Neural Process Lett (2007) 26:145–158
DOI 10.1007/s11063-007-9048-7

Comparison of Stochastic Global Optimization Methods to Estimate Neural Network Weights

Lonnie Hamm · B. Wade Brorsen · Martin T. Hagan

Published online: 1 September 2007

Abstract Training a neural network is a difficult optimization problem because of numerous local minima. Many global search algorithms have been used to train neural networks. However, local search algorithms are more efficient with computational resources, and therefore numerous random restarts with a local algorithm may be more effective than a global algorithm. This study uses Monte-Carlo simulations to determine the efficiency of a local search algorithm relative to nine stochastic global algorithms when using a neural network on function approximation problems. The computational requirements of the global algorithms are several times higher than the local algorithm and there is little gain in using the global algorithms to train neural networks. Since the global algorithms only marginally outperform the local algorithm in obtaining a lower local minimum and they require more computational resources, the results in this study indicate that with respect to the specific algorithms and function approximation problems studied, there is little evidence to show that a global algorithm should be used over a more traditional local optimization routine for training neural networks. Further, neural networks should not be estimated from a single set of starting values whether a global or local optimization method is used.

Keywords Evolutionary algorithms · Function approximation · Neural networks · Simulated annealing · Stochastic global optimization

L. Hamm
Straumur-Burdaras Investment Bank, London, UK

B. W. Brorsen (✉)
Department of Agricultural Economics, Oklahoma State University, 414 Agricultural Hall, Stillwater, OK 74078-6026, USA
e-mail: wade.brorsen@okstate.edu

M. T. Hagan
School of Electrical and Computer Engineering, Oklahoma State University, Stillwater, OK 74078, USA

1 Introduction

Training neural networks is challenging because the objective function being minimized contains numerous local minima. Therefore, some have used so called global search algorithms to train neural networks rather than the more traditional local search algorithms. Global optimization algorithms are a class of algorithms that seek to avoid getting trapped in local minima.

Global search algorithms can be divided into two broad categories, stochastic and deterministic. Deterministic global algorithms are not investigated here because they are too slow for problems with more than a few parameters. Instead, several stochastic global algorithms are investigated. Some examples of stochastic search algorithms are simulated annealing and various evolutionary algorithms such as genetic algorithms, evolutionary strategies, and evolutionary programming [1].

Simulated annealing was used by Porto et al. [2], Sexton et al. [3], and Ludermir [4] to train neural networks. Genetic algorithms are the most common evolutionary algorithms used to train neural networks, especially in recent applied work [5–7]. Some examples of using evolutionary programming or evolutionary strategies are Binner et al. [8], Porto et al. [2], and Fisher et al. [9]. Evolutionary algorithms have also been used to evolve the architecture of neural networks as well as the weights [10]. However, this study is only concerned with estimating the weights of neural networks using a fixed architecture. That is, a neural network with a fixed number of hidden layers and hidden neurons.

When an objective function has many local minima, a global search algorithm can potentially find a lower minimum than a local algorithm. But since local search techniques are much faster, restarts of a local search algorithm with random starting values are an alternative to using global algorithms. Many studies have sought to compare local and global search algorithms for training neural networks. However, the results reported in the literature are mixed. From an experimental evaluation standpoint, many of these studies lack rigor. Many of the studies looked at the performance of the algorithms on only one or two data sets and few of the studies compared results across different global optimization routines. In addition, the majority of the data sets were classification problems or fitting a function of a single input variable [11–13]. Less attention has been paid to function approximation problems. This study provides a rigorous comparison of nine global algorithms against an efficient local optimization routine across six data sets in a function approximation context. Since only function approximation problems are studied, the conclusions may not extend to classification problems. The stochastic global algorithms are: two simulated annealing algorithms, one simple random stochastic algorithm, one genetic algorithm and five evolutionary strategy algorithms. Depending upon the size of the neural network, the neural networks are estimated with either 250 or 500 random restarts of each algorithm. The distributions of the final objective function values from each of these restarts are then compared against each other. Speed is measured as the time taken to solve the 250 or 500 optimizations.

2 Stochastic Global Optimization

Local search techniques use the gradient of the objective function with respect to the model parameters to guide the search. The search will generally move downhill in the search space from its starting point towards the nearest minimum. Stochastic global optimization algorithms are not restricted to taking only a downhill step. In addition, they may simultaneously explore many different regions of the search space. A concept at the core of stochastic global optimization is the generation of a trial point $\tilde{\theta}$ by taking a step from the current point θ as follows:

$$\tilde{\theta} = \theta + \mathbf{r}, \tag{1}$$

where $\mathbf{r}$ is a random vector drawn from some probability density function. The random move is often referred to as a mutation in the evolutionary algorithm literature.

In the beginning of a simulated annealing algorithm [1], large steps are taken from the current point according to (1). As the algorithm progresses, the variance of $\mathbf{r}$ decreases, steps are smaller, and the probability of accepting inferior trial points decreases. The algorithm begins to focus on the most promising areas of the parameter space. Eventually, the hope is that it settles to a point that is close to the global minimum.

Genetic algorithms as optimization algorithms are based loosely on the concepts of selection and reproduction found in nature. As such, the literature draws much of its terminology from genetics and biology. Genetic algorithms require a set of potential or candidate solutions, called individuals, to the problem at hand. The candidate solutions or individuals are referred to collectively as a population. The individuals in a population are born, mate, and die. A typical number of individuals in a population would be between 20 and 100.

In some genetic algorithms, the individuals in the population are encoded as binary strings. However, for real-valued optimization problems, there are advantages to using a real-valued encoding. That is, leaving the individuals as floating point numbers as they are in simulated annealing [14,15]. Real-valued encoding is used in this research.

An iteration of a genetic algorithm is referred to as a generation. In each iteration or generation, the individuals are subject to the operations of mutation, crossover, and selection, which operate on the individuals to form the next generation of the population. The crossover operator is the distinguishing operator of genetic algorithms [16].

Crossover has traditionally been viewed as the main search operator with mutation being only a background operator. Crossover is the process by which "genetic" material from different individuals is combined to create a potentially superior offspring. Pairs of individuals are picked at random from the population to serve as parents. These parents are subjected to crossover to form offspring. A simple crossover operator is the one-point crossover scheme. Two parents $s = (s_1, \ldots, s_n)$ and $v = (v_1, \ldots, v_n)$ are drawn at random from the population. A random point $k \leq n$ is chosen and the elements above this point are swapped which produces the individuals $s' = (s_1, \ldots, s_{k-1}, s_k, v_{k+1}, \ldots, v_n)$ and $v' = (v_1, \ldots, v_{k-1}, v_k, s_{k+1}, \ldots, s_n)$. Many other crossover schemes have been used.

In the case of this research, an individual that produces a relatively low function value would have a high fitness value. A probability that is proportional to its fitness is assigned to each individual. Selection for the next generation's population is then based on this probability.

Similar to genetic algorithms, modern evolutionary strategy algorithms operate on a population of potential solutions. However, in contrast to genetic algorithms, mutation is the primary operator and recombination is only a background operator. One of the most common evolutionary strategies is the (μ, λ)-evolutionary strategy. The (μ, λ)-evolutionary strategy is so named because it generates λ offspring from μ parents, $\lambda > \mu \geq 1$. The λ offspring are generated by mutation. Some implementations of the (μ, λ)-evolutionary strategy also apply a recombination operator. The best μ individuals from the λ offspring are selected for inclusion in the next generation. The comma in (μ, λ) represents that only the offspring are available for selection by survival of the fittest to be included in the next iteration of the algorithm.

One of the defining characteristics of modern evolutionary strategies is their ability to evolve or self-adapt the variances and sometimes covariances of the mutations. Therefore, the individuals in the population are composed of the parameters being optimized, called object variables in the literature, and the variances and covariances, called strategy parameters, of a multivariate normal distribution for mutation.

3 Neural Network Architectures and Data

This study is restricted to training of the feedforward type of multilayer perceptron (MLP). The specific form used in this study is defined by:

$$f(\mathbf{x}_t, \theta) = \tilde{\mathbf{x}}_t' \phi + \sum_{j=1}^{p} \beta_j G(\tilde{\mathbf{x}}_t' \gamma_j) \tag{2}$$

where $\mathbf{x}_t$ is an $n \times 1$ vector of inputs or explanatory variables for observation t, $\tilde{\mathbf{x}}_t = (1, \mathbf{x}'_t)$, γ_j is an $(n+1) \times 1$ vector of weights connecting the inputs to hidden neuron j, $\theta = (\phi_0, \ldots, \phi_n, \beta_1, \ldots, \beta_p, \gamma_1', \ldots, \gamma_p')$ is the vector of model parameters or weights, p is the number of hidden neurons in the single hidden layer, and $G(\bullet)$ is the hidden layer activation function defined by:

$$G(z) = 1/[1 + \exp(-z)]. \tag{3}$$

Given a set of training data with T observations, the objective or cost function in this study is penalized least squares:

$$\begin{aligned} \min_{\theta \in \Theta} Q(\theta) &= \sum_{t=1}^{T} [y_t - f(x_t, \theta)]^2 + r_\phi \sum_{i=0}^{n} \phi_i^2 + r_\beta \sum_{j=1}^{p} \beta_j^2 \\ &\quad + r_\gamma \sum_{j=1}^{p} \sum_{i=0}^{n} \gamma_{ij}^2 \end{aligned} \tag{4}$$

where y_t is the dependent variable, sometimes called the target value, Θ is the space of feasible weights or model parameters, and $r_\phi,\ r_\beta,$ and r_γ are weight decay constants. The weight decay constants penalize large weight values and were employed in Franses and van Dijk [17]. Following Franses and van Dijk [17] the weight decay parameters are set equal to $r_\phi = .01$, and $r_B = r_\gamma = .0001$.

The six training data sets used in this study were taken from Franses and van Dijk [17] and are detailed in the following sections. These data sets are primarily economic time series. Economics provides a rich source of function approximation problems [18]. Table 1 summarizes the six neural network models.

3.1 Bilinear

Data with the bilinear model are generated by

$$y_t = \beta y_{t-2} \varepsilon_{t-1} + \varepsilon_t \tag{5}$$

with $\beta = 0.6$ in this study. The series is generated by setting $y_{-1} = y_0 = 0$ and drawing ε_t from a standard normal distribution. A total of 350 observations are generated with the first 100 discarded leaving 250 observations for the training set. Granger and Andersen [19]

Table 1 Summary of training data sets and neural network models

Data Set	Obs	NN architecture	# parameters
Bilinear	250	2-3-1, dc; logistic-identity	15
DAX	360	4-4-1, dc; logistic-identity	29
JYUS	364	2-3-1, dc; logistic-identity	15
JYUSTTR	326	2-3-1, dc; logistic-identity	15
Flare	533	22-8-3; logistic-identity	211
MG	500	5-6-1; logistic-identity	43

Note: Column 2 is the number of observations in the data set. The neural network architecture is shown in column 3. The number of neurons in consecutive layers is enumerated as input-hidden-output, with a dc following indicating a direct connection between the input and output neurons. Column 3 also likewise enumerates the activation functions beginning with the first hidden layer. Column 4 is the number of weights for the model

showed that linear models will not be successful in modeling this series. Following Franses and van Dijk [17], two lags are used as inputs to the model. This training set is referred to as the Bilinear data set.

3.2 JYUS

The second time series is the weekly returns on the Japanese Yen-US dollar exchange rate. The weekly return is given by

$$r_t = \ln P_t - \ln P_{t-1}, \tag{6}$$

where r_t is the return for week t and P_t is the price level of the Japanese Yen-US dollar exchange rate for week t. The training data consists of 364 observations from January 1986 through December 1992. The relationship between the JYUS returns and its lags is nonlinear [17]. Two lags of (6) are used as inputs. This data set is referred to hereafter as the JYUS data set. Exchange rates as with many financial time series are often modeled with generalized autoregressive conditional heteroskedasticity [20]. It is not clear that the presence of heteroskedasticity should in any way affect the ability of the various algorithms to find the global optimum, but we cannot eliminate that possibility.

3.3 JYUSTTR

A second model to predict the Japanese yen-US dollar exchange rate is constructed by using technical trading rules as inputs to a neural network prediction model. Specifically, moving average trading signals are used. Following the notation of Franses and van Dijk [17], we define a moving average for period t as

$$m_t(\tau) = \frac{1}{\tau}\sum_{i=0}^{\tau-1} p_{t-i} \tag{7}$$

A moving average technical trading rule can be constructed from (7) as follows

$$s_t(\tau_1, \tau_2) = m_t(\tau_1) - m_t(\tau_2). \tag{8}$$

where $\tau_1 < \tau_2$. Equation (8) defines what is commonly called a dual moving average crossover. Following Franses and van Dijk [17], τ_1 and τ_2 are set to 1 and 40. The time periods are the same as with the JYUS data set. Three lags of (8) are used as inputs. This data set will hereafter be referred to as JYUSTTR.

3.4 DAX

The DAX time series data concerns the prediction of weekly absolute returns on the DAX stock index of the Frankfurt stock exchange. The weekly absolute returns are similarly defined as in (6) except that the absolute value is taken. This task is given to be harder than predicting just return levels. Franses and van Dijk [17] showed evidence of nonlinearity between this series and its lags. The time period for this data is 1986–1992. Two lags are used as inputs and three hidden neurons are used.

3.5 Mackey-Glass

This study uses a discrete version of the Mackey-Glass equation as used in Gallant and White [21]:

$$g^*(x_{t-5}, x_{t-1}) = x_{t-1} + 10.5 \cdot \left[\frac{0.2 \cdot x_{t-5}}{1 + x_{t-5}^{10}} - 0.1 \cdot x_{t-1} \right]. \tag{9}$$

This series is said to be qualitatively like financial market data. The series can exhibit long stretches of volatile data of apparently random duration. The Mackey-Glass data was generated from (9) with starting values of $x_0 = 1.6$ and $x_i = 0, \quad i = -4, \ldots, -1$. There were 1000 observations generated with the first 500 discarded leaving 500 observations for training data. The neural network model has five inputs consisting of five lags of the Mackey-Glass (MG) series. As can be seen from (9), only lags $t-1$ and $t-5$ are necessary to approximate this series. However, in most actual applications of neural networks, the true dimension of the problem is unknown. Therefore, superfluous inputs are commonly part of neural network modeling. The neural network model has one hidden layer with six neurons, logistic activation functions for the hidden layer neurons and an identity transfer function for the output neuron.

3.6 Flare

The Flare data are solar-flare data. The objective is to predict the number of small, medium, and large size flares that will happen during the next 24-h period in a fixed active region of the suns surface. There are three dependent variables in the data set, one each to predict the number of small, medium, and large solar flares. There are 22 inputs describing the type and history of the active region and the previous flare activity. The first 533 observations from the data file flare1.dt are used for training. Based upon the training and prediction results on this data set from Prechelt [22], a network with eight neurons in a single hidden layer with the logistic transfer function is chosen and a identity activation function in the output layer. The scaling of the data is left as it is in the original file. This leaves all the input variables scaled from 0 to 1 and. All the outputs have minimum values of 0 with maximum values of .75, .375, and 1.00.

4 Optimization Algorithms

Stochastic global algorithms are theoretically good at widely exploring the potential solution space. However, they are slow at finding the local maximum once a promising area of the solution space is found. Therefore, it is common to combine a local algorithm with a global algorithm by using the weights obtained from the global algorithm as starting values for the local routine [23,24]. This two-phase approach has been used for training neural networks [25,26]. This research uses a two-phase approach by using the local search routine used in this study with each global algorithm. There is no widely accepted criterion on when to switch from the global algorithm to the local algorithm. Since the switch needs to occur before a local optimum is reached, convergence criteria that are used for local routines, such as the magnitude of the gradient, are not appropriate. Therefore, for simplicity and so that all algorithms would be treated in the same way, the global routines are run for 100,000 function evaluations. Speed could have been increased by picking a number smaller than 100,000 and accuracy could have been increased by picking a larger number. Thus, the choice of 100,000 represents a single point in the tradeoff between accuracy and speed. After 100,000 function evaluations is reached, the local routine then takes over and is run to convergence. Without using a two-phase approach the performance of the global methods was much worse than the local algorithm. The local and global optimization algorithms along with their abbreviations are enumerated below.

4.1 LO

The two local optimization algorithms used in this study are the quasi-Newton routine DUMING and the conjugate-gradient routine DUMCGG from the IMSL subroutine libraries IMSL Math/Library [27]. The quasi-Newton routine is used on the Bilinear, DAY, JYUS, JUUSTTR, and Mackey-Glass training problems. The conjugate-gradient routine is used on the much larger Flare training problem. The conjugate-gradient routine does not require calculation or storage of the BFGS approximation to the Hessian. For the DUMING routine, the maximum number of iterations is set to 20,000 and the maximum number of function and gradient calculations is set to 30,000. For the DUMCGG routine, the maximum number of function evaluations is set to 60,000. All other user definable parameters for the DUMING and DUMCGG routines, including the gradient and step size based convergence criteria, are set to their default.

4.2 NNGA

The NNGA algorithm is a genetic algorithm that uses the neural network specific crossover operator proposed by van Rooij et al. [28]. Mutation for the NNGA is accomplished with a normal distribution. The standard roulette selection mechanism is used for the selection operator. The NNGA uses a generational replacement scheme whereby the entire population is replaced in each generation. The replacement mechanism is implemented with elitism. The best performing chromosome is retained and replaces a randomly selected individual in the next generation. Following van Rooij et al. [28], the population size is set at 50. The bias of the fitness normalization, which maintains constant selective pressure, is set experimentally. The standard deviation of the mutation and the probability of mutation and crossover are also set experimentally.

4.3 EVOL

This algorithm is an evolutionary strategy taken from Schwefel [29] who refers to the algorithm as EVOL. In evolutionary strategy notation, the algorithm is referred to as a (1+1)-evolutionary strategy. The algorithm employs Gaussian mutation of the model parameters. The FORTRAN code included with Schwefel [29] was used. The code was modified to suppress the default convergence so that all algorithms were evaluated in the same fashion using the number of function evaluations.

4.4 KORR1, KORR2, KORR3, and KORR4

The KORR1, KORR2, KORR3, and KORR4 algorithms are variations of the KORR evolutionary strategy algorithm taken from Schwefel [29]. The KORR algorithm is a multimembered (μ, λ)-evolutionary strategy. As with the EVOL algorithm, the FORTRAN coding from Schwefel [29] was used. Similar to EVOL, the code was modified to replace the default convergence criterion with one based on the number of function evaluations used. For all four evolutionary algorithms, the covariance terms for mutation are set to zero and the number of parents and descendents is set to 10 and 60 respectively. The KORR1 algorithm utilized no recombination. The KORR2 algorithm is similar to KORR1 except intermediary recombination is used for evolution of the object variables or neural network weights. The KORR3 algorithm adds intermediary recombination of the step sizes of mutations to KORR1. KORR4 uses intermediary recombination to evolve both the object variables and the standard deviations of mutation.

4.5 SA1 and SA2

The SA1 algorithm is a Boltzmann annealing version of simulated annealing. This is sometimes referred to as classic simulated annealing [30]. The SA2 algorithm is a fast simulated-annealing algorithm [30]. The next section discusses the procedure for picking the beginning temperature and standard deviation.

4.6 SW

The SW algorithm is the Solis and Wets [31] algorithm proposed by Solis and Wets [31] and used in Baba et al [32]. The SW algorithm is sometimes used in combination with genetic algorithms [11].

5 Global Optimization Algorithm Parameters

Some parameters of the various stochastic global algorithms must be chosen well in order for these algorithms to perform well. Often parameters are chosen on an ad hoc basis. Much effort was expended to choose good parameters for these algorithms. The procedure used to pick a good set of algorithm parameters and to run simulation with can best be described as a two-stage procedure. In the first stage, a large number of combinations of parameters for each algorithm were investigated. For each combination of algorithm parameters, a limited number of restarts or estimations were performed. For the larger networks on the Flare and Mackey-Glass data sets, five restarts were run for each configuration of algorithm parameters. For the other data sets, 10 restarts were run. The specific parameter configuration with the lowest average objective function value computed across the restarts was chosen as the "winning" configuration. In the second stage, the "winning" configuration of algorithm parameters was used in running full-scale simulation of either 250 or 500 restarts depending upon the size of the network. The results from the full-scale simulations with the "winning" algorithm parameter configurations are presented in the next section.

As an example, for the simulated annealing algorithms and Bilinear data set, 10 restarts with each of 72 combinations of the algorithm parameters were tried. Full-scale simulations with the “winning” configuration of algorithm parameters from these restarts were used to run 500 restarts, the results of which are presented in the next section. Hypothesis tests showed a statistical difference in performance between different parameter combinations, which means the two-stage procedure to choose algorithm parameters gives a small increase in accuracy to the global algorithms. Therefore, it could be argued that the procedure gives an unfair advantage to the global algorithms. However, if the local optimization routine outperforms or is competitive with the global routines, this provides further evidence that random restarts with a local search algorithm is competitive with many global algorithms.

6 Results

The global algorithms marginally outperformed the local routine in most cases. However, in some cases the local search routine outperformed one or more of the global routines. With respect to the minimum value obtained across the restarts, the local routine obtained a solution that was equal or very close, and in some cases superior, to the minima obtained by the global algorithms.

Figures 1–6 show the objective function values using boxplots for each of the optimization routines and training data sets. As Fig. 6 shows, the NNGA algorithm performed significantly worse than all other algorithms on the Mackey-Glass training data. It can be seen from Figs. 1–6 that the global algorithms provide only marginally more probability, if any, of obtaining a solution that is in the lower end or left-hand side of the distribution of possible solutions. Also, all of the algorithms have some very poor solutions. Thus, using a single set of starting values with any of the algorithms could lead to solutions far from the global optimum.

Table 2 shows the computing time required for the global algorithms relative to the local optimization algorithm. For example, 145 times as many restarts could be performed with the local optimization routine as the NNGA algorithm with the Bilinear data. For the two larger problems, the time advantage of the local algorithm was not as great. With a fixed amount of computer time, many more restarts could be performed with the local routine than with the global algorithms. Therefore, given an equal amount of computational resources, considering the results in Table 2, the local search algorithms are superior to the nine global search algorithms tested. Furthermore, even ignoring computational time, there is no single global search algorithm that consistently or substantially outperformed all others.

7 Conclusions

At least with respect to the specific algorithms studied, the results provide little evidence to show that a global algorithm should be used over a more traditional local optimization routine for training neural networks. In fact, all of the global algorithm results reported use the global method parameters as starting values in a local optimization routine because otherwise the global methods would not be competitive. The results strictly apply only to the estimation methods and problems considered. Only function approximation problems are considered and by necessity the problems considered were smaller than many real-world problems. Also, note that penalized least squares was used rather than least squares. Further, the stochastic global algorithms were ended after 100,000 function evaluations and so the results represent a single point in the continuum that these algorithms provide in the tradeoff between

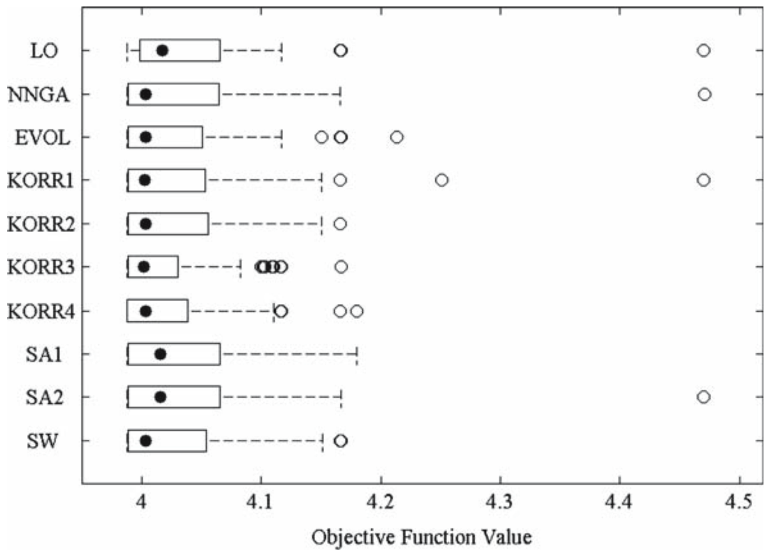

Fig. 1 Boxplot of objective function values from random restarts of different optimization algorithms for neural network training on the bilinear training data. The boxplots indicate the median, upper and lower quartiles, upper and lower adjacent values, and outside values. In the box plot, the solid dot indicates the median and the right and left ends of the box are the upper and lower quartiles. The vertical lines or whiskers outside the box mark the highest (lowest) data points within a range defined by the upper (lower) quartile + (−) 1.5 times the interquartile range. Any values outside of the whiskers are considered outside values and are plotted by open circles

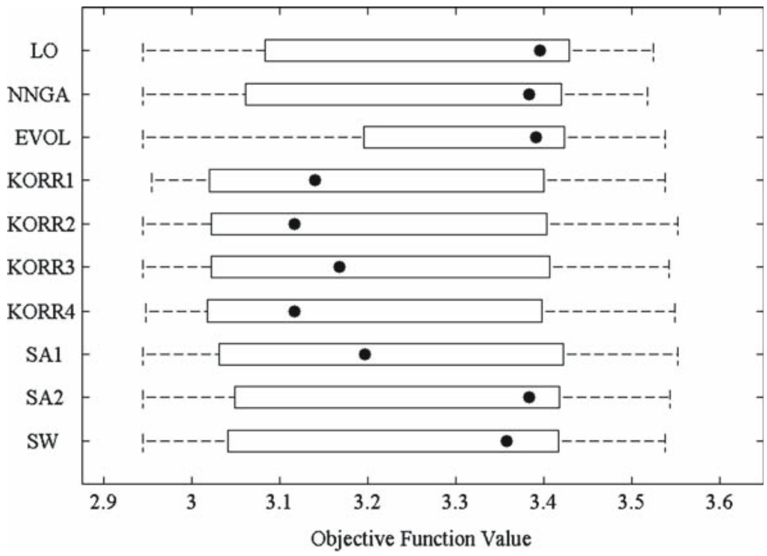

Fig. 2 Boxplot of objective function values from random restarts of different optimization algorithms for neural network training on the DAX training data. See Fig. 1 for a more detailed explanation of the information in this figure

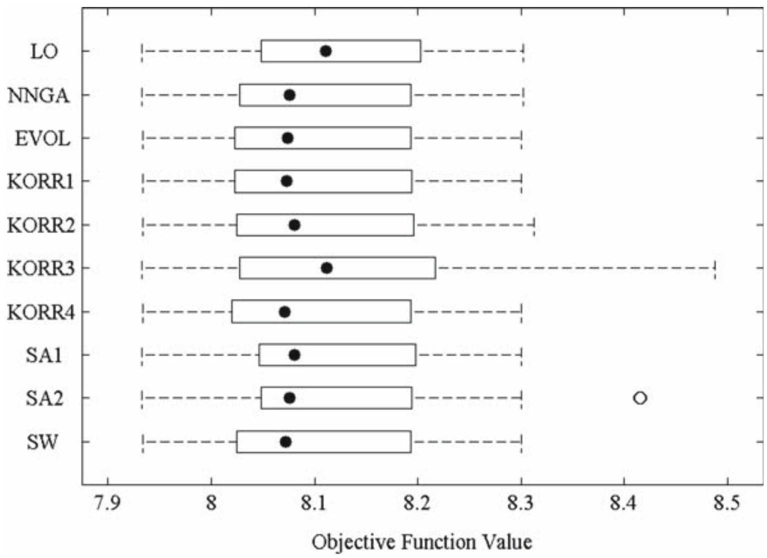

Fig. 3 Boxplot of objective function values from random restarts of different optimization algorithms for neural network training on the JYUS training data. See Fig. 1 for a more detailed explanation of the information in this figure

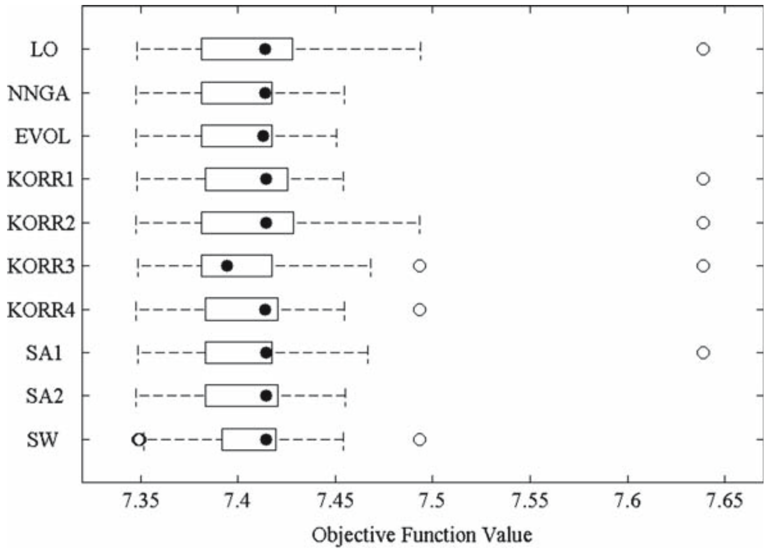

Fig. 4 Boxplot of objective function values from random restarts of different optimization algorithms for neural network training on the JYUSTTR training data. See Fig. 1 for a more detailed explanation of the information in this figure

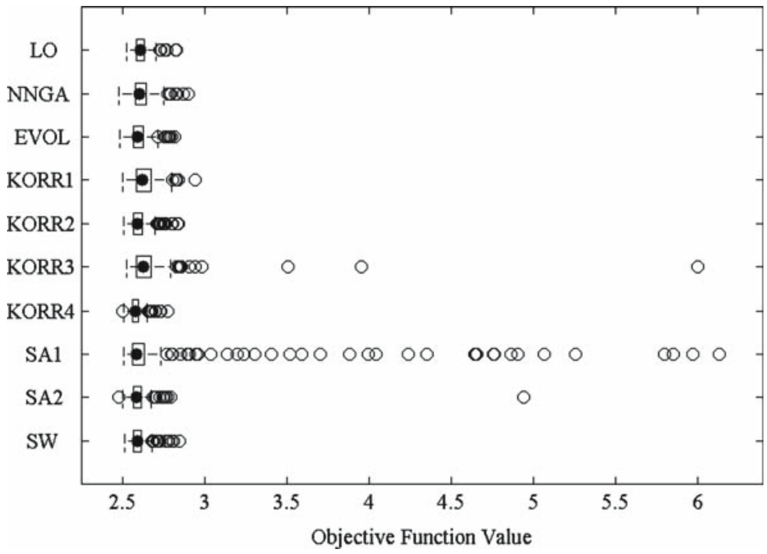

Fig. 5 Boxplot of objective function values from random restarts of different optimization algorithms for neural network training on the flare training data. See Fig. 1 for a more detailed explanation of the information in this figure

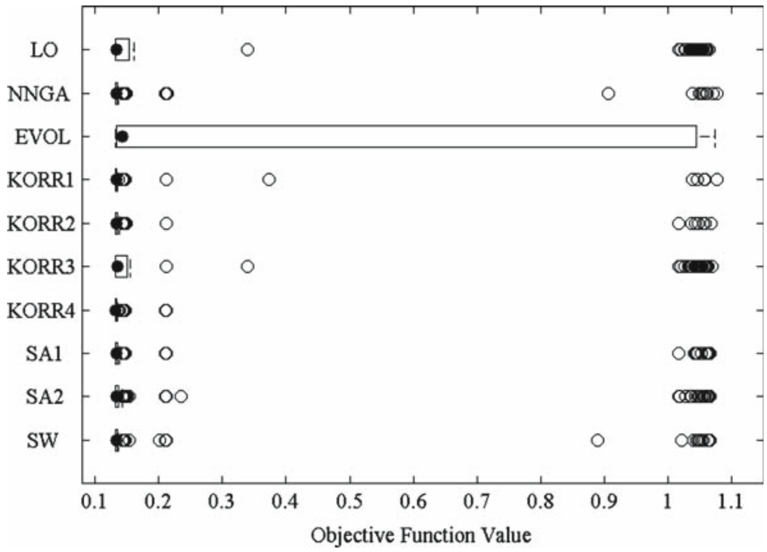

Fig. 6 Boxplot of objective function values from random restarts of different optimization algorithms for neural network training on the Mackey-Glass training data. See Fig. 1 for a more detailed explanation of the information in this figure

Table 2 Ratio of training times for global optimization algorithms in comparison to local optimization algorithms

Data Set	NNGA	EVOL	KORR1	KORR2	KORR3	KORR4	SA1	SA2	SW
Bilinear	145	174	175	175	175	175	174	185	159
DAX	48	59	61	60	60	64	60	63	54
JYUS	93	111	106	106	106	106	113	119	93
JYUSTTR	58	68	66	66	67	66	68	70	56
Flare	4	4	4	4	5	4	4	4	4
Mackey-Glass	12	14	14	13	14	13	15	14	13
Average	60.00	71.67	71.00	70.67	71.17	71.33	72.33	75.83	63.17

Note: The numbers indicate the ratio of the average training time for the global optimization routine divided by the average training time for the local optimization routine. For example, for the DAX neural network model, the NNGA took on average 48 times longer to train then the local optimization routine. The training times are averaged across all restarts

accuracy and computational time. There may be problems where global optimization methods are superior. However, even ignoring computational time, the stochastic global algorithms for training neural networks did little better than using a local algorithm alone. The results clearly indicate that neural networks should not be estimated from a single set of starting values whether a global or local optimization method is used.

References

1. Uryasev S, Pardalos PM (eds) (2001) Stochastic optimization: algorithms and applications. Kluwer Academic Publishers, Netherlands,
2. Porto VW, Fogel DB, Fogel LJ (1995) Alternative neural network training models. IEEE Expert 16–22
3. Sexton RS, Dorsey RE, Johnson JD (1999) Beyond backpropagation: using simulated annealing for training neural networks. J End User Comput 11:3–10
4. Ludermir TB (2003) Neural networks for odor recognition in artificial noses. Proceedings of the International Joint Conference on Neural Networks 1:143–148
5. Mirmirani S, Li HC (2004) Gold price, neural networks, and genetic algorithm. Comput Econ 23:193–200
6. Matilla-Garcia M, Arguello C (2005) A hybrid approach based on neural networks and genetic algorithms to study the profitability in the Spanish stock market. Appl Econ Lett 12:303–308
7. Zhao Z (2006) Steel columns under fire—A neural network-based strength model. Adv Eng Sof 32:97–105
8. Binner JM, Graham K, Gazely A (2004) Co-evolving neural networks with evolutionary strategies. A new application to Divisia money. Adv Econom 19:127–143
9. Fisher MM, Hlaváčková-Schindler K, Reismann M (1999) A global search procedure estimation in neural spatial interaction modeling. Pap Reg Sci 78:119–34
10. Maniezzo V (1994) Genetic evolution of the topology and weight distribution of neural networks. IEEE T Neural Networ 5(1):39–53
11. Plagianakos VP, Magoulas GD, Vrahatis MN (2001) Learning in multilayer perceptrons using global optimization strategies. Nonlinear Anal 47:3431–3436
12. Georgieva A, Jordanov I (2006) Supervised neural network training with a hybrid global optimization technique. 2006 International Joint Conference on Neural Networks, Vancouver, BC, Canada, July
13. Ye H, Lin Z (2003) Global optimization of neural network weights using subenergy tunneling functions and ripple search. Circuits and Systems, 2003, Isacs '03. Proceedings of the 2003 International Symposium on, vol. 5, Issue 25–28, May 2003, pp V-725–V-728
14. Michalewicz Z (1996) Genetic algorithms + data structures = evolution programs 3rd rev. and extended ed., Springer-Verlag, Berlin
15. Syswerda G (1991) Schedule optimization using genetic algorithms, In: Davis L (ed) Handbook of genetic algorithms. Van Nostrand Reinhold, New York, pp 332–349
16. Davis L (ed) (1991) Handbook of genetic algorithms. Van Nostrand Reinhold, New York

17. Franses PH, van Dijk D (2000) Nonlinear time series models in empirical finance. Cambridge University Press, Cambridge
18. Hamm L, Brorsen BW (2000) Trading futures markets based on signals from a neural network. Appl Econ Lett 7:137–140
19. Granger CW, Andersen AP (1978) An introduction to bilinear time series models. Vandenhoek and Ruprecht, Gottingen
20. Yang S-R, Brorsen BW (1995) Nonlinear dynamics of daily foreign exchange rates. Adv Quant Anal Finance Account 3:111–130
21. Gallant A, White H (1992) On learning the derivatives of an unknown mapping with multilayer feedforward networks. Neural Networks 5:129–138
22. Prechelt L (1994) Proben1—A set of neural network benchmark problems and benchmarking rules. Technical Report 21/94, Universitat Karlsrude. http://page.mi.fu-berlin.de/~prechelt/Biblio/1994-21.pdf
23. Pardalos PM, Romeijn E (eds) (2002) Handbook of global optimization vol 2: Heuristic approaches. Dordrecht, Netherlands,
24. Boender CGE, Romeijn HE (1995) Stochastic methods. In: Horst R, Pardalos PM (eds) Handbook of global optimization. Kluwer Academic Publishers, Netherlands, pp 829–869
25. Skinner AJ, Broughton JQ (1995) Neural networks in computational materials science: training algorithms. model Simul Mater Sci 3: 371–390
26. Yan W, Zhu Z, Hu R (1997) A hybrid genetic/bp algorithm and its application for radar target classification. Proceedings of the 1997 IEEE National Aerospace and Electronics Conference (NAECON) 2:981–984
27. IMSL Math/Library version 3.0 (1997) Visual Numerics, Houston, TX
28. van Rooij AJF, Jain LC, Johnson RP (1996) Neural network training using genetic algorithms. World Scientific Publishing Co., Singapore
29. Schwefel HP (1995) Evolution and optimum seeking. Wiley, New York
30. Szu H, Hartley R (1987) Fast simulated annealing. Phys Lett A 122:157–162
31. Solis FJ, Wets JB (1981) Minimization by random search techniques. Math Oper Res 6:19–30
32. Baba N, Mogami Y, Kohzaki M, Shiraihi Y, Yoshida Y (1994) A hybrid algorithm for finding the global minimum of error function of neural networks and its applications. Neural Networks 7:1253–1265