Research Article
SOMO-m Optimization Algorithm with Multiple Winners

Wei Wu and Atlas Khan

Department of Applied Mathematics, Dalian University of Technology, 116024 Dalian, China

Correspondence should be addressed to Atlas Khan, atlas900@gmail.com

Received 1 April 2012; Revised 6 June 2012; Accepted 6 June 2012

1. Introduction

Self-organizing map (SOM) neural networks have been widely applied in information sciences [1–7]. In particular, an SOM-based optimization (SOMO) algorithm is proposed in [8, 9] to find a winning neuron, through a competitive learning process, that stands for the minimum of an objective function. They compared the SOM-based optimization (SOMO) algorithm with genetic algorithms [10, 11] and particle swarm optimization algorithm [12–17], and they showed that the SOM-based optimization (SOMO) algorithm can locate the minimum much faster than the genetic algorithm and the particle swarm optimization algorithm.

The aim of this paper is to generalize SOMO algorithm to so called SOMO-m algorithm to find m winning neurons in a single learning process. A separation technique is introduced...
in the learning process to prevent the accumulation of the \( m \)-winning neurons. Numerical experiments show that, for \( m \geq 2 \), SOMO-\( m \) algorithm converges faster than SOMO algorithm when they are used for finding the minimum of functions. A more important merit of SOMO-\( m \) algorithm with \( m \geq 2 \) is that it can find two or more minimums simultaneously in a single learning iteration process, while the original SOMO algorithm has to fulfil the same task much less efficiently by restarting the learning iteration process twice or more times.

This paper is organized as follows. In Section 2, a brief introduction of SOMO algorithm is given. Our SOMO-\( m \) algorithm is proposed in Section 3. Section 4 is devoted to some supporting numerical simulations.

2. Original SOM-Based Optimization (SOMO) Algorithm

Self-organizing map (SOM) is an unsupervised learning algorithm proposed by Kohonen [18–20]. The principal goal of the SOM algorithm is to map an incoming pattern in a higher dimensional space into a lower (usually one or two) dimensional space, and perform this transformation adaptively in a topological ordered fashion. The applications of SOM range widely from simulations used for the purpose of understanding and modeling of computational maps in the brain to subsystems for engineering applications such as speech recognition, vector quantization and cluster analysis [18–26].

Different from the usual SOM algorithm, an SOM-Based optimization (SOMO) algorithm is introduced in Su and Zhao [8] for continuous optimization. In the following, let us describe SOMO algorithm [8] used for finding the minimum point \( x^* \) of a function \( f(x) \), \( x \in \mathbb{R}^n \). The SOMO network contains \( M \times N \) neurons arranged as a two dimensional array. For each neuron \((i,j)\), its weight \( W_{i,j} \) is a vector in \( \mathbb{R}^n \), where \( 1 \leq i \leq M \) and \( 1 \leq j \leq N \) for some positive integers \( M \) and \( N \). For a special input \( \bar{x} = (x_1, \ldots, x_n) = (1,1,\ldots,1) \in \mathbb{R}^n \), the winner out of all the neurons is defined as

\[
(i^*, j^*) = \arg \min_{1 \leq i \leq M, 1 \leq j \leq N} f \left( \left( W_{i,j} \right)_1 \times x_1, \ldots, \left( W_{i,j} \right)_n \times x_n \right)
\]

\[
= \arg \min_{1 \leq i \leq M, 1 \leq j \leq N} f \left( \left( W_{i,j} \right)_1, \ldots, \left( W_{i,j} \right)_n \times 1 \right) 
\]

\[
= \arg \min_{1 \leq i \leq M, 1 \leq j \leq N} f \left( W_{i,j} \right). 
\]

The idea of SOM training is applied to the network such that the weight \( W_{i^*,j^*} \) of the winner will get closer and closer to the minimum point \( x^* \) during the iterative training process. The detail of the training process is as follows:

Step 1. Initialization.

Substep 1 (Initialization of the neurons on the four corners). The weight vectors of the neurons on the corners are initialized as follows:

\[
W_{1,1} = (l_1, l_2, \ldots, l_n)^T, \\
W_{M,N} = (h_1, h_2, \ldots, h_n)^T,
\]
\[
\begin{align*}
W_{1,N} &= (l_1, l_2, \ldots, l_{\lfloor n/2 \rfloor}, h_{\lfloor n/2 \rfloor+1}, \ldots, h_n)^T, \\
W_{M,1} &= (h_1, h_2, \ldots, h_{\lfloor n/2 \rfloor}, l_{\lfloor n/2 \rfloor+1}, \ldots, l_n)^T.
\end{align*}
\] (2.2)

Here, the two points \((l_1, l_2, \ldots, l_n)^T\) and \((h_1, h_2, \ldots, h_n)^T\) are randomly chosen and far enough from each other.

**Substep 2** (Initialization of the neurons on the four edges). The initialization of the weights of the neurons on the four edges is as follows:

\[
\begin{align*}
W_{1,j} &= \frac{W_{1,N} - W_{1,1}}{N-1} (j - 1) + W_{1,1} \\
&= \frac{j - 1}{N-1} W_{1,N} + \frac{N - j}{N-1} W_{1,1}, \\
W_{M,j} &= \frac{W_{M,N} - W_{M,1}}{N-1} (j - 1) + W_{M,1} \\
&= \frac{j - 1}{N-1} W_{M,N} + \frac{N - j}{N-1} W_{M,1}, \\
W_{i,1} &= \frac{W_{M,1} - W_{1,1}}{M-1} (i - 1) + W_{1,1} \\
&= \frac{i - 1}{M-1} W_{M,1} + \frac{M - i}{M-1} W_{1,1}, \\
W_{i,N} &= \frac{W_{M,N} - W_{1,N}}{M-1} (i - 1) + W_{1,N} \\
&= \frac{i - 1}{M-1} W_{M,N} + \frac{M - i}{M-1} W_{1,N},
\end{align*}
\] (2.3)

where \(i = 2, \ldots, M - 1\) and \(j = 2, \ldots, N - 1\).

**Substep 3** (Initialization of the remaining neurons).

\[
\begin{align*}
W_{i,j} &= \frac{W_{i,N} - W_{i,1}}{N-1} (j - 1) + W_{i,1} \\
&= \frac{j - 1}{N-1} W_{i,N} + \frac{N - j}{N-1} W_{i,1}.
\end{align*}
\] (2.4)

where \(i = 2, \ldots, M - 1\) and \(j = 2, \ldots, N - 1\).

**Substep 4** (Random noise). A small amount of random noise is added for each weight so as to keep the weight vectors from being linearly dependent as follows:

\[
W_{i,j} = W_{i,j} + \xi
\] (2.5)

for \(1 \leq i \leq M\) and \(1 \leq j \leq N\), where \(\xi\) denotes a small random noise.
Step 2. Winner finding.

\[
(i^*, j^*) = \arg \min_{1 \leq i \leq M, 1 \leq j \leq N} f(W_{i,j})
\] (2.6)

Step 3. Weights updating. The weights of the winner and its neighbors are adjusted by the following formula:

\[
W_{i,j}(t+1) = W_{i,j}(t) + \eta \beta(i^*, j^*, i, j) \left[ W_{i^*, j^*}(t) - W_{i,j}(t) \right] \\
+ \lambda (1 - \beta(i^*, j^*, i, j)) p
\]

for \(1 \leq i \leq M, 1 \leq j \leq N, t = 0, 1, \ldots,\) (2.7)

where the parameters \(\eta\) and \(\lambda\) are real-valued constants which can be either constants predefined by the user or time-varying parameters which decrease gradually when time \(t\) increases. \(d(i^*, j^*, i, j)\) is the lateral distance between winning neuron \((i^*, j^*)\) and neuron \((i, j)\); the randomly chosen vector \(p\) is called the perturbation vector, and

\[
\beta(i^*, j^*, i, j) = 1 - \frac{d(i^*, j^*, i, j)}{\sqrt{M^2 + N^2}}.
\] (2.8)

Step 4. Go to step 2 until a prespecified number of generations is achieved, or some kind of termination criterion is satisfied.

3. SOMO-\(m\) Algorithm

In this section, let us present our SOMO-\(m\) algorithm. We divide the section into two subsections, dealing with the training algorithms for finding one minimum and two minima of a function, respectively.

3.1. SOMO-\(m\) Algorithm for Finding One Minimum

The SOMO-\(m\) algorithm for finding one minimum of a function \(f(x)\) is as follows:

Step 1. The initialization of SOMO-\(m\) algorithm is the same as that of SOMO algorithm.
Step 2. This step aims to find $m$ winning neurons, denoted by $(i_1^*, j_1^*), (i_2^*, j_2^*), \ldots, (i_m^*, j_m^*)$ with the best objective function values among the neurons as follows:

\[
(i_1^*, j_1^*) = \arg\min_{1 \leq i \leq M, 1 \leq j \leq N} f(W_{i,j}), \\
(i_2^*, j_2^*) = \arg\min_{i \in \{1 \ldots M\} \setminus S_1, j \in \{1 \ldots N\} \setminus T_1} f(W_{i,j}), \\
\vdots \\
(i_m^*, j_m^*) = \arg\min_{i \in \{1 \ldots M\} \setminus S_m, j \in \{1 \ldots N\} \setminus T_m} f(W_{i,j}),
\]

(3.1)

where

\[
S_1 = \{i : |i - i_1^*| \leq R_1\}, \quad T_1 = \{j : |j - j_1^*| \leq R_1\}, \\
S_2 = \{i : |i - i_2^*| \leq R_2\}, \quad T_2 = \{j : |j - j_2^*| \leq R_2\}, \\
\vdots \\
S_m = \{i : |i - i_m^*| \leq R_m\}, \quad T_m = \{j : |j - j_m^*| \leq R_m\}, \\
S^m = (S_1 \cup \ldots \cup S_m), \quad T^m = (T_1 \cup \ldots \cup T_m),
\]

(3.2) (3.3) (3.4) (3.5)

and $R_1, R_2, \ldots, R_m$ are suitably chosen sizes of neighborhoods.

Step 3. The weights of the winners and its neighbors are adjusted by the following formula:

\[
W_{i,j}(t+1) = W_{i,j}(t) + \sum_{r=1}^{m} \eta_r \beta(i_1^*, j_1^*, i, j) \left[ W_{i,j}(t) - W_{i,j}(t) \right] \\
+ \sum_{r=1}^{m} \lambda_r (1 - \beta(i_1^*, j_1^*, i, j)) p \quad 1 \leq i \leq M, 1 \leq j \leq N,
\]

(3.6)

where

\[
\beta(i_1^*, j_1^*, i, j) = 1 - \frac{d(i_1^*, j_1^*, i, j)}{\sqrt{M^2 + N^2}}.
\]

(3.7)

The $\eta_r$ and $\lambda_r$ are real-valued parameters which can be constants predefined by the user or time-varying parameters, which decreases gradually with increasing time $t$, and $p$ is a perturbation vector.

Step 4. Go to step 2 until a prespecified number of generations is achieved or some kind of termination criteria is satisfied.
We first remark that in the above Step 2, we require the \( m \) winners to be away from each other by using the neighborhoods \( S_m \) and \( T_m \), such that the winners can spread around rather than accumulate in one point. Secondly, we find in our numerical experiments that it is OK to choose \( \lambda_r \) and \( \eta_r \) as either small constants independent of time \( t \) or small variables decreasing with increasing time \( t \). Thirdly, we find that it is better to choose \( \eta_1 < \eta_2 \leq \cdots \leq \eta_m \).

The reason for this may be the following: A smaller constant \( \eta_1 \) might help for the first winner to converge to the minimum, while the larger \( \eta_r \) for \( r > 1 \) might help for the other winners to reach a larger searching area.

### 3.2. SOMO-2 Algorithm for Two Minima

Here we discuss SOMO-2 algorithm for finding two minima of a function simultaneously. It is an easy matter to generalize the algorithm for finding three or more minima. SOMO-2 algorithm has the same training steps as those in the last subsection with \( m = 2 \) for finding one minimum, except the step of weights updating process, which is as follows:

For the neurons \((i, j)\) in the neighborhood of the first winner \((i^*_1, j^*_1)\) satisfying \(p_1 \leq i \leq p_2, q_1 \leq j \leq q_2\), where

\[
\begin{align*}
p_1 &= \max(i^*_1 - R_1, 1), \\
p_2 &= \min(i^*_1 + R_1, M),
\end{align*}
\]

\(q_1 = \max(j^*_1 - R_1, 1),\)

\(q_2 = \min(j^*_1 + R_1, N).\)  

The weights updating rule is

\[
W_{i,j}(t + 1) = W_{i,j}(t) + \eta_1 \beta_1(i^*_1, j^*_1, i, j) \left[ W_{i^*_1,j^*_1}(t) - W_{i,j}(t) \right] + \lambda_1 (1 - \beta_1(i^*_1, j^*_1, i, j)) p, \tag{3.9}
\]

where

\[
\beta_1(i^*_1, j^*_1, i, j) = 1 - \frac{d(i^*_1, j^*_1, i, j)}{\sqrt{M^2 + N^2}}. \tag{3.10}
\]

For the rest \((i, j)\) neurons,

\[
W_{i,j}(t + 1) = W_{i,j}(t) + \eta_2 \beta_2(i^*_2, j^*_2, i, j) \left[ W_{i^*_2,j^*_2}(t) - W_{i,j}(t) \right] + \lambda_2 (1 - \beta_2(i^*_2, j^*_2, i, j)) p, \tag{3.11}
\]

where

\[
\beta_2(i^*_2, j^*_2, i, j) = 1 - \frac{d(i^*_2, j^*_2, i, j)}{\sqrt{M^2 + N^2}}. \tag{3.12}
\]

We see that, in the training, the second winner \((i^*_2, j^*_2)\) does not affect the neighboring neurons of the first winner \((i^*_1, j^*_1)\), but does affect all the other neurons. Therefore, hopefully
the first winner will converges to the minimum point of the function \( f(x) \), while the second winner converges to another minimum point.

4. Simulation Results

4.1. Objective Functions

In this subsection, we use our SOMO-\( m \) methods to minimize the following functions.

1. Step function

\[
 f(x) = \sum_{i=1}^{30} (|x_i + 0.5|)^2. \tag{4.1}
\]

2. Griewant function

\[
 f(x) = \frac{1}{4000} \sum_{i=1}^{30} (x_i - 100)^2 - \prod_{i=1}^{10} \cos \left( \frac{x_i - 100}{\sqrt{i}} \right) + 1. \tag{4.2}
\]

3. Giunta function

\[
 f(x) = \sum_{i=1}^{30} \sin \left( \frac{16}{15} x_i - 1 \right) + \sin^2 \left( \frac{16}{15} x_i - 1 \right) + \frac{1}{50} \sin \left( 4 \frac{16}{15} x_i - 1 \right) + 0.3. \tag{4.3}
\]

4. \( U_1 \) function

\[
 f = -\exp(-8 \sin(\pi x)) - \exp(9.85 \sin(2\pi y)). \tag{4.4}
\]

5. \( U_2 \) function

\[
 f = \exp(-8 \sin(\pi x) + \sin(\pi y)) - \exp(\sin(2.5\pi x) + \sin(\pi y)). \tag{4.5}
\]

4.2. Parameters of Simulation

In Table 1 we present the global minima, dimensions, and the upper bound of the number of generations for optimization algorithms. Figure 1 illustrates the graphs of these five functions in two dimensional space. The neurons are arranged as a 30 \( \times \) 30 array, namely, \( M = N = 30 \). For SOMO algorithm, we set \( \eta = 0.2 \) and \( \lambda = 0.01 \); for SOMO-2, \( \eta_1 = 0.2, \eta_2 = 0.3, \lambda_1 = 0.01 \), and \( \lambda_2 = 0.001 \); for SOMO-3, \( \eta_1 = 0.2, \eta_2 = 0.3, \eta_3 = 0.35, \lambda_1 = 0.01, \lambda_2 = 0.001, \) and \( \lambda_3 = 0.0005 \).
Table 1: Parameters for five functions.

<table>
<thead>
<tr>
<th>Test function</th>
<th>Dimensions</th>
<th>Initial range</th>
<th>Minimum</th>
<th>Number of generations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step</td>
<td>10</td>
<td>$-100 \leq x_i \leq 100$</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>Griewank</td>
<td>10</td>
<td>$-600 \leq x_i \leq 600$</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>Giunta</td>
<td>30</td>
<td>$-10 \leq x_i \leq 10$</td>
<td>$\approx 0.9$</td>
<td>100</td>
</tr>
<tr>
<td>$U_1$ function</td>
<td>2</td>
<td>$0 \leq x_i \leq 1$</td>
<td></td>
<td>100</td>
</tr>
<tr>
<td>$U_2$ function</td>
<td>2</td>
<td>$0 \leq x_i \leq 1$</td>
<td></td>
<td>100</td>
</tr>
</tbody>
</table>

Figure 1: Graphs of the five objective functions.
Table 2: Comparison of SOMO, SOMO-2, and SOMO-3 for finding one minimum.

<table>
<thead>
<tr>
<th>Test function</th>
<th>Algorithm</th>
<th>Mean</th>
<th>Standard deviation</th>
<th>Time mean (s)</th>
<th>Time SD (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SOMO</td>
<td>$1.6778e-016$</td>
<td>$2.4504e-016$</td>
<td>$1.3813$</td>
<td>$1.4008$</td>
</tr>
<tr>
<td></td>
<td>SOMO-2</td>
<td>$1.1321e-019$</td>
<td>$8.3427e-020$</td>
<td>$1.0973$</td>
<td>$1.1419$</td>
</tr>
<tr>
<td></td>
<td>SOMO-3</td>
<td>$7.8070e-020$</td>
<td>$5.7362e-020$</td>
<td>$0.8743$</td>
<td>$0.8686$</td>
</tr>
<tr>
<td></td>
<td>SOMO</td>
<td>$5.1773e-015$</td>
<td>$2.8126e-016$</td>
<td>$1.7502$</td>
<td>$1.1681$</td>
</tr>
<tr>
<td></td>
<td>SOMO-2</td>
<td>$2.8126e-016$</td>
<td>$3.7420e-016$</td>
<td>$1.1681$</td>
<td>$1.1324$</td>
</tr>
<tr>
<td></td>
<td>SOMO-3</td>
<td>$2.2204e-017$</td>
<td>$6.14813e-017$</td>
<td>$0.9849$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SOMO</td>
<td>$0.9670$</td>
<td>$1.4901e-009$</td>
<td>$1.0116$</td>
<td>$1.0049$</td>
</tr>
<tr>
<td></td>
<td>SOMO-2</td>
<td>$0.9670$</td>
<td>$9.2269e-011$</td>
<td>$0.7087$</td>
<td>$0.7053$</td>
</tr>
<tr>
<td></td>
<td>SOMO-3</td>
<td>$0.9670$</td>
<td>$5.8655e-012$</td>
<td>$0.6217$</td>
<td>$0.6451$</td>
</tr>
<tr>
<td></td>
<td>SOMO</td>
<td>$-1.896401167771170e + 004$</td>
<td>$1.018269572071048e - 007$</td>
<td>$0.8009$</td>
<td>$0.7584$</td>
</tr>
<tr>
<td></td>
<td>SOMO-2</td>
<td>$-1.896401167771132e + 004$</td>
<td>$2.722830731013220e - 009$</td>
<td>$0.6168$</td>
<td>$0.6438$</td>
</tr>
<tr>
<td></td>
<td>SOMO-3</td>
<td>$-1.896401167771335e + 004$</td>
<td>$2.146099170750331e - 011$</td>
<td>$0.4657$</td>
<td>$0.4778$</td>
</tr>
<tr>
<td>Giunta</td>
<td>SOMO</td>
<td>$-7.36465824205229$</td>
<td>$9.781475238601051e - 014$</td>
<td>$2.1060$</td>
<td>$2.1278$</td>
</tr>
<tr>
<td></td>
<td>SOMO-2</td>
<td>$-7.36465824208548$</td>
<td>$2.101813757029311e - 015$</td>
<td>$1.5919$</td>
<td>$1.5198$</td>
</tr>
<tr>
<td></td>
<td>SOMO-3</td>
<td>$-7.36465824208548$</td>
<td>$2.39424084931165e - 015$</td>
<td>$0.9831$</td>
<td>$0.9525$</td>
</tr>
</tbody>
</table>

4.3. Simulations of SOMO-m Algorithm for One Minimum

In this subsection, we investigate the performance of SOMO-m algorithm for finding one minimum. For each function, each algorithm conducted 30 runs. The stop criterion for each run is that either 100 generations are iteratively generated, or before then if the difference of two successive minima in the iteration process is less than a prescribed tolerance $\epsilon > 0$. The best solutions found for each function and each run were recorded and, for instance, the mean column in Table 2 presents the average of the 30 best solutions of the 30 runs, respectively. To measure and compare the performance of SOMO-3, the mean time, that is, the average of processing time over 30 runs is recorded. Table 2 tabulates the comparison of the simulation results of SOMO, SOMO-2, and SOMO-3 algorithms. The mean column and the standard deviation (SD) column represent the mean and the standard deviation (SD) of the best solutions of 30 runs. The highlighted (bold) entries correspond to best solutions found by SOMO-3. Figure 2 shows the performances of SOMO, SOMO-2, and SOMO-3 algorithms in typical runs.

Now, we discuss the accuracy of the algorithms. We measure the errors for each algorithm over the 30 runs as follows:

$$E = \frac{1}{30} \sum_{l=1}^{30} \left| x_l^j - x^* \right|,$$

where $x_l^j$ is the approximate solution of $l$th run, and $x^*$ is the real minimum of the function. Table 3 tabulates the errors of SOMO, SOMO-2, and SOMO-3 algorithms, respectively.
Figure 2: Best performance curves for SOMO, SOMO-2, and SOMO-3 algorithms, respectively.
### Table 3: Error of SOMO, SOMO-2, and SOMO-3.

<table>
<thead>
<tr>
<th>Test function</th>
<th>Algorithm</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SOMO</td>
<td>7.2868e – 017</td>
</tr>
<tr>
<td>Step</td>
<td>SOMO-2</td>
<td>4.0385e – 019</td>
</tr>
<tr>
<td></td>
<td>SOMO-3</td>
<td>5.3815e – 020</td>
</tr>
<tr>
<td>Griewank</td>
<td>SOMO</td>
<td>6.4936e – 017</td>
</tr>
<tr>
<td></td>
<td>SOMO-2</td>
<td>2.2973e – 019</td>
</tr>
<tr>
<td></td>
<td>SOMO-3</td>
<td>6.8401e – 020</td>
</tr>
<tr>
<td>Giunta</td>
<td>SOMO</td>
<td>0.067056308053595</td>
</tr>
<tr>
<td></td>
<td>SOMO-2</td>
<td>0.067056308053587</td>
</tr>
<tr>
<td></td>
<td>SOMO-3</td>
<td>0.067056308053585</td>
</tr>
</tbody>
</table>

### Table 4: Comparison of SOMO-2 and the original SOMO algorithms for finding two minima.

<table>
<thead>
<tr>
<th>Test function</th>
<th>Algorithm</th>
<th>Mean</th>
<th>Standard deviation</th>
<th>Time mean (s)</th>
<th>Time SD (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SOMO</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$U_2$ function</td>
<td>First minimum</td>
<td>-7.36465824205229</td>
<td>9.781475238601051e – 014</td>
<td>2.1060</td>
<td>2.1278</td>
</tr>
<tr>
<td></td>
<td>Second minimum</td>
<td>-5.99322845918991</td>
<td>0.00379247207700</td>
<td>2.9716</td>
<td>2.865</td>
</tr>
<tr>
<td>SOMO-2</td>
<td>First minimum</td>
<td>-7.36465824124180</td>
<td>4.047965312150572e – 009</td>
<td>1.4967</td>
<td>1.4237</td>
</tr>
<tr>
<td></td>
<td>Second minimum</td>
<td>-6.07080647154294</td>
<td>6.002561408745671e – 013</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SOMO</td>
<td>First minimum</td>
<td>-1.895835670663015e + 004</td>
<td>3.927781801202260e – 010</td>
<td>1.2414</td>
<td>1.1168</td>
</tr>
<tr>
<td></td>
<td>Second minimum</td>
<td>-1.895837311768572e + 004</td>
<td>4.152186941602799e – 009</td>
<td>1.1343</td>
<td>1.0938</td>
</tr>
<tr>
<td>$U_1$ function</td>
<td>First minimum</td>
<td>-1.895837311745155e + 004</td>
<td>2.482103550304831e – 007</td>
<td>0.5536</td>
<td>0.5631</td>
</tr>
<tr>
<td></td>
<td>Second minimum</td>
<td>-1.895837311768680e + 004</td>
<td>2.906907306783492e – 009</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

We draw the following conclusions from the simulations in this subsection for finding one minimum.

1. SOMO-3 algorithm shows the best results for all functions based on the comparisons of the means of the best objective value and the processing time.
2. As shown in Figure 2, the SOMO-3 algorithm locates the minima faster than SOMO and SOMO-2 algorithms.
3. In Table 3, the bold entries show that the error corresponding to SOMO-3 algorithm is smaller than those of SOMO and SOMO-2 algorithms.

### 4.4. Simulations of SOMO-2 Algorithm for Two Minima

In this subsection, we present the simulation results for the case of finding simultaneously two minima of a function. The best solutions found for each run after prespecified number of generations were recorded. Table 4 tabulates the comparison of the simulation results. The
mean column and the standard deviation column represents the mean and the standard deviation of the best solutions of 30 runs.

Based on observation from Table 4, our conclusion for finding simultaneously two minima of a function is as follows:

The proposed SOMO-2 algorithm can find two minima of a function simultaneously in a single learning iteration process, while the original SOM-based optimization (SOMO) algorithm has to fulfil the same task much less efficiently by restarting the learning iteration process twice or more times.

Acknowledgments

The authors wish to thank the Associate Editor and the anonymous reviewers for their helpful and interesting comments. This work was supported by National Science Foundation of China (11171367) and the Fundamental Research Funds for the Central Universities of China.

References


Submit your manuscripts at http://www.hindawi.com