Merge of Evolutionary Computation with Gradient Based Method for Optimization Problems

Joel Hewlett, Bogdan Wilamowski
Electrical and Computer Engineering
Auburn University
Auburn, AL 36849, United States
Email: hewlejd@auburn.edu, wilam@ieee.org

Günhan Dündar
Electrical and Electronic Engineering
Bogazici University
Istambul, Turkey
Email: dundar@boun.edu.tr

***Abstract*— The paper describes an optimization method which combines advantages of both evolutionary computation and gradient based methods. The proposed method follows the general concept of evolutionary computation, but uses an approximated gradient for generating subsequent populations. The gradient is not explicitly computed, but is instead estimated using minimum solutions from neighboring populations. Experimental data shows that the proposed method is not only robust, but is also comparable to gradient methods with respect to speed of convergence.**

I. Introduction

Gradient based search methods are known to be very efficient, especially for cases in which the surface of the solution space is relatively smooth, with few local minima. Unfortunately, in cases where the solution space lacks this relative uniformity, gradient methods become easily trapped in the local minima commonly found on rough or complex surfaces. Methods of evolutionary computation (including genetic algorithms), on the other hand, are much more effective for traversing these more complex surfaces, and are inherently better suited for avoiding local minima. However, like the gradient based approach, evolutionary computation has it's own significant weakness. This stems from the fact that despite its reliability, solutions are often not optimal. Furthermore, both methods are known to converge very slowly [4] [5][7].

The objective behind this work is to take advantage of both methods by combining the desirable characteristics of each. Unlike standard evolutionary computation, populations are generated using the gradient, which is not directly calculated, but is instead extracted from the properties of the existing population. Several similar approaches have been undertaken along this path [1] [2][3][4][5][6], but the method which is proposed here has less computational complexity and is more suitable for online hardware training. Simple computations are repeated with every iteration, and the gradient is updated simply by subtracting the coordinates of the best solutions from the current and previous populations. The steepest decent method, which is the most commonly used gradient method, tends to approach the solution asymptotically, which results in a much slower rate of convergence. By comparison, the proposed method converges much more rapidly.

The paper is organized as follows: Section II describes the proposed method in detail. Section III offers several modifications, including methods for generating populations along the gradient and for modifying the population density. Some experimental results and statistical data are shown in section IV.

II. The Proposed Method

The proposed method is a hybrid algorithm which offers both the relative speed of gradient descent and the methodical power of evolutionary computation. Like the latter, this hybrid algorithm relies on a randomly generated population of initial points. It also shares the advantage of being an exclusively feed-forward process. What separates this approach from standard methods of evolutionary computation is the way in which the successive populations are generated. This is where the proposed method borrows more from the gradient based approach. The hybrid approach relies on the calculation of a "rough" gradient using the individual errors associated with a given population. The minimum of these errors is determined and an entirely new population is generated about the corresponding point. This offers a more directional approach to the generation of successive populations. The result is an algorithm which converges much more rapidly than the combinational approach commonly associated with genetic algorithms, while at the same time reducing the risks presented by local minima.

A. Detailed Description

The method mentioned above is best represented as a four step process. Although the algorithm technically involves only three steps per iteration, an additional step is required to begin the process.

1) Step 1: Choosing a Starting Point and Radius: Let f be a function defined in $\mathbf{R}^n$. Choose an initial center point c and radius r. An initial minimum m must also be selected. For the first iteration, let $m = c$.

2) Step 2: Generating a Random Population: With n being the number of points per population, define a set of vectors $V = \{\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_n\}$ such that

$$\mathbf{v}_{ik} = r_k X \mathbf{Y} + c_k \; for \; i = 1...n \tag{1}$$

where X is a random number ranging from 0 to 1, $\mathbf{Y}$ is a normalized random vector, and k is the current iteration.

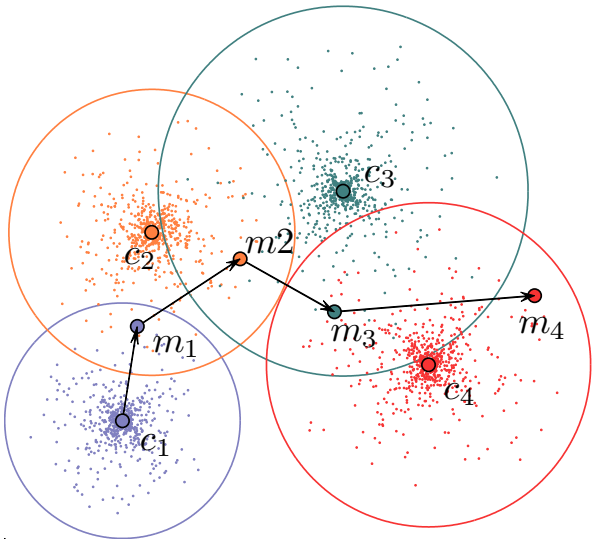

Fig. 1. Four iterations in two dimensions

Using (1) creates a set of random vectors whose members are centered about c_k with magnitudes ranging from 0 to r_k. Therefore $\mathbf{v}_{ik}$ represent positional vectors of the points p_{ik} which lie within the defined region.

3) Step 3: Approximation of the Gradient: Evaluate $f(p_{ik})$ for $i = 1...n$. If $\forall i(\min(f(p_{ik})) > f(m_k))$, repeat step two. Otherwise, let m_{k+1} be p for which $f(p) = \min(f(p_{ik}))$. An approximate gradient vector $\mathbf{g}_k$ can then be defined by taking the difference between m_{k+1} and m_k.

$$\mathbf{g}_k = m_{k+1} - m_k$$

4) Step 4: Creating a New Center and Radius: The new center should be shifted slightly so that the following population will lie in the general direction of the approximated gradient. Using

$$c_{k+1} = \alpha\mathbf{g} + m_k \; for \; \alpha \geq 1$$

allows the size of the shift to be controlled by α. If no shift is desired, α can be set to 1, in which case $c_{k+1} = m_{k+1}$. In order to ensure that m_{k+1} lies within the new region, we need $r_{k+1} \geq \|c_{k+1} - m_{k+1}\|$. This can be set using

$$r_{k+1} = \beta\|c_{k+1} - m_{k+1}\| \; for \; \beta \geq 1$$

Once r_{k+1} and c_{k+1} are determined, steps two through four are repeated until $f(m)$ is within the desired tolerance.

The two dimensional example case in fig. 1 illustrates the process through four iterations. For this particular example, $\alpha = 2$ and $\beta = 3/2$. This can be clearly seen by the way r_k, represented by the large circles, perfectly bisects $\mathbf{g}_{k-1}$ for each iteration. Also, notice that for the first iteration, the gradient vector $\mathbf{g}$ extends from c to m, whereas in all subsequent iterations $\mathbf{g}$ is defined from m_k to m_{k+1}.

III. Possible Modifications

Spherically shaped regions may not be optimal when generating successive populations. Using conical regions which extend in the direction of the approximate gradient might greatly increase the speed of convergence. The following is just one of many similar modifications.

A. Conically Shaped Regions

We have a vector $\mathbf{v}$ which we wish to use as our axis of symmetry. A hyperconical region is best represented using polar coordinates. Only the zenith angle ϕ and radial component ρ are needed. This is an especially useful way of describing a hyperconical region since it is valid for any subspace of $\mathbf{R}^n$. The drawback of this method is that it requires that the axis of symmetry lie along the zenith axis. In order to extend this method to cases using arbitrary axes of symmetry, we'll need to change to a basis U whose zenith axis is in the direction of $\mathbf{v}$. Our region can then be very easily defined using this new set of coordinate axes. If we then wish to represent a point or vector in this region in terms of the standard basis E, we can easily change bases from U back to E using a transition matrix.

Defining a new basis

The process described below first requires the formation of a new basis using the n-dimensional axial vector $\mathbf{v}$. To do this we must find $n-1$ other linearly independent vectors. The first step is to devise a generalized method for generating this set of linearly independent vectors.

1) Generating an ordinary basis: In order to generate a set of linearly independent vectors, we will first need some useful tools for determining whether they are in fact linearly independent. The three following theorems are especially relevant.

Theorem 3.1: Let $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n$ be n vectors in $\mathbf{R}^n$ and let

$$\mathbf{x}_i = (x_{1i}, x_{2i}, ..., x_{ni})^T$$

for $i = 1, ..., n$. If $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n)$, then the vectors $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n$ will be linearly independent if and only if $\mathbf{X}$ is singular.

Theorem 3.2: An n $\times$ n matrix $\mathbf{A}$ is singular if and only if

$$\det(\mathbf{A}) = 0$$

Theorem 3.3: Let $\mathbf{A}$ be an n $\times$ n matrix.

(i) If $\mathbf{A}$ has a row or column consisting entirely of zeros, then $\det(\mathbf{A}) = 0$.

(ii) If $\mathbf{A}$ has two identical rows or two identical columns, then $\det(\mathbf{A}) = 0$.

Combining these three theorems yields a particularly useful result.

Corollary 3.4: Let A be an n $\times$ n matrix. The column space of A forms an ordinary basis for $\mathbf{R}^n$ if and only if

(i) A contains no rows or columns consisting entirely of zeros.

(ii) No two rows or columns of A are identical.

The conditions from Corollary 3.4 can nearly always be satisfied by taking the standard basis E and simply replacing $\mathbf{e}_1$ with $\mathbf{v}$. The only case in which this will not result in an ordinary basis of $\mathbf{R}^n$ is when $v_1 = 0$, which can be easily remedied by replacing e_{i1} with $(v_1 + 1)$ for $i = 2, ..., n$. This method will work for any $\mathbf{v}$ except $\mathbf{0}$. To summarize,

1) Let $E = \{\mathbf{e}_1, \mathbf{e}_2, ..., \mathbf{e}_n\}$. Replace $\mathbf{e}_1$ with $\mathbf{v}$.
2) Replace e_{i1} with $(v_1 + 1)$ for $i = 2, ..., n$.

The resulting set of vectors, which we will call S, is an ordinary basis for $\mathbf{R}^n$ with $\mathbf{s}_1 = \mathbf{v}$.

Although this method does produce a basis for $\mathbf{R}^n$ relative to $\mathbf{v}$, it still lacks one essential characteristic. In order for the vector lengths and angles of separation to be preserved when changing bases, S must be orthogonal. Clearly, S is not an orthogonal basis, however this can be fixed using the Gram-Schmidt Orthogonalization Process.

The Gram-Schmidt Process

The Gram-Schmidt Process is a method for orthogonalizing a set of vectors in an inner product space, most commonly the Euclidian space $\mathbf{R}^n$. The GramSchmidt process takes a finite, linearly independent set $V = \{v_1, ..., v_n\}$ and generates an orthogonal set $V' = \{u_1, ..., u_n\}$ that spans the same subspace as V.

Theorem 3.5 (The Gram-Schmidt Process): Let $\{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$ be a basis for the inner product space V. Let

$$\mathbf{u}_1 = \left(\frac{1}{\|\mathbf{x}_1\|}\right)\mathbf{x}_1$$

and define $\mathbf{u_2}, ..., \mathbf{u_n}$ recursively by

$$\mathbf{u}_{k+1} = \frac{1}{\|\mathbf{x}_{k+1} - \mathbf{p}_k\|}(\mathbf{x}_{k+1} - \mathbf{p}_k) \quad for \quad k = 1, ..., n-1$$

where

$$\mathbf{p}_k = \langle \mathbf{x}_{k+1}, \mathbf{u}_1\rangle \mathbf{u}_1 + \langle \mathbf{x}_{k+1}, \mathbf{u}_2\rangle \mathbf{u}_2 + ... + \langle \mathbf{x}_{k+1}, \mathbf{u}_k\rangle \mathbf{u}_k$$

is the projection of $\mathbf{x}_{k+1}$ onto Span$(\mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_k)$. The set

$$\{\mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_n\}$$

is an orthonormal basis for V.

By applying Theorem 3.5 to the ordinary basis S, we obtain an orthogonal basis S' which spans the same subspace as S who's first element $\mathbf{u}_1$ shares the same axis as $\mathbf{s}_1$ and therefore lies in the direction of our initial vector $\mathbf{v}$. Thus the net effect is an orthonormal basis S' which allows for orthonormal transformations between itself and the standard basis E.

Defining a Hyperconically bounded region

With the basis S', defining a hyperconically bounded region about $\mathbf{v}$ is a trivial matter. Using hyperspherical coordinates, the region can be described using only the radial component ρ and the zenith angle ϕ. Any point obtained by manipulating the other angular components will lie on the boundary of this region. For a region described by a maximum radial component ρ_{max} and a maximum zenith angle ϕ_{max}, any point satisfying $0 \leq \rho \leq \rho_{max}$ and $0 \leq \phi \leq \phi_{max}$ is guaranteed to lie within the desired region. Now all we need to do is perform an orthonormal transformation from S' back to E.

Changing bases from S' to E

Changing bases is a relatively simple matter. A transition matrix from S' to E can easily be found. However, before this matrix can be used to change the basis of a point which lies within the predefined region, its coordinates must be converted from spherical to cartesian. Thus we must find some generalized method for making this conversion.

2) Hyperspherical to cartesian coordinate conversion: We have already defined a coordinate system in an n-dimensional Euclidean space which is analogous to the spherical coordinate system defined for 3-dimensional Euclidean space. These coordinates consist of a radial coordinate ρ, and $n-1$ angular coordinates $\phi_1, \phi_2, ..., \phi_{n-1}$. If x_k are the Cartesian coordinates, then we may define

$$x_k = \begin{cases} \rho \cdot \cos(\phi_k) & for\ k = 1 \\ \rho \cdot \prod_{i=1}^{k-1} \sin(\phi_i) \cdot \cos(\phi_k) & for\ k = 2, ..., n-1 \\ \rho \cdot \prod_{i=1}^{k-1} \sin(\phi_i) & for\ k = n \end{cases} \tag{2}$$

This offers a generalized method by which an n-dimensional vector or point in hyperspherical coordinates may be converted into its cartesian representation. Once the conversion has been made, the given point or vector can be converted from one basis to another using a transition matrix.

3) Creating a transition matrix from S' to E:

Theorem 3.6: (Transition matrices) Let $\{\mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_n\}$ be an ordered basis for $\mathbf{R}^n$, and let $\mathbf{c}$ be a coordinate vector with respect to $\{\mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_n\}$. To find the corresponding coordinate vector $\mathbf{x}$ with respect to $\{\mathbf{e}_1, \mathbf{e}_2, ..., \mathbf{e}_n\}$, simply multiply the matrix $U = [\mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_n]$ by $\mathbf{c}$.

$$\mathbf{x} = U\mathbf{c}$$

Summary

A step-by-step summary of the combined process.

Step 1: Using $[\mathbf{v}, \mathbf{e}_2, ..., \mathbf{e}_n]$, create a new basis for $\mathbf{R}^n$ by replacing e_{i1} with $v_1 + 1$ for $i = 2, ..., n$. We will call the resulting matrix S and refer to it's column space as $\{\mathbf{s}_1, \mathbf{s}_2, ..., \mathbf{s}_n\}$.

Step 2: Using $S = \{\mathbf{s}_1, \mathbf{s}_2, ..., \mathbf{s}_n\}$, let

$$\mathbf{u}_1 = \left(\frac{1}{\|\mathbf{s}_1\|}\right)\mathbf{s}_1$$

and define $\mathbf{u_2}, ..., \mathbf{u_n}$ recursively by

$$\mathbf{u}_{k+1} = \frac{1}{\|\mathbf{s}_{k+1} - \mathbf{p}_k\|}(\mathbf{s}_{k+1} - \mathbf{p}_k) \quad for \quad k = 1, ..., n-1$$

The resulting matrix $[\mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_n]$will form an orthonormal basis for $\mathbf{R}^n$ which we will call U.

Step 3: Define a hyperconical region about $\mathbf{u}_1$ by choosing a maximum radial component ρ_{max} and a maximum zenith angle ϕ_{max}.

Step 4: Generate vectors within the defined region by choosing ρ and ϕ such that $0 \leq \rho \leq \rho_{max}$ and $0 \leq \phi \leq \phi_{max}$.

Step 5: Let $\mathbf{x}$ be one of the vectors chosen in step four. Convert $\mathbf{x}$ from hyperspherical to cartesian coordinates using (2).

Step 6: Express $\mathbf{x}$ with respect to $[\mathbf{e}_1, \mathbf{e}_2, ..., \mathbf{e}_n]$ using the U as a transition matrix.

$$\mathbf{x}' = U\mathbf{x}$$

This newly generated vector $\mathbf{x}'$ has two very important characteristics, which make it particularly useful. They are:

(i) The length of $\mathbf{x}'$ is no greater than ρ_{max}

(ii) The angle of separation between $\mathbf{x}'$ and the original vector $\mathbf{v}$ is no greater than ϕ_{max}

A modification

It might be desirable to have ρ diminish as ϕ increases. This can be done by expressing ρ as a function of ϕ.

$$\rho(\phi) = \rho_{max}\left[1 - \left(\frac{\phi}{\phi_{max}}\right)^{\beta}\right]$$

The variable β can be adjusted to control the shape of the given region. As β becomes larger, the shape of the region becomes increasingly conical. Fig. 2 offers a graphical representation of how β effects the region's shape.

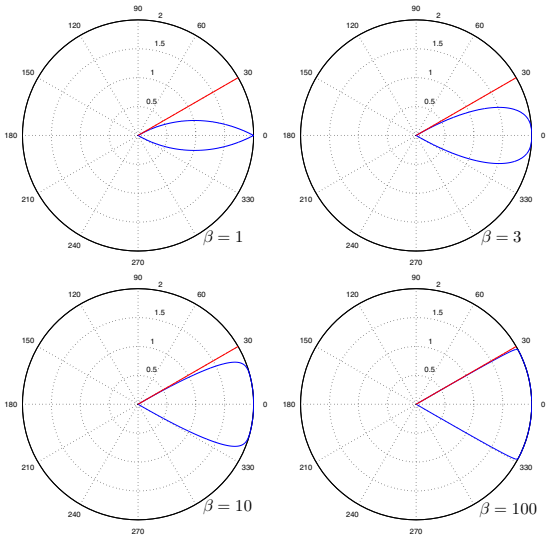

Fig. 2. $\rho_{max} = 2$, $\phi_{max} = 30°$

B. Modified Population Distribution

One of the dangers of using a uniform population distribution is its susceptibility to local minima. If the algorithm gets into a low-lying area which is larger than the current maximum population radius, the algorithm will be permanently stuck. In order to avoid this scenario, occasional points must be plotted outside the desired maximum radius.

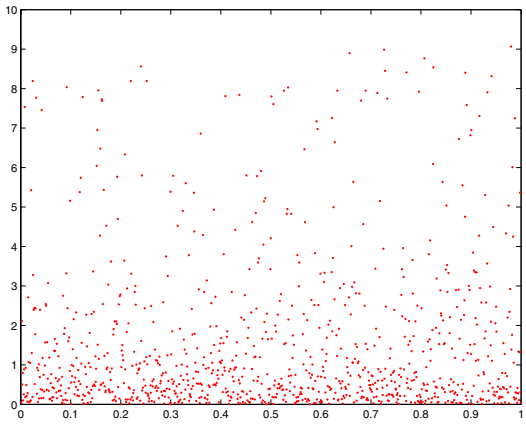

Fig. 3. A Plot of 1000 random points using (3)

One effective and efficient way of doing this is to use a population density which decreases exponentially as a function of the radius. This can be achieved by replacing $r_k X$ in (1) with

$$\frac{1}{X + \frac{1}{r_k}} - \frac{r_k}{r_k - 1} \tag{3}$$

which has a range of

$$\left(0, \frac{r_k}{1 + \frac{1}{r_k}}\right)$$

Although the upper bound for $\|v_{ik}\|$ is no longer r_k, for larger values of r_k,

$$\frac{r_k}{1 + \frac{1}{r_k}} \approx r_k$$

Figure 3 shows how (3) effects density.

IV. Experimental Results

Two variations of the algorithm were used for testing. The first set of tests were performed using the standard algorithm as described in section II. The second set used the modified population distribution from section III-B. The algorithms were then tested using population sizes of 5, 10, and 20 points per iteration. Statistical Data was compiled for both algorithms using 100 runs for each population size.

Due to the common practical application of optimization in the field of artificial neural networks, initial tests were made using the algorithm for training a simple two layer network. The network was then retrained using a standard training method for comparison. Statistical data and error plots are included for each test, including error back propagation, which was the compared method. All figures and statistics were taken for successful runs only, with the exception of success rate, which was defined as the ratio of successful runs to non-convergent ones. From the results it is clear that both algorithms perform comparably to EBP, with the modified version being by far the most consistent.

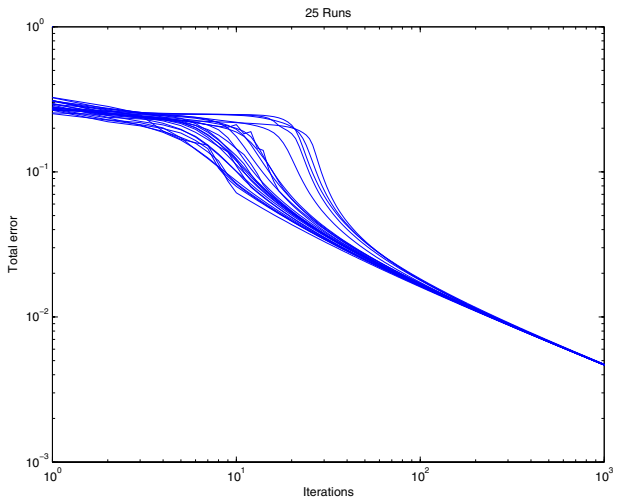

Fig. 4. Error back propagation using $\alpha = 0.5$

Error back propagation

Gain	5	
Learning constant	0.5	
Total runs	100	
Min run	138	ite
Max run	259	ite
Ave. run	152	ite
Standard deviation	15	ite
Success rate	66	%

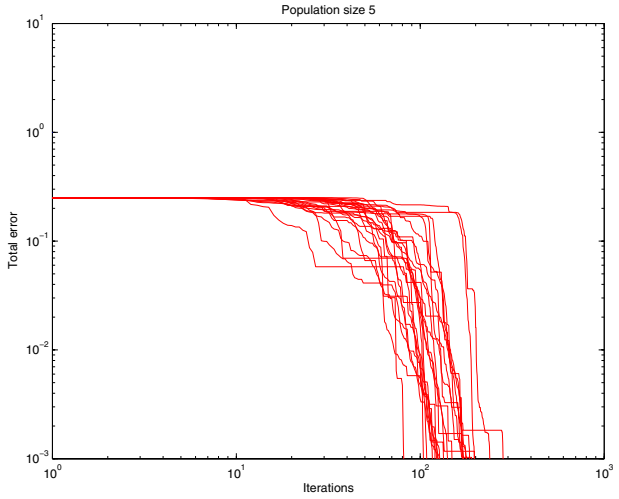

Fig. 5. Feed forward algorithm using modified distribution

Modified FF algorithm

Population	5	
Initial radius	5	
Total runs	100	
Min run	57	ite
Max run	376	ite
Ave. run	105	ite
Standard deviation	47	ite
Success rate	82	%

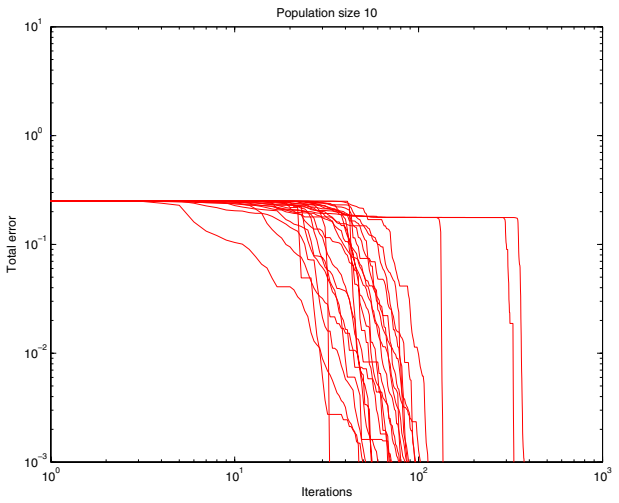

Fig. 6. Feed forward algorithm using modified distribution

Modified FF algorithm

Population	10	
Initial radius	5	
Total runs	100	
Min run	21	ite
Max run	302	ite
Ave. run	67	ite
Standard deviation	45	ite
Success rate	81	%

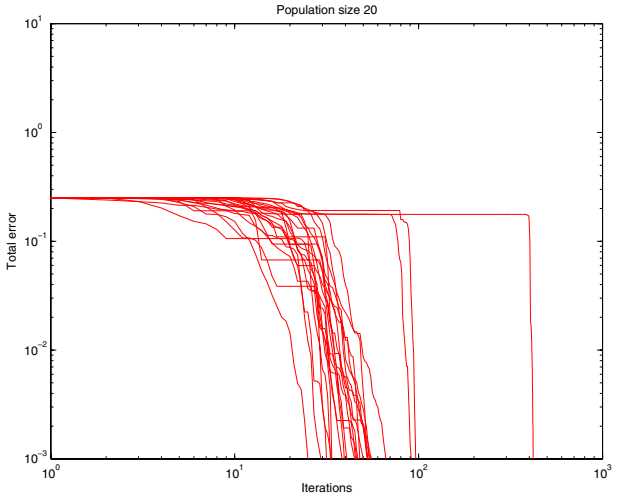

Fig. 7. Feed forward algorithm using modified distribution

Modified FF algorithm

Population	20	
Initial radius	5	
Total runs	100	
Min run	17	ite
Max run	84	ite
Ave. run	36	ite
Standard deviation	11	ite
Success rate	81	%

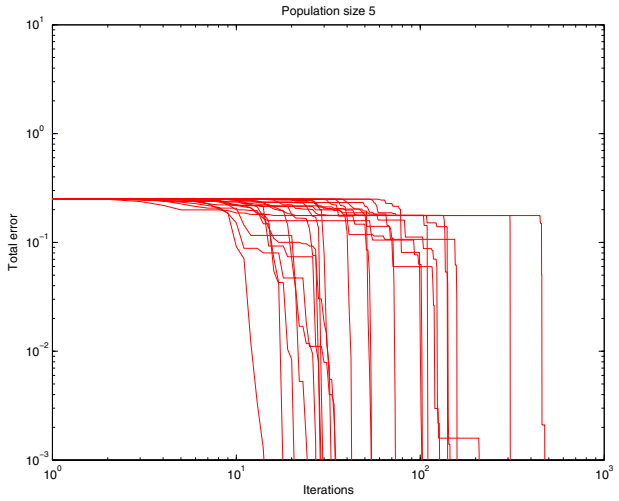

Fig. 8. Feed forward algorithm using $\beta = 1.5$, $\alpha = 2$

Standard FF algorithm

Population	5	
Min run	17	ite
Max run	301	ite
Ave. run	82	ite
Success rate	55	%

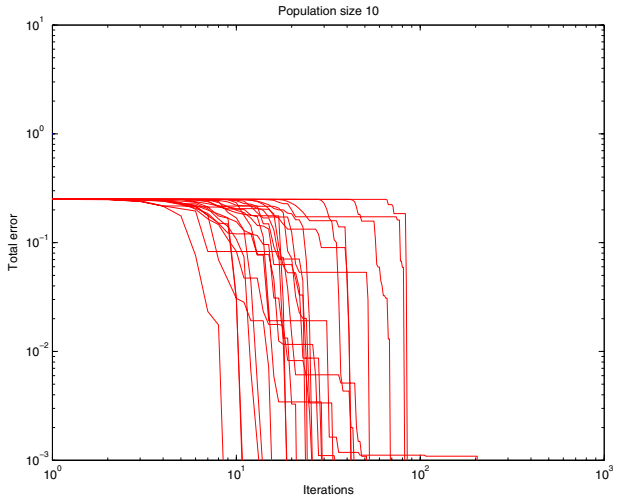

Fig. 9. Feed forward algorithm using $\beta = 1.5$, $\alpha = 2$

Standard FF algorithm

Population	10	
Min run	11	ite
Max run	159	ite
Ave. run	34	ite
Success rate	66	%

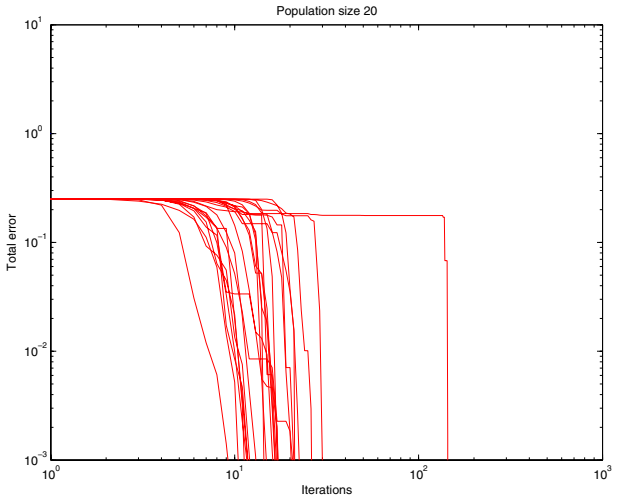

Fig. 10. Feed forward algorithm using $\beta = 1.5$, $\alpha = 2$

Standard FF algorithm

Population	20	
Min run	9	ite
Max run	197	ite
Ave. run	29	ite
Success rate	70	%

V. Conclusion

A method of indirect gradient computation to speed up the evolutionary optimization process was presented. New populations of solutions are randomly generated along the estimated gradient line. The method was verified using examples to find optimal weights in neural networks. It was shown that with little computational complexity, the proposed algorithm converges faster than the traditionally used method of error back propagation. Interestingly, the proposed algorithm reaches the optimum solution very rapidly and does not exhibit the asymptotic convergence typical of error back propagation.

References

[1] H. A. Abbass, A. M. Bagirov, J. Zhang, "The discrete gradient evolutionary strategy method for global optimization," *Congress on Evolutionary Computation - CEC'03* Volume 1, 8-12 Dec. 2003 pp. 435 - 442 Vol.1

[2] M. Bundzel and P. Sincak, "Combining Gradient and Evolutionary Approaches to the Artificial Neural Networks Training According to Principles of Support Vector Machines," *IEEE International Joint Conference on Neural Networks - IJCNN'06*, 16-21 July 2006, pp. 2068 - 2074.

[3] X. Hu; Z. Huang; Z. Wang "Hybridization of the multi-objective evolutionary algorithms and the gradient-based algorithms," *Congress on Evolutionary Computation - CEC'03*, Volume 2, 8-12 Dec. 2003 pp. 870 - 877 Vol.2

[4] M. Manic and B. Wilamowski, "Random Weights Search in Compressed Neural Networks using Overdetermined Pseudoinverse," *Proc. of the IEEE Interantional Symposium on Industrial Electronics - ISIE'03* Rio de Janeiro, Brazil, June 9-11, 2003, pp. 678 - 683.

[5] M. Manic and B. Wilamowski "Robust Neural Network Training Using Partial Gradient Probing," *IEEE International Conference on Industrial Informatics - INDIN'03* Banff, Alberta, Canada, August 21-24, 2003, pp. 175 - 180.

[6] J. Y. Wen, Q. H. Wu, L. Jiang, S.J. Cheng, "Pseudo-gradient based evolutionary programming,", *Electronics Letters* Volume 39, Issue 7, 3 April 2003 pp. 631 - 632

[7] B. Wilamowski "Neural Networks and Fuzzy Systems," *The Microelectronic Handbook* CRC Press -2006 chapters 124.1 to 124.8.