# Global Optimization Methods

by

**Lonnie Hamm**

**B. Wade Brorsen**

Correspondence Author: B. Wade Brorsen
Department of Agricultural Economics
Oklahoma State University
414 Agricultural Hall
Stillwater, Oklahoma 74078-6026
Tel: (405) 744-6836
Fax: (405) 744-8210
E-mail: brorsen@okstate.edu

*Paper presented at the Western Agricultural Economics Association Annual Meetings, Long Beach, CA, July, 2002*

Lonnie Hamm is a Ph.D. candidate, Department of Agricultural Economics, Oklahoma State University, Stillwater, Oklahoma. B. Wade Brorsen, is a regents professor and Jean & Patsy Neustadt chair, Department of Agricultural Economics, Oklahoma State University, Stillwater, Oklahoma

b

# Global Optimization Methods

**Abstract** – Training a neural network is a difficult optimization problem because of numerous local minimums. 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 relative efficiency of a local search algorithm to 9 stochastic global algorithms. 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.

**Keywords** – Evolutionary algorithms, neural networks, simulated annealing, stochastic global optimization

## 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 minimums.

Global search algorithms can be divided into two broad categories, stochastic and deterministic. Deterministic global algorithms (eg. grid search) 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.

Simulated annealing was used by Porto, Fogel, and Fogel; Sexton, Dorsey, and Johnson; and Day and Camporese to train neural networks. Genetic algorithms are the most common evolutionary algorithm used to train neural networks (see Chen and O'Connell; and Yao). Some examples of using evolutionary programming or evolutionary strategies are Porto, Fogel, and Fogel; Scholz; and Wienholt. Evolutionary algorithms have also been used to evolve the architecture of neural networks as well as the weights (Maniezzo; Harp, Samad, and Guha,). However, this study is only concerned with estimating the weights of neural networks with a fixed architecture. That is, a neural network with a fixed number of hidden layers and hidden neurons.

Global search algorithms have a potential advantage in accuracy, but local search techniques are much faster. Therefore, 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. Little 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 four data sets in a function approximation context. The stochastic global algorithms are, 2 simulated annealing algorithms, 1 simple random stochastic algorithm, 1 genetic algorithm and 5 evolutionary strategy algorithms. 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.

## 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 proceed 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 $\theta$ as follows:

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

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, large steps are taken from the current point according to (1). As the algorithm progresses, smaller and smaller steps are taken. The degree of perturbation caused by the random vector $\mathbf{r}$ in (1) is proportional to a parameter in the simulated annealing algorithm called the temperature. As the algorithm progresses, the temperature cools. A normal distribution is commonly used to generate the random steps. A random step can be accepted or rejected as the starting point for the next iteration. Most simulated annealing algorithms automatically accept a step that results in an improvement of the objective function. However, a feature of simulated annealing is that the algorithm can also accept an inferior step that results in a worsening of the objective function.

The acceptance of an inferior point is based on a probability generated by a function $g(\Delta Q, T)$ where $\Delta Q$ is the change in the cost function and *T* is the temperature. The function $g(\Delta Q, T)$ is an increasing function of $\Delta Q$ and a decreasing function of *T*. The larger the increase in the objective function, the lower the probability of accepting the point and the lower the temperature, the lower the probability of accepting the point. The value of $g(\Delta Q, T)$ is compared to a uniform random number $s$ in [0,1]. If $s < g(\Delta Q, T)$, the inferior point is accepted as the starting point for the next iteration. In the beginning, the algorithm randomly bounces around by taking large steps and accepting a high percentage of inferior trial points. As the algorithm progresses, the temperature 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, we hope 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. The driving force behind the operation of genetic algorithms, as it is in evolution of a species, is survival of the fittest. 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. (See Michalewicz; Wright; and Syswerda for more details.) 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 mutation operator is as that given in (1) and is applied to each individual in the population. The crossover operator is next applied to each individual. The crossover operator is the distinguishing operator of genetic algorithms (Davis). 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 are drawn at random from the population. A random point in the individuals is chosen after which all elements between the two parents are swapped. Many other crossover schemes are have been used.

The last operator to be employed in each generation is the selection operator. The selection operator forms the population for the next generation by selecting those individuals that are most "fit". The fitness is based on how well the individual solves the problem at hand. In the case of this research, an individual that produces a relatively low cost function value would have a high fitness value. A probability value 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 multi-membered evolutionary strategies is the $(\mu,\lambda)$ -evolutionary strategy. The

$(\mu, \lambda)$-evolutionary strategy is so named because it generates $\lambda$ offspring from $\mu$ parents, $\lambda > \mu \geq 1$. The comma in $(\mu, \lambda)$ represents the fact 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.

**Simulation Details**

The simulations use either 250 or 500 random restarts of each algorithm to compare the final objective function values for each algorithm. For the two largest neural network models, 250 random restarts are used to conserve computational resources.

**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:

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

where $\mathbf{x}_t$ is an $n \times 1$ vector of inputs or explanatory variables for observation *t*, $\tilde{\mathbf{x}}_t = (1, \mathbf{x}_t')$, $\gamma_j$ 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:

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

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

(4) $$\min_{\theta\in\Theta} Q(\theta) = \sum_{t=1}^{T}[y_t - f(x_t,\theta)]^2 + r_\phi \sum_{i=0}^{n}\phi_i^2$$

$$+ r_\beta \sum_{j=1}^{p}\beta_j^2 + r_\gamma \sum_{j=1}^{p}\sum_{i=0}^{n}\gamma_{ij}^2$$

where $y_t$ is the dependent variable, sometimes called the target value, $\Theta$ is the space of feasible weights or model parameters, and $r_\phi$, $r_\beta$, and $r_\gamma$ are weight decay constants. The weight decay constants penalize large weight values and were employed in Franses and van Dijk. Following Franses and van Dijk the weight decay parameters are set equal to $r_\phi = .01,$ and $r_B = r_\gamma = .0001$. The number of hidden neurons for each of the four data sets is chosen based upon the results presented in Franses and van Dijk.

The four training data sets used in this study were taken from Franses and van Dijk and are detailed in the following sections. Table 1 summarizes the four neural network models.

1) *Bilinear*

Data with the bilinear model are generated by

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

with $\beta = 0.6$ in this study. The series is generated by setting $y_0 = y_{-1} = 0$ and drawing $\varepsilon_t$ from a Normal(0,1) distribution. A total of 350 observations are generated with the first 100 discarded leaving 250 observations for the training set. Granger and Andersen showed that linear models will not be successful in modeling this series. Following Franses and van Dijk, two lags are used as inputs to the model. This training set is referred to as the Bilinear data set.

2) *JYUS*

The exchange rate data are weekly returns on the Japanese

Yen-US dollar exchange rate given by

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

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 (Franses and van Dijk). Two lags of (6) are used as inputs. This data set is referred to hereafter as the JYUS data set.

3) *Mackey-Glass*

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

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

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 $x_i = 0, \; i = -1, \ldots, -4$. 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 problems is unknown. Therefore, superfluous inputs are commonly part of neural network modeling. The neural network model has one hidden layer with 6 neurons, logistic activation functions for the hidden layer neurons and an identity transfer function for the output neuron.

4) *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-hour period in a fixed active region of the suns surface. There are 3 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, a network with 8 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 flare1.dt. 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.

**Optimization algorithms**

Stochastic global algorithms are theoretically good at widely exploring the potential solution space. However, they are poor at honing in on a particular solution 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. This hybrid approach has been used for training neural networks (Skinner and Broughton; Yan, Zhu, and Hu). This research uses a hybrid approach by using the local search routine used in this study with each global algorithm. Since global algorithms are not very good at fine tuning a local minimum, convergence criterions that are used for local routines, such as the magnitude of the gradient, are not appropriate. Therefore, for simplicity, all of the global routines are run for 100000 function evaluations. The local routine then takes over and is run to convergence. The local and global optimization algorithms along with their abbreviations are enumerated below.

*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). The quasi-Newton routine is used on the Bilinear, JYUS, 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 30000.  For the DUMCGG routine, the maximum number of function evaluations is set to 60000.  All other user definable parameters for the DUMING and DUMCGG routines, including the gradient and step size based convergence criterions, are set to their default.

*2)  NNGA*

The NNGA algorithm is a genetic algorithm that uses the neural network specific crossover operator proposed by van Rooij, Jain, and Johnson.  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, Jain, and Johnson, 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.

*3)  EVOL*

This algorithm is an evolutionary strategy taken from Schwefel 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 was used.

*4)  KORR1, KORR2, KORR3 and KORR4*

The KORR1, KORR2, KORR3, and KORR4 algorithms are variations of the KORR evolutionary strategy algorithm taken from Schwefel. The KORR algorithm is a multimembered $(\mu, \lambda)$-evolutionary strategy. As with the EVOL algorithm, the FORTRAN coding from Schwefel was used. Similar to EVOL, the code was modified to suppress the default convergence criterion. Instead a criterion based upon the number of function evaluations was used. For all three 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.

*5) SA1and SA2*

The SA1 algorithm is a Boltzmann annealing version of simulated annealing. This is sometimes referred to as classic simulated annealing. See (Szu and Harley). Details of the procedure for picking the beginning temperature and standard deviation are discussed in the next section. The SA2 algorithm is a fast simulated-annealing algorithm (Szu and Harley). The next section discusses the procedure for picking the beginning temperature and standard deviation.

*6) SW*

The SW algorithm is the Solis and Wets Algorithm proposed by Solis and Wets and used in Baba, et al. and Baba.

**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. For each algorithm, a wide range of values was tried with a limited number of restarts of 25. For example, for the simulated annealing algorithms, 72 combinations of the algorithm parameters were tried. The best five of these were then chosen to run with 500 restarts. The results for that algorithm are then reported as the best one out of these five. This procedure gives an unfair advantage to the global algorithms. However, if the local optimization routine outperforms the global routines, this only further evidence that random restarts with a local search algorithm is competitive with many global algorithms.

**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 minimums obtained by the global algorithms.

Figures 1-4 show the objective function values using boxplots for each of the optimization routines and training data sets. As figure 4 shows, the NNGA algorithm performed significantly worse than all other algorithms on the Mackey-Glass training data. It can be seen from figures 1-4 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 9 global search algorithms tested. Furthermore, even ignoring computational time, there is no single algorithm that consistently or substantially outperformed all others.

## Conclusions

The results indicate that with respect to the specific algorithms 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. The results strictly apply only to the estimation methods and problems considered. There may be problems where global optimization methods are superior.

However, even ignoring computational time, there is still little evidence to support the use of stochastic global algorithms for training neural networks. We would suggest that stochastic global algorithms be used only as a last resort such as when poor scaling or nondifferentiability prevent using a local quasi-Newton algorithm.

**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 |
| JYUS | 364 | 2-3-1, dc; logistic-identity | 15 |
| Flare | 533 | 22-8-3; logistic-identity | 211 |
| MG | 500 | 5-6-1; logistic-identity | 43 |

Notes: 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.

**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 |
| JYUS | 93 | 111 | 106 | 106 | 106 | 106 | 113 | 119 | 93 |
| Flare | 4 | 4 | 4 | 4 | 5 | 4 | 4 | 4 | 4 |
| Mackey-Glass | 12 | 14 | 14 | 13 | 14 | 13 | 15 | 14 | 13 |
| Average: | 63.5 | 77.75 | 74.75 | 74.5 | 75 | 74.5 | 76.5 | 80.75 | 64.75 |

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.

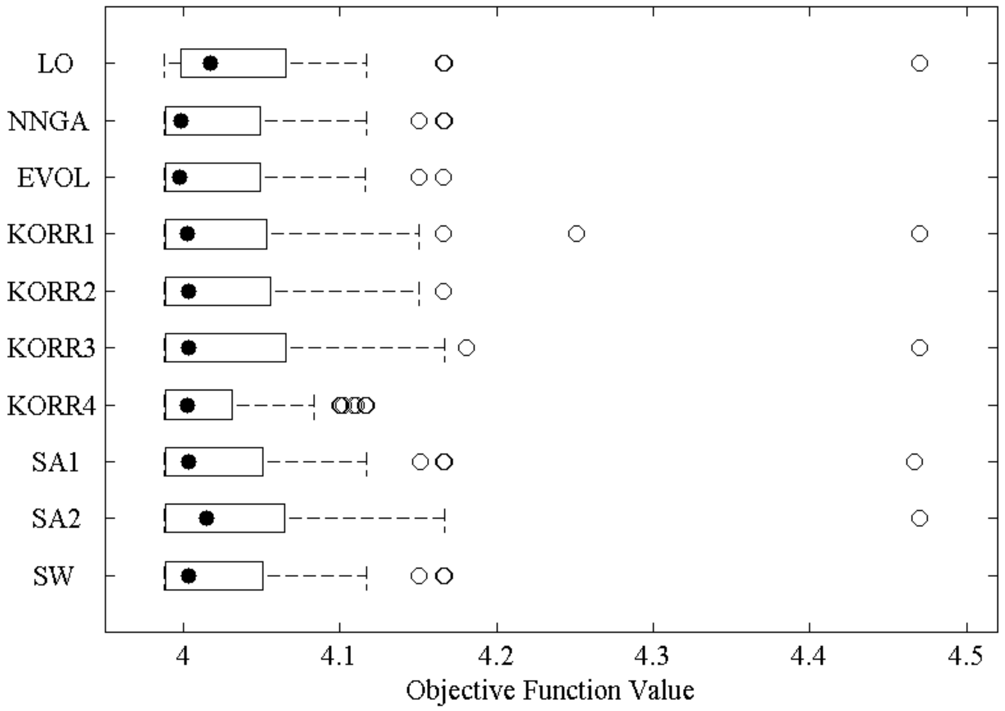

**Figure 1. Boxplot of objective function values from random restarts of different optimization algorithms for neural network training on the bilinear training data. The boxplots indicate the median, upper and lower quartiles, upper and lower adjacent values, and outside values. In the box plot, the 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.**

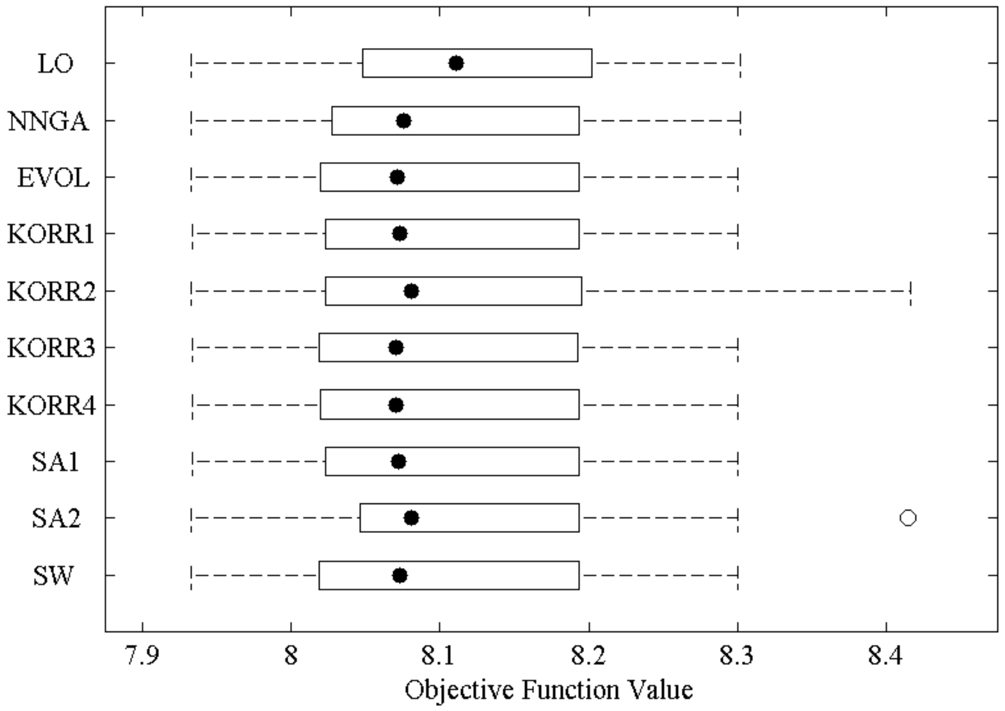

**Figure 2. Boxplot of objective function values from random restarts of different optimization algorithms for neural network training on the JYUS training data.**

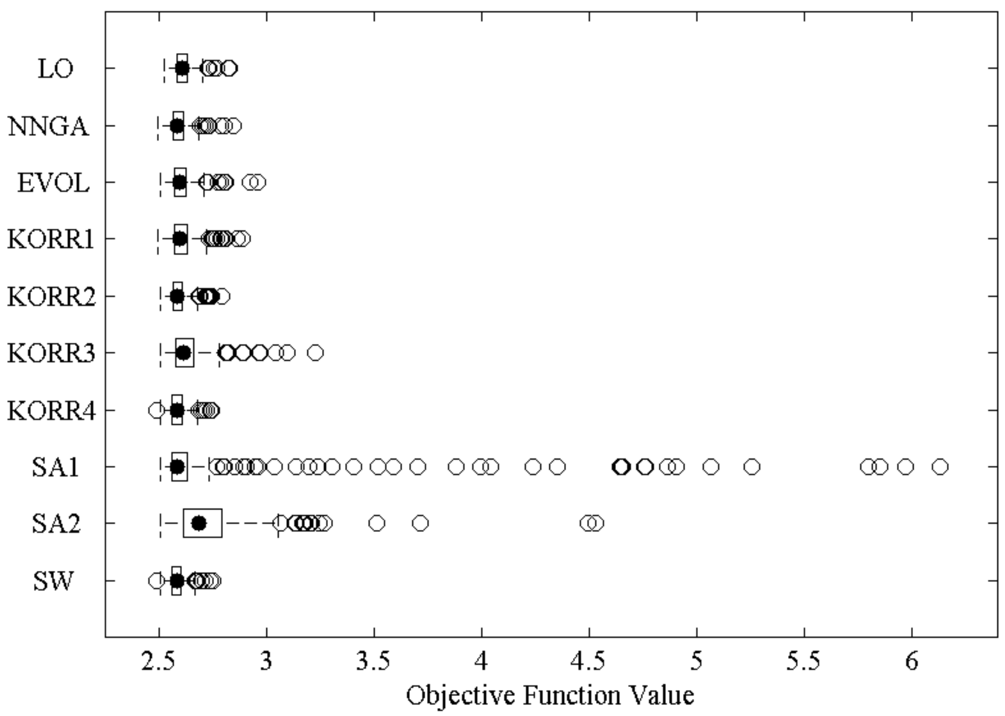

**Figure 3. Boxplot of objective function values from random restarts of different optimization algorithms for neural network training on the flare training data.**

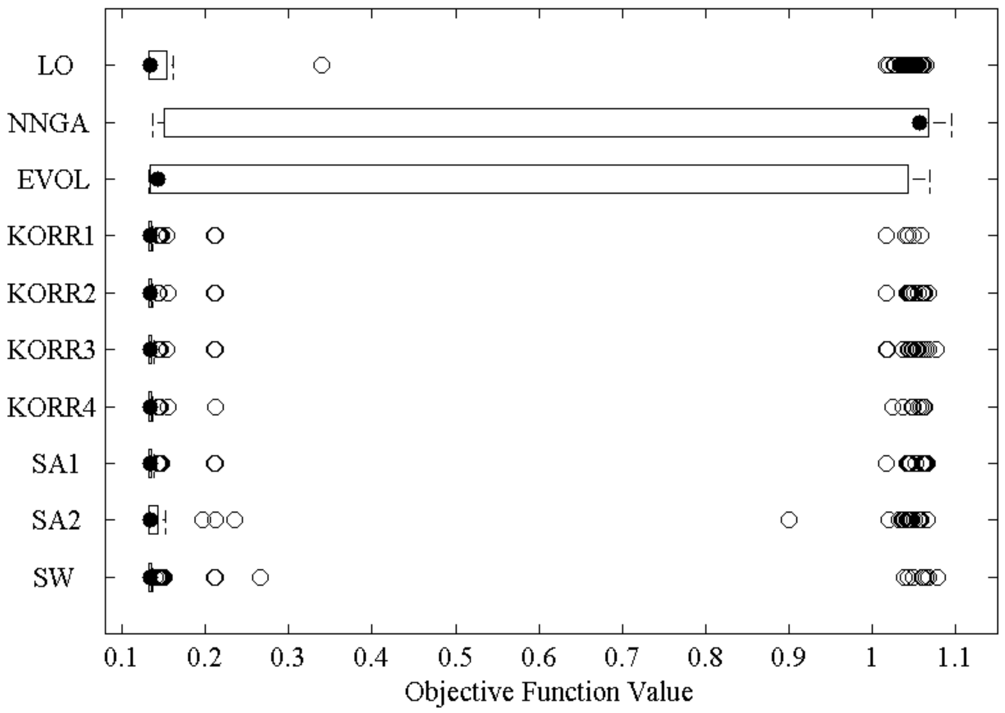

**Figure 4. Boxplot of objective function values from random restarts of different optimization algorithms for neural network training on the Mackey-glass training data.**

## References

Baba, N. “A New Approach for Finding the Global Minimum of Error Function of Neural Networks." *Neural Networks* 2(1989):367-373.

Baba, N. Yoshio Mogami, Motokazu Kohzaki, Ysuhiro Shiraihi, and Yutaka Yoshida. "A Hybrid Algorithm for Finding the Global Minimum of Error Function of Neural Networks and Its Applications." *Neural Networks* 7(1994):1253-1265.

Chen, and R.M. O'Connell. "Active Power Line Conditioner with a Neural Network Control." *IEEE Transactions on Industry Applications* 33(1997):1131-1136.

Davis, L. Ed. *Handbook of Genetic Algorithms.* New York: Van Nostrand Reinhold, 1991.

Day, S.P. and D.S. Camporese. "A Stochastic Training Technique for Feed-Forward Neural Networks." *Proceedings of the International Joint Conference on Neural Networks.* (undated).

Franses, P.H. and D. van Dijk. *Nonlinear Time Series Models in Empirical Finance*. Cambridge: Cambridge University Press, 2000.

Gallant, A. and H. White. “On Learning the Derivatives of an Unknown Mapping with Multilayer Feedforward Networks."*Neural Networks* 5(1992):129-138.

Granger, C.W. and A.P. Andersen. *An Introduction to Bilinear Time Series Models*. Gottingen: Vandenhoek and Ruprecht, 1978.

Harp, S. T. Samad, and A. Guha. "Towards the Genetic Synthesis of Neural Networks." *Proceedings of the Third International Conference on Genetic Algorithms.* San Mateo, CA:Morgan Kaufmann, 1989, pp. 360-369.

IMSL Math/Library version 3.0. "Visual Numerics." Houston, TX, 1997.

Maniezzo, V. "Genetic Evolution of the Topology and Weight Distribution of Neural Networks." *IEEE Transactions on Neural Networks* 5(No. 1, 1994):39-53.

Michalewicz, Z. "Genetic Algorithms + Data Structures = Evolution Programs." 3rd rev. and extended ed., Berlin: Springer-Verlag, 1996.

Porto, V.W., D.B. Fogel, and L.J. Fogel, "Alternative Neural Network Training Models." *IEEE Expert.* (June, 1995):16-22.

Scholz, M. “A Learning Strategy for Neural Networks Based on a Modified Evolutionary Strategy." In Proceedings of Parallel Problem Solving from Nature, 1st Workshop, PPSN I, Dortmund Germany, October 1-3, 1990, pp. 314-318.

Schwefel, H.P. *Evolution and Optimum Seeking*. New York: Wiley, 1995.

Sexton, R.S., R.E. Dorsey, and J.D. Johnson. “Beyond Backpropagation: Using Simulated Annealing for Training Neural Networks." *Journal of End User Computing*, 11(1999):3-10.

Skinner, A.J., and J. Q. Broughton. "Neural Networks in Computational Materials Science: Training Algorithms." *Modelling Simulation Materials Science Engineering* 3(1995):371-390.

Solis, F.J., and J.B. Wets. “Minimization by Random Search Techniques." *Mathematics of Operations Research* 6(1981):19-30.

Syswerda, G. "Schedule Optimization Using Genetic Algorithms." In *Handbook of Genetic Algorithms*, L. Davis (ed) New York: Van Nostrand Reinhold, pp. 332-349, 1991.

Szu, H. and R. Hartley. "Fast Simulated Annealing." *Physics Letters A* 122(1987):157-162.

van Rooij, A.J.F., L.C. Jain, and R.P. Johnson. *Neural Network Training Using Genetic Algorithms.* Singapore: World Scientific Publishing Co. 1996.

Wienholt, W. “Minimizing the System Error in Feedforward Neural Networks with Evolution Strategy." In *Proceedings of the International Conference on Artificial Neural Networks* (Spring, 1993): 490-493.

Wright, A.H. "Genetic Algorithms for Real Parameter Optimization." *Foundations of Genetic Algorithms* G.J.E. Rawlins, Ed. San Mateo, CA: Morgan Kaufmann Publishers, pp. 205-218, 1991.

Yan, W. Z. Zhu, and R. Hu, "A Hybrid Genetic/BP Algorithm and Its Application for Radar Target Classification." in *Proceedings of the 1997 IEEE National Aerospace and Electronics Conference,* NAECON, vol. 2, pp. 981-984.

Yao, X., “Evolving Artificial Neural Networks." Proceedings of the IEEE, vol. 87, no. 9, 1999.