of hidden units chosen is less than the input–output dimension, and thus the hidden units perform compression. Four encoder problems of differing network structure were chosen: the 10-5-10 Encoder (with ten training patterns); the 12-2-12 Encoder (with 12 training patterns); the 16-2-16 Encoder (with 16 training patterns); and the 32-3-32 Encoder (with 32 training patterns). The 40-20-40 criterion was used to halt training. An asymmetric sigmoid function (zero to one) was used as the activation function. The maximum training time was set at 2000 epochs. The temperature parameter for SARPROP was set to 0.050. The Encoder results are displayed in Table I.

The next series of comparisons were performed using the odd parity data sets: Parity 3 to 8. The network structures chosen for each problem were: 3-3-1 (Parity 3); 4-6-1 (Parity 4); 5-7-1 (Parity 5); 6-9-1 (Parity 6); 7-11-1 (Parity 7); and 8-16-1 (Parity 8). Training was halted when the 40-20-40 classification criterion was satisfied on the training set. A symmetric sigmoid function (-0.5 to 0.5) was used as the activation function. The maximum training time was set at 2000 epochs. The temperature parameter for SARPROP was set to 0.010. The Parity results are displayed in Table II.

Comparisons were also performed on two "real-world" data sets. The first was Fisher's classic Iris data set that classifies irises into three classes. The Iris data set was obtained from the UCI machine learning database [26]. The second was the Thyroid data set that classifies a patient into three classes: normal (not hypothyroid), hyperfunction, and subnormal functioning. The Thyroid data set was obtained from the Proben1 [27] database (also available in original form from the UCI database). An asymmetric sigmoid function (zero to one) was used as the activation function for both these data sets.

The Iris data set consists of 120 training patterns and 30 test patterns. The maximum training time was set at 2000 epochs. Training was halted when the 40-20-40 classification criterion was met, at which point the percentage correctly classified on the test set was measured. The network structure used was 4-2-3. The temperature parameter for SARPROP was set to 0.050. The Iris results are displayed in Table III.

The Thyroid data set consists of 3600 training patterns and 1800 test patterns (which came from the thyroid1.td file in Proben1). Training was halted at an rms of 0.06, at which point the test set was applied. The maximum training time

TABLE II
PARITY RESULTS

	TRAINING (EPOCHS)		
	Avg.	Median	Non-Cvg.
Par3:			
RPROP	60.12	18.5	1
SARPROP	24.22	24	0
ReSARPROP	46.88	21	0
Par4:			
RPROP	734.34	172	16
SARPROP	60.32	44.5	0
ReSARPROP	136.24	66	0
Par5:			
RPROP	448.84	48	8
SARPROP	58.88	42	0
ReSARPROP	155.3	56.5	0
Par6:			
RPROP	1313.84	2000	31
SARPROP	127.96	87	0
ReSARPROP	232.02	216	0
Par7:			
RPROP	912.74	485	18
SARPROP	180.82	102	1
ReSARPROP	200.06	99.5	0
Par8:			
RPROP	1509.6	2000	34
SARPROP	471.8	241.5	6
ReSARPROP	415.7	332.5	0

TABLE III
IRIS RESULTS

	TRAINING (EPOCHS)			
	Avg.	Median	Non-Cvg.	Test%
RPROP	569.12	396.5	5	83.33
SARPROP	352.66	283.5	0	86.67
ReSARPROP	352.66	283.5	0	86.67

TABLE IV
THYROID RESULTS

	TRAINING (EPOCHS)			
	Average	Median	Non-Cvg.	Test%
RPROP	693.06	353	9	97.78
SARPROP	434.96	312.5	2	97.66
ReSARPROP	439.44	312.5	2	97.75

was set at 2000 epochs. The network structure used was 21-4-3. The temperature parameter for SARPROP was set to 0.010. The Thyroid results are displayed in Table IV.

Regression simulations were next performed on the complex interaction function, described in detail in [28], and shown below

$$f(x_1, x_2) = 1.9(1.35 + e^{x_1}\sin(13(x_1 - 0.6)^2)e^{-x_2}\sin(7x_2)).$$

The set up of training and test data follows the method of [28]. For each function two sets of training data were created, one noise-free and one noisy, using 225 random values. The noisy data was created by adding independent and identically distributed Gaussian noise, with zero mean and 0.065 variance. A test set of size 2500 was generated on a regularly spaced grid $[0,1]^2$. The network structure used was 2-14-1. The maximum training time was set at 10 000 epochs. Training was continued until RMS errors of 0.15 and 0.28 were reached for the noise-free and noisy data sets, respectively, at which point the performance on the test set was measured. The measure used was the fraction of variance unexplained (FVU) [28], which is proportional to the total sum of squares error. A symmetric sigmoid function (-0.5 to 0.5) was used as the activation function for all neurons except the output neuron, which used a linear activation function. The temperature parameter for SARPROP was set to 0.010. The results are shown in Table V.

V. DISCUSSION

The results on all benchmarks demonstrate the effectiveness of the combination of noise and weight decay in allowing SARPROP to escape from local minima. This is especially

TABLE V
COMPLEX INTERACTION FUNCTION RESULTS

	TRAINING (EPOCHS)			
	Average	Median	Non-Cvg.	FVU
CIF:				
RPROP	9793.96	10000	48	0.052
SARPROP	8530.22	10000	33	0.040
ReSARPROP	7193.12	6934.5	8	0.038
CIF – noise:				
RPROP	7230.18	9800.5	24	0.070
SARPROP	5708.28	5086.5	12	0.058
ReSARPROP	5413.32	5508.5	3	0.066

apparent in the Parity results, where SARPROP is very successful at converging to a solution, while RPROP often fails to do so. For example, on the Parity 6 data set RPROP failed to converge 62% of the time, compared with SARPROP which converged in every run. In addition, for the data sets which have a test set, SARPROP can be seen to often produce better generalization results than RPROP.

ReSARPROP produces even better convergence results than SARPROP, especially on the more difficult problems such as Parity 8 and the regression data sets. From the total of 700 simulations performed for each algorithm, ReSARPROP only fails to reach the convergence criteria 13 times compared to SARPROP's 54 and RPROP's 194. While ReSARPROP is able to converge more frequently than SARPROP, in general it is more expensive computationally due to the fact that it may perform a number of training restarts. Again, generalization improvements are seen for ReSARPROP over RPROP. The reasons for SARPROP and ReSARPROP's improved generalization results can be attributed to their use of weight decay.

The combination of noise and weight decay can be seen to be successful in allowing the SARPROP trained networks to converge to good local minima. Importantly this combination does not increase training times, as is often the case with SA methods. In fact, decreases in training times compared to RPROP were generally observed. These results support Burton and Mpitsos' [18] hypothesis that the addition of noise can allow access to paths along the error surface which allow more rapid convergence. A good example of SARPROP and ReSARPROP's convergence speed is given with the Encoder results. RPROP is easily able to solve these data sets, with it always converging to a solution. Even on these data sets, however, SARRPOP and ReSARPROP are still able to maintain and sometimes even improve on the convergence results of RPROP.

One disadvantage of the SARPROP algorithm is its reliance on the temperature parameter. This parameter must be set prior to training, and its optimal value is dependent on the data set. ReSARPROP overcomes this problem by automatically selecting a schedule of temperature values. This not only removes the need to set any parameters for ReSARPROP, but can also improve the convergence performance of the algorithm. In addition, ReSARPROP can easily store the weight values each time it converges to a solution before training is restarted. This ensures that the best performing solutions are not lost, but can be reinstalled at any point.

Previous work with SA in neural networks allows a limited comparison between RPROP, some SA techniques and a random search technique. Fang and Li [9] compared BP against SA using Uniform, Gaussian, and Cauchy distributions, and also against Baba's random search algorithm [17]. In general they found the SA techniques and Baba's algorithm to be faster than the basic BP algorithm. The speed increases were found to be of the order of two to seven depending on the problem examined. RPROP is shown to result in speedups over an optimized BP with momentum of at least this [8] using similar benchmarks, with RPROP often converging to solutions which BP never achieves. SARPROP and ReSARPROP perform at least as well as RPROP, and thus compare favorably against the SA and random search techniques on these benchmarks. It is expected that as the size of the networks increase, the convergence speed of these global optimization algorithms would degrade significantly in comparison with the performance of SARPROP and ReSARPROP, which are largely gradient based.

Unlike a number of algorithms employing a global search, SARPROP and ReSARPROP do not guarantee convergence to the global minimum. Many global search algorithms employing stochastic methods such as [9]–[15] theoretically converge to a global minimum. As noted in [16], however, this convergence is not guaranteed in finite time, but only in the limit. SARPROP and ReSARPROP sacrifice the global convergence property for speed of convergence. RPROP is one of the fastest BP variants and even in data sets where convergence to poor local minima does not occur, SARPROP is able to maintain and even improve the performance of RPROP. Fast convergence combined with the computational simplicity of the algorithm makes SARPROP and ReSARPROP applicable for training larger networks where global optimization, and even computationally expensive gradient descent methods, are unsuitable.

VI. CONCLUSION

A SA addition to the RPROP algorithm, SARPROP, has been proposed. The success of the combination of noise and weight decay in increasing both the convergence speed, and the chance of convergence has been demonstrated on a number of benchmark problems. Combining SARPROP with a restart training phase improves convergence and provides the additional benefit of having no parameters required to be set by the user prior to training.

ACKNOWLEDGMENT

The authors wish to thank the anonymous reviewers for their helpful comments.

REFERENCES

[1] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, vol. 1: Foundations*, D. E. Rumelhart and J. L. McClelland, Eds. Cambridge, MA: MIT Press, 1986, pp. 318–362.

[2] M. F. Moller, "A scaled conjugate gradient algorithm for fast supervised learning," *Neural Networks*, vol. 6, pp. 525–533, 1993.

[3] M. T. Hagan and M. B. Menhaj, "Training feedforward networks with the Marquardt algorithm," *IEEE Trans. Neural Networks*, vol. 5, pp. 989–993, 1994.

[4] R. A. Jacobs, "Increased rates of convergence through learning rate adaption," *Neural Networks*, vol. 1, pp. 295–307, 1988.

[5] T. Tollenaere, "SuperSAB: Fast adaptive backpropagation with good scaling properties," *Neural Networks*, vol. 3, pp. 561–573, 1990.

[6] S. E. Fahlman, "An empirical study of learning speed in backpropagation networks," Dept. Comput. Sci., Carnegie Mellon Univ., Pittsburgh, PA, Tech. Rep., CMU-CS-88-162, 1988.

[7] R. Salomon and J. L. van Hemmen, "Accelerating backpropagation through dynamic self-adaption," *Neural Networks*, vol. 9, pp. 589–601, 1996.

[8] M. Riedmiller and H. Braun, "A direct adaptive method for faster backpropagation learning: The RPROP algorithm," in *Proc. ICNN 93*, San Francisco, CA, 1993, pp. 586–591.

[9] L. Fang and T. Li, "A globally optimal annealing learning algorithm for multilayer perceptrons with applications," in *Proc. AI'90: Australian Joint Conf. Artificial Intell.* Perth, Australia: World Scientific, 1990, pp. 201–206.

[10] D. H. Ackley, G. E. Hinton, and T. J. Sejowski, "A learning algorithm for Boltzman machines," *Cognitive Sci.*, vol. 9, pp. 147–169, 1985.

[11] G. E. Hinton and T. J. Sejowski, "Learning and relearning in Boltzman machines," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, vol. 1: Foundations*, D. E. Rumelhart and J. L. McClelland, Eds. Cambridge, MA: MIT Press, 1986, pp. 282–317.

[12] D. Whitley, T. Starkweather, and C. Bogart, "Genetic algorithms and neural networks: Optimizing connections and connectivity," *Parallel Comput.*, vol. 14, pp. 347–361, 1990.

[13] D. B. Fogel, L. J. Fogel, and V. W. Porto, "Evolving neural networks," *Biol. Cybern.*, vol. 63, pp. 487–493, 1990.

[14] N. Baba, "A new approach for finding the global minimum of error function of neural networks," *Neural Networks*, vol. 2, pp. 367–373, 1989.

[15] R. Brunelli, "Training neural nets through stochastic minimization," *Neural Networks*, vol. 7, pp. 1405–1412, 1994.

[16] Z. Tang and G. J. Koehler, "Deterministic global optimal FNN training algorithms," *Neural Networks*, vol. 7, pp. 301–311, 1994.

[17] N. Baba, Y. Mogami, M. Kohzaki, Y. Shiraishi, and Y. Yoshida, "A hybrid algorithm for finding the global minimum of error function of neural networks and its applications," *Neural Networks*, vol. 7, pp. 1253–1265, 1994.

[18] R. M. Burton and G. J. Mpitsos, "Event dependent control of noise enhances learning in neural networks," *Neural Networks*, vol. 5, pp. 627–637, 1992.

[19] M. A. Styblinski and T. S. Tang, "Experiments in nonconvex optimization: Stochastic approximation with function smoothing and simulated annealing," *Neural Networks*, vol. 3, pp. 467–483, 1990.

[20] M. Riedmiller, "RPROP—Description and implementation details," Univ. Karlsruhe, Germany, Tech. Rep., 1994.

[21] P. J. M. van Laarhoven and E. H. L. Arts, *Simulated Annealing: Theory and Applications*. Dordrecht, The Netherlands: D. Reidel, 1988.

[22] H. Szu, "Nonconvex optimization by fast simulated annealing," in *Proc. IEEE*, 1987, vol. 75, pp. 1538–1540.

[23] A. Weigend, D. Rumelhart, and B. Huberman, "Generalization by weight elimination with application to forecasting," *Advances in Neural Information Processing III*, R. Lippman, J. Moody, and D. Touretzy, Eds. San Mateo, CA: Morgan Kaufman, 1991, pp. 857–882.

[24] N. K. Treadgold and T. D. Gedeon, "A simulated annealing enhancement to resilient backpropagation," in *Proc. Int. Panel Conf. Soft and Intell. Comput.*, Budapest, Hungary, 1996, pp. 289–293.

[25] W. Finnoff, F. Hergery, and H. G. Zimmerman, "Improving model selection by nonconvergent methods," *Neural Networks*, vol. 6, pp. 771–783, 1993.

[26] P. M. Murphy and D. W. Aha, UCI Repository of Machine Learning Databases, Univ. California, Irvine, Dept. Inform. Comput. Sci., 1994. Available FTP: http://www.ics.uci.edu/∼mlearn/MLRepository.html

[27] L. Prechelt, Proben1—A Set of Neural-Network Benchmark Problems, Univ. Karlsruhe, Germany, 1994. Available FTP: ira.uka.de/pub/neuron/proben1.tar.gz

[28] J. Hwang, S. Lay, R. Maechler, and D. Martin, "Regression modeling in backpropagation and projection pursuit learning," *IEEE Trans. Neural Networks*, vol. 5, pp. 342–353, 1994.

Nicholas K. Treadgold received the B.E. (Hons.) degree in electrical engineering and M.Sc. in computer science in 1993 and 1995, respectively, from the University of New South Wales, Sydney. Since 1996 he has been working toward the Ph.D. degree in artificial neural-network training algorithms and architectures at the same university.

His research interests include feedforward and recurrent neural networks, reinforcement learning, and evolutionary algorithms.

Tamas D. Gedeon received the B.Sc. (Hons.) and Ph.D. degrees in computer science from The University of Western Australia, Perth, in 1981 and 1989, respectively, and a Graduate Diploma in Management from the Australian Graduate School of Management, the University of New South Wales, Sydney, in 1994.

He has held academic positions at the University of Western Australia, Flinders University of South Australia in Adelaide, and Brunel University in West London. He has held visiting academic positions at the University of Kent, Canterbury, and the Technical University of Budapest. He is currently Head of the Department of Information Engineering in the School of Computer Science and Engineering, the University of New South Wales, Sydney. His research is focussed on the development of automated systems for information extraction, and for the synthesis of the extracted information into humanly useful information resources in multimodal information processing systems. The primary techniques have been neural network and fuzzy logic methods. Major application areas are in hypertext for legal data and petro-physical inferencing.