Combining Simulated Annealing and Gradient Descent for Improved Convergence in Neural Networks

N.K. Treadgold and T.D. Gedeon
Department of Information Engineering
School of Computer Science & Engineering, The University of New South Wales
{ nickt | tom }@cse.unsw.edu.au

ABSTRACT

A problem with gradient descent algorithms is their convergence to poorly performing local minima. Global search algorithms address this problem, but at the cost of greatly increased training times. This work looks at combining gradient descent with the global search technique of Simulated Annealing. Simulated Annealing is added to RPROP, a powerful gradient descent algorithm for training feedforward neural networks. The resulting algorithm, SARPROP, is shown 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. A further enhancement to SARPROP is the addition of a restart training phase that allows a more thorough search of the error surface.

1 Introduction

There are two traditional methods for training feedforward neural networks: gradient descent, which includes algorithms such as Back Propagation [10] and Conjugate Gradient methods [6], and global optimisation techniques such as Simulated Annealing (SA) and Genetic Algorithms. Both suffer from problems: gradient descent methods inherently converge to local minima that may produce poor solutions, and global optimisation is computationally expensive.

The combination of gradient descent and some form of global search is the obvious solution to this dilemma. This work combines a quick and computationally cheap algorithm, RPROP, and SA, with the aims of maintaining quick convergence and reducing convergence to poor local minima. This algorithm, named SARPROP, is based on previous work [12], but has a number of extensions.

SA methods are a well known technique in training artificial neural networks, and have been applied to the Back Propagation algorithm [2] with good results in terms of speed of convergence. Other algorithms exploring the combination of gradient descent and global search are [1,11].

2 RPROP

There have been a number of refinements made to the BP algorithm [3,5] with the most successful in general being Resilient Back Propagation (RPROP) [8,9]. 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 the sign of the gradient (positive or negative). This allows the step size to be adapted without having the size of the gradient interfere with the adaptation process [9]. In a number of previous BP variants, the learning parameter, η, has been varied adaptively [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.

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 initialised to the value Δ_0. The update value is modified in the following manner: if the current gradient ($\delta E/\delta w_{ij}(t)$) multiplied by the gradient of the previous step is positive (that is, the gradient direction has remained the same), then the update value is multiplied by a value η^+ (which is greater than one). Similarly, if the gradient product is negative, the update value is multiplied by the value η^- (which is less than one). The update value remains the same if the product equals zero. 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 values: a maximum Δ_{max}, and a minimum Δ_{min}.

The RPROP algorithm has a number of advantages. It is fast to converge compared to BP and a number of other BP variants, and its performance is relatively invariant to initial parameter selection [9]. It is also only slightly more computationally complex than BP.

3 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 Simulated Annealing. SA, in general, involves the addition of a random noise factor during weight updates. 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 [2] 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 a weight when both the error gradient changes sign in successive epochs, and the magnitude of the update value is less than a 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 minimise 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 (Figure 1), while minimising the disturbance to the adaptation process. The amount of noise added decreases as the training continues.

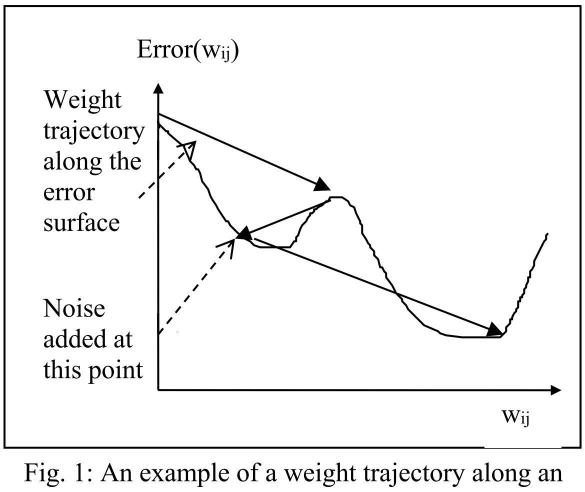
Error(w_{ij})
Weight trajectory along the error surface
Noise added at this point
w_{ij}

Fig. 1: An example of a weight trajectory along an error surface, and the result of SARPROP's noise addition.

SARPROP uses SA not only on the noise added to the weight updates, but also on the amount of weight decay the network uses. SARPROP implements weight decay by imposing a penalty term onto the error function, which results in a modification to the error gradient. The weight decay term in SARPROP is set up such that small valued weights decay more rapidly than larger weights. This form of weight decay was found to produce much better convergence properties than the one used in an earlier version of SARPROP [12] which employed a weight decay rate proportional to the weight size. The weight decay term is associated with a SA term, which results in the influence of the weight decay decreasing as training proceeds. The SARPROP error gradient is shown below:

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

where $SA = 2^{-T*epoch}$ and T = temperature

The reason for adding a weight decay term to the error function is to constrain the weights to smaller values at the beginning of training. In effect this smoothes out the error surface, reducing the chance of the network getting stuck in small local minima. In addition, convergence may be speeded up since a smoother error surface reduces the chance of gradient crossings. Gradient crossings cause the update values to decrease, which may slow learning. The search is complemented by the simultaneous addition of noise to the weight updates.

After these enhancements have been incorporated, the resulting SARPROP algorithm is shown in Figure 2, in which *r* is a random number between 0 and 1. The only parameter value requiring setting prior to training is the temperature parameter, *T*, a part of the SA term.

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. 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 weight values, so that any prior training knowledge is maintained. Since training is restarted, the SA terms are reset, and hence the error surface is once again greatly smoothed and a large noise value is reinitialised. This 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. Normally 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, when the network is restarted this temperature can be reset to allow for slower annealing. A good performing temperature schedule used for the restarts is the sequence: 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 is 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 decay to diminish sufficiently for convergence to occur. The minimum training time in epochs is given by the formula: 6/Temperature.

```
$\forall i,j$: $\Delta_{ij}(t) = \Delta_0$
$\forall i,j$: $\delta E/\delta w_{ij}(t-1) = 0$
**Repeat**
    **Compute SARPROP 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**
        **if** $(\Delta_{ij}(t-1) < 0.4* SA^2)$ **then**
            $\Delta_{ij}(t) = \Delta_{ij}(t-1) * \eta^- + 0.8*r*SA^2$
        **else**
            $\Delta_{ij}(t) = \Delta_{ij}(t-1) * \eta^-$
        $\Delta_{ij}(t) =$ **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. 2: The SARPROP algorithm.

4. 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 [9]: $\eta^+ = 1.2$, $\eta^- = 0.5$, $\Delta_{max} = 50$, $\Delta_{min} = 1\text{x}10^{-6}$. In addition, a constant value of 0.0001 was added to the derivative of the sigmoid in order to overcome the 'flat spot' problem [3] for all algorithms. The initial update value, Δ_0, was set to 0.1 for all problems. The value chosen for Δ_0 has been shown to be of little significance in regards to the performance of RPROP since it is quickly adapted [9]. All weights were initialised to random values in the range -0.7 to 0.7. A symmetric activation function (-0.5 to 0.5) was used. The temperature value used for SARPROP was chosen to maximise its performance.

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. A maximum number of training epochs was also selected, and training was halted if this number was reached. In this case the training was defined to be non-converging. 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 non-converged training runs. For classification problems the 40-20-40 threshold and margin criterion was used [3]. In using the 40-20-40 criterion, a class has been learnt 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.

4.1 Parity Data Sets

The first 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); 8-16-1 (Parity 8). The maximum training time was set at 2000 epochs. Training was halted when the 40-20-40 classification criterion was satisfied on the training set. The temperature parameter for SARPROP was set to 0.010. The Parity results are displayed in Table 1.

4.2 Iris Data Set

Comparisons were next performed on Fisher's classic Iris data set which classifies irises into three classes. The Iris data set was obtained from the UCI machine learning database [7], and consists of 120 training patterns and 30 test patterns selected at random. 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 2.

4.3 Regression Data Sets

Regression simulations were next performed on the complex interaction function, described in detail in [4], 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 [4]. 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 unit variance. For each function an independent 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 10000 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) [4], which is proportional to the total sum of squares error. The temperature parameter for SARPROP was set to 0.010. The results are shown in Tables 3 and 4 for the noise-free and noisy data sets respectively.

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

Table 1: Parity results.

	TRAINING (EPOCHS)			
	Avg.	**Median**	**NonCvg**	**Test%**
RP	569.12	396.5	5	83.33
SP	352.66	283.5	0	86.67
ReSP	352.66	283.5	0	86.67

Table 2: Iris results

5. 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 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 SAPRROP which converged in every run. In addition, for the data sets which have a test set, SARPROP can be seen to produce better generalisation results than RPROP.

	TRAINING (EPOCHS)			
	Avg.	**Median**	**NonCvg**	**FVU**
RP	9793.96	10000	48	0.052
SP	8530.22	10000	33	0.040
ReSP	7193.12	6934.5	8	0.038

Table 3: Complex interaction function results – noise free

	TRAINING (EPOCHS)			
	Avg.	**Median**	**NonCvg**	**FVU**
RP	7230.18	9800.5	24	0.070
SP	5708.28	5086.5	12	0.058
ReSP	5413.32	5508.5	3	0.066

Table 4: Complex interaction function results – noisy

ReSARPROP produces even better convergence results than SARPROP, especially on the more difficult problems such as Parity 8 and the regression data sets. 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, generalisation improvements are seen for ReSARPROP over RPROP.

The combination of noise and weight decay can be seen to be successful in allowing the SARPROP 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' [2] hypothesis that the addition of noise can allow access to paths along the error surface which allow more rapid convergence.

The reasons for SARPROP and ReSARPROP's improved generalisation results can be attributed to their use of weight decay, a form of regularisation. Weight decay restricts the type of functionality which the network can produce. It favours networks producing smoother functions, which are more likely to represent the underlying functions of real world data.

The use of SA and weight decay is similar in purpose to the technique of convolution function smoothing [11]. In this technique the error function is convoluted with a smoothing function in order to smooth the error surface, thereby removing many poor local minima. The amount of smoothing is reduced as training continues. A similar function is performed through the use of weight decay and SA.

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 also improves 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.

6. Conclusion

A Simulated Annealing addition to the RPROP algorithm, SARPROP, has been proposed. The success of the combination of noise and weight decay in increasing both 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 removing any parameters needing to be set prior to training.

References

[1] 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.

[2] 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.

[3] S.E. Fahlman, “An Empirical Study of Learning Speed in Back-Propagation Networks”, CMU-CS-88-162, Technical Report, Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1988.

[4] J. Hwang, S. Lay, R. Maechler, and D. Martin, “Regression Modeling in Back-Propagation and Projection Pursuit Learning,” *IEEE Trans. Neural Networks* Vol. 5, pp. 342-353, 1994.

[5] R.A. Jacobs, “Increased Rates of Convergence Through Learning Rate Adaption,” *Neural Networks*, vol. 1, pp. 295-307, 1988.

[6] M.F. Moller, “A Scaled Conjugate Gradient Algorithm for Fast Supervised Learning,” *Neural Networks*, vol. 6, pp. 525-533, 1993.

[7] P.M. Murphy and D.W. Aha, UCI Repository of machine learning databases, Irvine, CA: University of California, Department of Information and Computer Science, 1994.

[8] M. Riedmiller, “Rprop - Description and Implementation Details,” Technical Report, University of Karlsruhe, 1994.

[9] M.Riedmiller and H. Braun, “A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm,” *Proc. of the ICNN 93, San Francisco*, pp. 586-591, 1993.

[10] D.E. Rumelhart, G.E. Hinton, and R.J. Williams, *Learning internal representations by error propagation*. In: Rumelhart, D.E. and McClelland, J.L., (Ed.) *Parallel distributed processing: Explorations in the microstructure of cognition. Vol 1. Foundations*, Cambridge, MA: MIT Press, pp. 318-362, 1986.

[11] M.A. Styblinski and T.S. Tang, “Experiments in Nonconvex Optimisation: Stochastic Approximation with Function Smoothing and Simulated Annealing,” *Neural Networks*, vol. 3, pp. 467-483, 1990.

[12] N.K. Treadgold and T.D. Gedeon, “A Simulated Annealing Enhancement to Resilient Backpropagation,” *Proc. Int. Panel Conf. Soft and Intelligent Computing*, *Budapest*, pp. 289-293, 1996.